Sichere Kommunikation

Teilnehmer:

8 Schülerinnen und Schüler	Heinrich-Hertz-Gymnasium Herder-Gymnasium Käthe-Kollwitz-Gymnasium

Gruppenleiter:

Barbara Jung	Humboldt-Universität zu Berlin
Leon Ochmann	Humboldt-Universität zu Berlin

1. Codes

Im folgenden Abschnitt wird auf eines der Probleme digitaler Kommunikation, die potentielle Störung übertragener Daten, eingegangen. Durch ein störungsanfälliges Übertragungsmedium, beispielsweise nicht perfekt funktionierende Kabel, können Übertragungsfehler entstehen und somit Nachrichten stark verfälscht werden. Um eben dies zu verhindern, kommen fehlerkorrigierende Codierungsverfahren zum Einsatz. Diese benutzen Codewörter, mit Hilfe derer sich im besten Fall die eigentliche Nachricht aus den gestörten Daten rekonstruieren lässt.

Definition 1. Ein *Codewort* ist eine Kette von Zeichen, die für eine Nachricht steht und anstelle ihrer verschickt wird, in diesem Kontext aus Gründen der Vorbeugung von Unkenntlichkeitsmachung durch Störungen bei der Übertragung.

Bemerkung 1. Wir betrachten hier den Standardfall, nämlich Ketten aus Binärziffern.

Definition 2. Der *Abstand* zweier Codewörter ist die Anzahl ihrer sich unterscheidenden Positionen. Beispielsweise unterscheiden sich die Codewörter 1110 und 0010 an der ersten und zweiten Position und haben somit den Abstand 2.

Bemerkung 2. Eben dieser Abstand ist wichtig für die folgenden Codierungen, denn je größer der Abstand zwischen Codewörtern ist, desto mehr potentielle Fehler kann man in geschickt gewählten Codes korrigieren. Wenn sich zwei beliebige Codewörter nämlich an mindestens $2d+1$ Positionen unterscheiden, so können bei der Übertragung bis zu d Bits verfälscht werden. Das verfälschte Codewort hat zum ursprünglichen dann den Abstand d, zu allen anderen Codewörtern ist der Abstand mindestens $d+1$. Man kann also immer noch erkennen, welches das ursprünglich gesendete Codewort war.

1.1. Der Hamming-Code

Dieses Codierungsverfahren erweitert die zu sendenden Informationsbits um weitere sogenannte Prüfbits. Die Standardform des Hamming-Codes verwendet vier Informations- und drei Prüfbits. Die Prüfbits werden aus den Informationsbits ermittelt und dann mit diesen versendet. Damit kann man einen Fehler in der Nachricht erkennen und ihn auch direkt berichtigen. Der Abstand zweier Codewörter beträgt im Hamming-Code immer mindestens drei, dies ist also konsistent mit der vorangegangenen Bemerkung.

Im Hamming-Code mit 7 Bits bezeichen wir die 4 Informationsbits mit x_1, x_2, x_3, x_4 und die 3 Prüfbits mit p_1, p_2, p_3. Die Prüfbits errechnen sich aus den folgenden Formeln:

$$\begin{aligned} x_1 + x_2 + x_3 &= p_1 \\ x_1 + x_2 + x_4 &= p_2 \\ x_1 + x_3 + x_4 &= p_3. \end{aligned}$$

Dem aus den Informationsbits $x_1x_2x_3x_4 = 1110$ bestehenden Codewort werden also beispielsweise die Prüfbits $p_1p_2p_3 = 100$ angefügt. (Wir befinden uns hier im Binärsystem, rechnen also $1+1=0$.) Daraus leiten sich die folgenden drei Prüfgleichungen ab:

$$\begin{aligned} x_1 + x_2 + x_3 + p_1 &= 0 \\ x_1 + x_2 + x_4 + p_2 &= 0 \\ x_1 + x_3 + x_4 + p_3 &= 0. \end{aligned}$$

Diese Gleichungen gelten, wenn die Nachricht verschickt wird. Sind sie beim Ankommen der Nachricht immer noch erfüllt, so ist die Nachricht wahrscheinlich richtig übertragen worden (oder es sind mindestens zwei Fehler aufgetreten). Aufgrund der Struktur der Gleichungen kann man außerdem bei Fehlern erkennen, welches Bit gestört wurde.
Ist nämlich x_1 falsch, würde die rechte Seite jeder Gleichung falsch, also gleich 1 sein. Wäre x_2 falsch,

wären die ersten beiden Gleichungen falsch, weil x_2 exklusiv dort vorkommt. Analog gilt dies für x_3 und x_4 mit der ersten und dritten sowie den letzten beiden Gleichungen. Ist nur eine Gleichung nicht erfüllt, so ist das zu dieser Gleichung gehörige Prüfbit falsch. (Dies setzt natürlich voraus, dass in den sieben Bits nur ein Fehler existiert.) Dieses Fehlerkorrigierungsverfahren wird nun noch einmal an einem Beispiel verdeutlicht.

Beispiel 1. Die Informationsbits 1110 sollen verschickt werden. Die sieben Bits des Codeworts seien in der Reihenfolge $x_1x_2x_3x_4p_1p_2p_3$ geordnet. Wir nehmen nun an, das Informationsbit x_3 wird fehlerhaft übertragen, also

$$1110100 \longrightarrow 1100100.$$

Aus den Prüfgleichungen ergibt sich

$$\begin{aligned} 1+1+0+1&=1\\ 1+1+0+0&=0\\ 1+0+0+0&=1, \end{aligned}$$

wir erkennen also den Fehler bei x_3 und decodieren

$$1100100 \longrightarrow 1110100 \longrightarrow 1110.$$

Bemerkung 3. Man kann den Hamming-Code theoretisch beliebig erweitern, mit vier Prüfbits kann man beispielsweise bis zu elf Informationsbits fehlerkorrigierend codieren. Dabei bleibt der Mindestabstand der Codewörter allerdings immer noch 3, man kann also immer noch nur einen Fehler korrigieren. Dies ist bei sehr langen Codewörtern oft nicht genug.

1.2. Der Hadamard-Code

Wir werden uns nun mit den sogenannten Hadamard-Codes befassen. Diese sind Codes mit Codewörtern der Länge $n = 2^m$ ($m \in \mathbb{N}$), die einen relativ großen Abstand, nämlich $n/2$ zueinander haben. Dementsprechend kann der Hadamard-Code, im Gegensatz zum Hamming-Code, nicht nur einen, sondern $(n/4) - 1$ Fehler korrigieren. Aufgrund dieser Eigenschaft wird der Hadamard-Code bei stark fehlerhaften Kommunikationen verwendet. Beispielsweise kam er bei mehreren Marsmissionen zum Einsatz.

Definition 3. Wir *skalarmultiplizieren* zwei Vektoren miteinander, indem wir die einzelnen Komponenten miteinander multiplizieren und die Ergebnisse zusammenaddieren, sodass das Ergebnis am Ende eine einzelne Zahl ist, zum Beispiel

$$(2,4,1)\cdot(3,-2,3) = 2\cdot 3 + 4\cdot(-2) + 1\cdot 3 = 6 - 8 + 3 = 1.$$

Wir nennen das Ergebnis das *Skalarprodukt* der beiden Vektoren.

Definition 4. Eine *Hadamard-Matrix* ist eine Matrix der Größe $n = 2^m$, deren Zeilen sowie Spalten miteinander vektormultipliziert immer 0 ergeben. Wir erzeugen Hadamard-Matrizen induktiv, indem wir von der kleinsten Hadamard-Matrix ausgehen, nämlich

$$H_2 = \begin{pmatrix} +1 & +1 \\ +1 & -1 \end{pmatrix}.$$

Das Skalarprodukt der zwei Zeilen/Spalten dieser Matrix ist tatsächlich $1\cdot 1 + 1\cdot(-1) = 0$. Nun bilden wir H_4 mit Hilfe von H_2 als

$$H_4 = H_2 \times H_2 = \begin{pmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{pmatrix} = \begin{pmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{pmatrix}.$$

Wir sehen, dass auch das Skalarprodukt zweier beliebiger verschiedener Zeilen/Spalten dieser Matrix 0 ergibt, was aus der entsprechenden Eigenschaft der Matrix H_2 folgt. Nun bilden wir auf die gleiche Weise die Matrix H_8 usw. Somit lässt sich rekursiv eine beliebig große Hadamard-Matrix H_n mit $n = 2^m$ ($m \in \mathbb{N}$) aufbauen. Induktiv lässt sich für jede dieser Matrizen zeigen, dass das Produkt zweier ihrer Zeilen/Spalten 0 ergibt. Nun gilt weiterhin: Wenn das Skalarprodukt zweier Vektoren, deren Komponenten jeweils $+1$ oder -1 sind, 0 ergibt (die Vektoren also zueinander senkrecht stehen), müssen sie sich wegen $1^2 = (-1)^2 = 1$ und $1 \cdot (-1) = (-1) \cdot 1 = -1$ in genau der Hälfte ihrer Komponenten unterscheiden.

Wir können nun die Zeilen einer Hadamard-Matrix H_n als Codewörter im Binärsystem betrachten, indem wir jede „$+1$“durch eine „0“und jede „-1“durch eine „1“ersetzen. Der Abstand zweier Codewörter beträgt nach den vorangegangenen Überlegungen genau $n/2$.

Wir haben nun n Codewörter der Länge n, jedoch sind im Allgemeinen mehr Codewörtern für eine effizientere Kommunikation wünschenswert. Den n Codewörtern aus den Zeilen der Hadamard-Matrix H_n kann man tatsächlich noch n weitere mit dem geforderten Mindestabstand $n/2$ hinzufügen, indem man die Inversen der ursprünglichen Codewörter bildet. Diese sind folgendermaßen definiert.

Definition 5. Das *Inverse* eines (binären) Codeworts wird gebildet, indem man im ursprünglichen Codewort jede 0 durch eine 1 ersetzt und umgekehrt. Beispielsweise ist das Inverse zu 1110 also 0001.

Um $2n$ Codewörter zu generieren, bilden wir nun zu jedem originalen Codewort des Hadamard-Codes auch das Inverse. Da die originalen Codewörter sich in genau der Hälfte der Stellen unterscheiden, gilt gleiches offensichtlich auch für die inversen Codewörter untereinander. Nun bleibt die Frage, ob jedes originale Codewort zu jedem inversen Codewort einen Mindestabstand von $n/2$ hat, damit die Fehlerkorrektur erhalten bleibt. Betrachten wir dazu ein beliebiges, aber festes, Originalcodewort und sein Inverses. Das Originalcodewort unterscheidet sich zu allen anderen Originalcodewörten an genau $n/2$ Stellen. An diesen jeweiligen Stellen gleicht das Inverse den Originalcodewörtern also. An den $n/2$ anderen Stellen, an denen das Originalcodewort den anderen Originalcodewörten gleicht, unterscheidet sich das Inverse jedoch von ihnen. Da schließlich das Originalcodewort und sein Inverses sich nach Konstruktion an allen Stellen unterscheiden, ist der Hamming-Abstand von $n/2$ also immer gewährleistet.

2. Modulo-Rechnung

2.1. Grundlagen

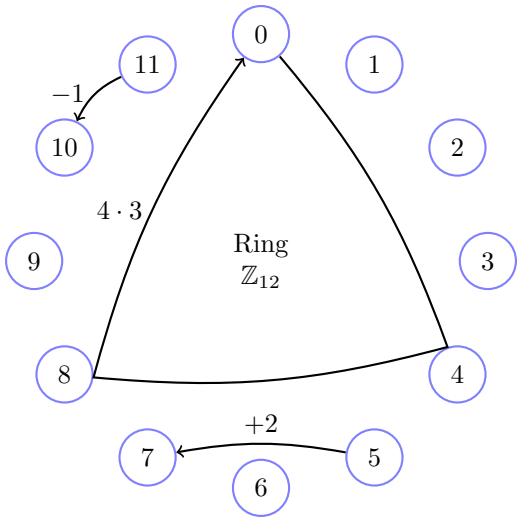
Es seien $a, b \in \mathbb{N}$ natürliche Zahlen. Die Schreibweise $a \equiv b \mod n$ notiert, dass a und b bei Division durch n den gleichen Rest lassen, also beispielsweise $19 \equiv 7 \mod 4$. Mit $r_n(a)$ bezeichnen wir den Rest von a bei Division durch n, $r_n(a)$ ist also eine der Zahlen $0, \ldots, n-1$. Die nun folgenden Verschlüsselungsmethoden verwenden statt der gewohnten Zahlbereiche $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ sogenannte Restklassenringe, die folgendermaßen definiert sind.

Definition 6. Der Ring $\mathbb{Z}_n, n \in \mathbb{N}$ besteht aus der Menge $\{0, \ldots, n-1\}$, deren Elemente die Restklassen bei Teilung durch n repräsentieren. Die Addition zweier Restklassen a und b ist durch $a + b := r_n(a + b)$, und die Multiplikation durch $a \cdot b := r_n(a \cdot b)$ definiert. Im Ring $\mathbb{Z}_4$ gilt also $2 + 3 = r_4(2 + 3) = r_4(5) = 1$, und analog $2 \cdot 3 = 2$.

Die folgende Abbildung illustriert die Addition, Subtraktion und Multiplikation im Ring $\mathbb{Z}_{12}$. Bei der Addition springt man die gegebene Zahl im Ring vor, bei der Subtraktion zurück. Multipliziert man eine Zahl $a \cdot b$, wiederholt man die Addition von b mit sich selbst a-mal.

Addition:	$5 + 2 = 7$
Subtraktion:	$11 - 1 = 10$
Multiplikation:	$4 \cdot 3 = 0$

$5 \cdot 0 = 0$	$5 \cdot 6 = 6$
$5 \cdot 1 = 5$	$5 \cdot 7 = 11$
$5 \cdot 2 = 10$	$5 \cdot 8 = 4$
$5 \cdot 3 = 3$	$5 \cdot 9 = 9$
$5 \cdot 4 = 8$	$5 \cdot 10 = 2$
$5 \cdot 5 = 1$	$5 \cdot 11 = 7$
	$5 \cdot 12 = 0$



Bei der Multiplikationen mit 4 fällt auf, dass sich der Zyklus nach 3 Schritten wiederholt. Multipliziert man jedoch mit 5, so wird jede Zahl des Rings, insbesondere die 1, angenommen. Dies führt zu folgender Definition.

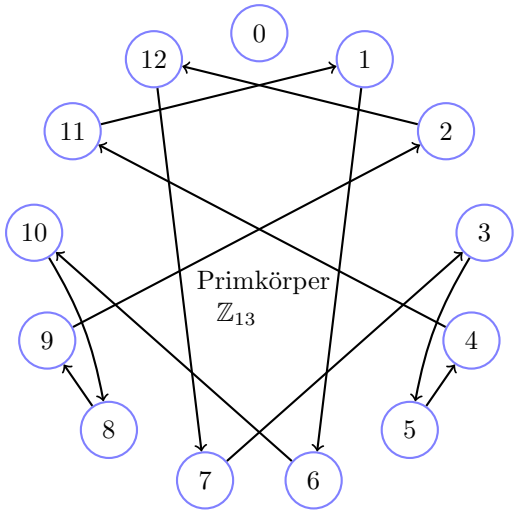
Definition 7. Es sei $a \in \mathbb{Z}_n$. Gibt es ein $b \in \mathbb{Z}_n$, für das gilt $a \cdot b = 1$, so heißt a *invertierbar* und das *Inverse* b wird auch als a^{-1} geschrieben. Man überlegt sich, dass das Inverse von a genau dann existiert, wenn $ggT(a, n) = 1$ gilt. Wir bezeichnen die invertierbaren Zahlen in $\mathbb{Z}_n$ mit $\mathbb{Z}_n^\times$ und die Anzahl der Elemente in $\mathbb{Z}_n^\times$ mit $\varphi(n)$.

Einfache Überlegungen zeigen, dass für verschiedene Primzahlen p und q die Gleichheiten $\varphi(p) = p-1$ und $\varphi(pq) = (p-1) \cdot (q-1)$ gelten. Diese speziellen Werte von $\varphi(n)$ werden weiter unten wieder auftauchen. Wenn in einem Ring alle Zahlen außer der Null invertierbar sind, nennt man ihn Körper. Die Ringe $\mathbb{Z}_p, p \in \mathbb{P}$ eine Primzahl, sind Körper, sogenannte Primkörper.

Definition 8. Wenn die Potenzen einer Zahl $g \in \mathbb{Z}_p^\times$ jede Zahl in $\mathbb{Z}_p^\times$ annehmen, heißt g Primitivwurzel oder Generator von $\mathbb{Z}_p^\times$.

Im Körper $\mathbb{Z}_{13}^\times$ gibt es beispielsweise die Generatoren 2, 6, 7 und 11. Die Generatoreigenschaft von 6 wird in der folgenden Abbildung illustriert.

k	6^k
0	1
1	6
2	10
3	8
4	9
5	2
6	12
7	7
8	3
9	5
10	4
11	11
12	1



Anhand der obigen Tabelle mit den Potenzen des Generators 6 in $\mathbb{Z}_{13}$ sehen wir: Aufgrund der Potenzgesetze lässt sich eine Multiplikation der Potenzen in eine Addition der Exponenten übersetzen und umgekehrt. Nach $13 - 1 = 12$ Schritten wiederholen sich die Werte. Wir werden gleich zeigen, dass dies für beliebige Primkörper $\mathbb{Z}_p$ gilt. Die Elemente aus $\mathbb{Z}_p^\times$ unter Multiplikation verhalten sich also genauso wie die Elemente aus $\mathbb{Z}_{p-1}$ unter Addition. Das Rechnen mit den Exponenten erfolgt also in $\mathbb{Z}_{p-1}$.

2.2. Kleiner Fermatscher Satz

Satz 1. *Es sei $p \in \mathbb{P}$ eine Primzahl. Dann gilt für alle $a \in \mathbb{Z}$*

$$a^p \equiv a \mod n$$
$$\textit{bzw. } a^{p-1} \equiv 1 \mod p \quad \textit{für } a \not\equiv 0 \mod n.$$

Für den Beweis benutzen wir folgendes Werkzeug:

Lemma 1. *Seien $a, b \in \mathbb{Z}$, dann gilt*

$$(a+b)^p \equiv a^p + b^p \mod p.$$

Beweis: Nach der Binomialformel gilt

$$(a+b)^p = \sum_{i=0}^{p} \binom{p}{i} a^i b^{p-i} = a^p + b^p + \sum_{i=1}^{p-1} \binom{p}{i} a^i b^{p-i}.$$

Damit unsere Annahme stimmt, muss gezeigt werden, dass $\binom{p}{i}$ für $0 < i < p$ durch p teilbar ist. Nun gilt

$$\binom{p}{i} = \frac{p!}{i!(p-i)!}.$$

Der Quotient ist eine natürliche Zahl. Der Faktor p im Zähler kann nicht gekürzt werden, da p die größte auftretende Primzahl ist. Daher ist p Teiler von $\binom{p}{i}$ □

Beweis: Kleiner Fermatscher Satz: Wir betrachten zunächst $a \geq 0$ und führen eine Induktion durch. Für den Fall $a = 0$ gilt die Annahme. Für den Induktionsschritt folgt aus dem Hilfssatz und der Induktionsannahme

$$(a+1)^p \equiv a^p + 1^p \equiv a + 1 \mod p.$$

Für $a \leq 0$ ist $-a \geq 0$, und es gilt nach dem oben gezeigten $(-a)^p \equiv -a \mod p$. Für $p = 2$ ist $a \equiv -a$, und die Aussage folgt. Für eine ungerade Primzahl p gilt $a^p = (-1)^p(-a)^p \equiv (-1)^p(-a) \equiv a \mod p$, und die Aussage folgt wiederum. □

2.3. Erweiterter kleiner Fermatscher Satz

Es seien $p, q \in \mathbb{P}$, $p \neq q$, zwei Primzahlen, $N = p \cdot q$, und $a \in \mathbb{Z}$ mit $\mathrm{ggT}(a, N) = 1$. Dann gilt

$$a^{(p-1)(q-1)} \equiv 1 \mod pq.$$

Beweis: Wir wissen

$$a^{(p-1)\cdot(q-1)} = \left(a^{(p-1)}\right)^{q-1} = \left(a^{(q-1)}\right)^{p-1}$$
$$\text{und} \quad \left(a^{(p-1)}\right)^{(q-1)} \equiv 1^{(q-1)} \equiv 1 \mod p$$
$$\text{und} \quad \left(a^{(q-1)}\right)^{(p-1)} \equiv 1^{(p-1)} \equiv 1 \mod q$$

nach dem kleinen Fermatschen Satz. Somit sind p und q Teiler von $a^{(p-1)(q-1)} - 1$. Da p und q ungleiche Primzahlen und dadurch teilerfremd sind, folgt $N = p \cdot q | a^{(p-1)(q-1)} - 1$. Damit gilt die Behauptung des erweiterten kleinen Fermatschen Satzes. □

Beispiel 2. Sei $a = 2$, $p = 3$ und $q = 5$. Zu zeigen ist $2^{(3-1)\cdot(5-1)} \equiv 1 \mod (3 \cdot 5)$. Wir berechnen

$$2^{(3-1)\cdot(5-1)} = 2^8 = 256 = 15 \cdot 17 + 1 \equiv 1 \mod 15.$$

3. Diffie-Hellmann

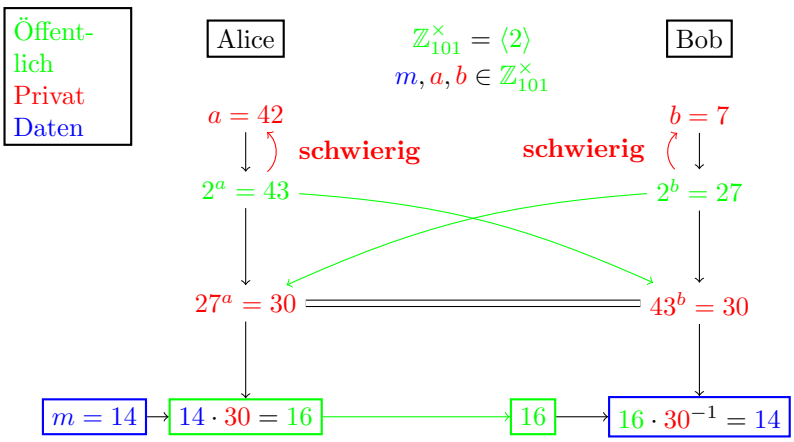
3.1. Das Verfahren

Das Diffie-Hellmann-Verfahren ist eine Methode, um sicher einen Schlüssel für eine Ende-zu-Ende-Kommunikation zu erstellen. Diese Kommunikation kann dann über eine unsichere bzw. öffentliche Verbindung geschehen. Nehmen wir an, die beiden Personen Alice und Bob wollen einen Schlüssel generieren, um danach eine symmetrische Verschlüsselung durchzuführen.

Algorithmus.

1. Beide Personen einigen sich auf einen öffentlichen Primkörper $\mathbb{Z}_p$ und einen Generator g von $\mathbb{Z}_p^\times$.
2. Jeder wählt eine geheime Zahl $a \in \{2, \dots, p-2\}$ bzw. $b \in \{2, \dots, p-2\}$ des Körpers.
3. Jeder potenziert den Generator mit seiner Geheimzahl und veröffentlicht das Ergebnis g^a bzw. g^b. Die Sicherheit dieser Schlüsselgeneration beruht darauf, dass es sehr schwierig ist, mit dem Ergebnis und dem Generator auf den geheimen Exponenten zu schlussfolgern, denn $g^n, n \in \mathbb{N}$ springt wie bereits gesehen „willkürlich“ im Ring umher.
4. Nun potenzieren beide das Ergebnis des Partners mit ihrer Geheimzahl und erhalten identische Schlüssel $\left(g^b\right)^a = \left(g^a\right)^b = g^{ab}$.
5. Jetzt kann Alice eine Nachricht $m \in \mathbb{Z}_p^\times$ mit dem symmetrischen Verschlüsselungverfahren ElGamal an Bob übermitteln. Dafür multipliziert Alice die Nachricht m mit dem Schlüssel und sendet das Produkt $c = m \cdot g^{ab}$ an Bob.
6. Bob kann wiederum das Produkt mit dem Inversen des Schlüssel dechiffrieren und erhält wieder die ursprüngliche Nachricht $m = c \cdot \left(g^{ab}\right)^{-1}$.

Wir illustrieren das Verfahren nun am Beispiel $p = 101$, $m = 14$, $a = 42$, $b = 7$.



3.2. Baby-Step-Giant-Step

Überlegung. Wenn man das Diffie-Hellman-Verfahren knacken will, müsste man nur mithilfe von $A = g^a$ und g den Wert von a herausfinden. Der naive Ansatz der einem zuerst in den Sinn kommt, wäre alle Potenzen $g^0, \ldots, g^{p-1}$ auszuprobieren und mit A zu vergleichen. Dies wäre jedoch sehr ineffizient und würde für eine hohe Laufzeit sorgen, und wenn man versucht, eine Nachricht zu entschlüsseln, ist der Zeitfaktor essentiell.

Definition 9. Der diskrete Logarithmus $\log_g(A) = x$ zur Basis g eines Elementes A ist definiert als die Lösung der Gleichung $g^x = A$ in einem Körper $\mathbb{Z}_p^\times$. Dabei kann x maximal Werte zwischen 0 und $p-1$ annehmen. Aufgrund der Natur der Primkörper ist es sehr schwer, dafür eine Lösung zu finden, da auf solchen Körpern die typische Strategie durch Vergleichen der Größen nicht funktionieren, weil im Allgemeinen kein Zusammenhang zwischen Größe des Exponenten und Wert besteht.

Ansatz. Eine Idee, die benötigte Zeit zu reduzieren, ist der sogenannte Baby-Step-Giant-Step-Algorithmus, mit welchem wir den diskreten Logarithmus effizienter berechnen können. Er arbeitet nur mit einer Zeitkomplexität von $O(\sqrt{p})$, was weitaus besser ist als eine Laufzeit von $O(p)$, welche wir beim Durchprobieren der Potenzen von g hätten. Um diese Verbesserung zu erreichen, zerlegen wir den Exponenten durch Teilung mit Rest.

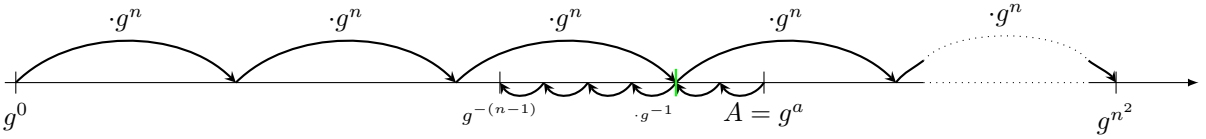
Algorithmus. Wir definieren zuerst $n \coloneqq \left\lceil \sqrt{|\mathbb{Z}_p^\times|} \right\rceil = \min\left\{k \in \mathbb{N} : k^2 \geq p-1\right\}$. Es ist also n^2 eine obere Schranke für die Kardinalität von $|\mathbb{Z}_p^\times|$. Durch eine Division mit Rest von a durch n erhalten wir $a = qn + r$, mit $r \in \{0, ..., n-1\}$ und $q \in \{0, ..., n\}$. Es gilt dann $A = g^a = g^{qn+r}$, was sich in $Ag^{-r} = g^{qn}$ umformen lässt. Um nun q und r, und damit a, zu finden, gehen wir folgendermaßen vor.

1. Wir bestimmen zum Lösen der Gleichung $Ag^{-r} = g^{qn}$ zwei Listen von Gruppenelementen.
   Die Baby-Step-Liste: $Ag^0, Ag^{-1}, Ag^{-2}, \ldots, Ag^{-(n-1)}$.
   Die Giant-Step-Liste: $g^0, g^n, g^{2n}, \ldots, g^{n^2}$.

2. Nun werden die Listen verglichen, was eine Aufgabe ist, die ein Computer sehr effektiv durchführen kann. Sobald dann eine Übereinstimmung gefunden wurde, kann $a = qn + r$ bestimmt werden.

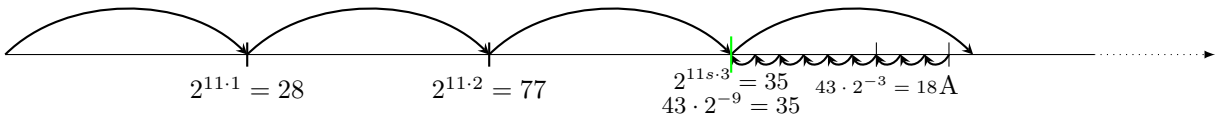
3. Der geheime Schlüssel g^{ab} kann nun mithilfe von a und dem öffentlichen g^b berechnet werden.

Anschaulich wird der Algorithmus in der folgenden Abbildung dargestellt.



Dieser Zahlenstrahl ist nach den Exponenten von g geordnet und nicht nach den tatsächlichen Werten, da diese aufgrund der Natur der Primkörper „wild“ umherspringen würden. Der Ort der Kollision ist grün markiert. Da jedes $a \in \{0, \ldots, p-1\}$ in der Form $a = qn + r$ mit $r \in \{0, ..., n-1\}$ und $q \in \{0, ..., n\}$ dargestellt werden kann, wird immer eine Übereinstimmung gefunden, weshalb dieser Algorithmus immer funktioniert.

Beispiel 3. Wir betrachten das Beispiel $A = 43$, $g = 2$ im Primkörper $\mathbb{Z}_{101}^\times$. Wir wählen $n = 11$ wegen $11^2 \geq 100 = |\mathbb{Z}_{101}^\times|$. Die folgende Skizze demonstriert den Algorithmus, wobei die Zeichnung dort abgebrochen wurde, wo ein höherer Exponent als der von A erreicht wurde, was man in der Realität nicht wissen kann.



Wie man nun an der Abbildung ablesen kann, ist $q = 3$ und $r = 9$. Das heißt, wir erhalten $a = 3 \cdot 11 + 9 = 42$. Das kann man schnell überprüfen indem man in g^a einsetzt: Es gilt tatsächlich $2^{42} = 43$.

4. RSA

4.1. Das RSA-Verfahren

RSA ist ein asymmetrisches Kryptografie-Verfahren, welches den sicheren Austausch von Nachrichten über eine öffentliche Leitung ermöglicht. Damit finden sich zahlreiche Anwendungsgebiete im alltäglichen Leben, wie beispielsweise beim Datenaustausch im Internet.

Algorithmus.

1. Der Empfänger wählt zwei Primzahlen p, q mit $p \neq q$.
2. Er bestimmt die Produkte $N = p \cdot q$ und $\varphi(N) = (p-1) \cdot (q-1)$.
3. Anschließend wählt er $e \in \mathbb{N}_{\geq 1}$ mit $e < \varphi(N)$ und $ggT(e, \varphi(N)) = 1$ und macht N und e öffentlich.
4. Nun bestimmt er das multiplikativ Inverse d zu e modulo $\varphi(N)$. Es gilt also
$$e \cdot d \equiv 1 \mod \varphi(N) \ .$$
5. Der Sender sendet seine Nachricht m verschlüsselt als $c = m^e \mod N$.
6. Zum Entschlüsseln bestimmt der Empfänger $m = c^d \mod N$.

Behauptung. Wenn man den privaten Schlüssel auf den verschlüsselten Text anwendet, erhält man den Originaltext. Es gilt also:
$$(m^e)^d \equiv m \mod N \ .$$

Beweis: Es gilt $(m^e)^d = m^{e \cdot d}$. Wir können $e \cdot d$ nach Voraussetzung durch $k \cdot \varphi(N) + 1$ ersetzen. Setzt man dies nun oben ein, erhält man $m^{e \cdot d} = m^{k \cdot \varphi(N)+1} = m^{k \cdot \varphi(N)} \cdot m = \left(m^{\varphi(N)}\right)^k \cdot m$. Aus dem erweitertem Satz von Fermat folgt:
$$m^{\varphi(N)} \equiv 1 \mod N,$$
und damit $\left(m^{\varphi(N)}\right)^k \cdot m \equiv 1^k \cdot m \equiv m \mod N$. Somit gilt $(m^e)^d \equiv m \mod N$. □

Beispiel 4. Sei $N = 7 \cdot 11 = 77$ und $e = 13$. Mit Hilfe von $\varphi(N) = 6 \cdot 10 = 60$ lässt sich $d = 37$ bestimmen. Weiterhin betrachtet man die Nachricht $m = 14$. Es ergibt sich:
$$\left(14^{13}\right)^{37} = 14^{13 \cdot 37} = 14^{8 \cdot 60 + 1} = 14^{8 \cdot 60} \cdot 14 = \left(14^{60}\right)^8 \cdot 14$$
mit $\left(14^{60}\right) \equiv 1 \mod 77$. Es folgt $\left(14^{60}\right)^8 \cdot 14 \equiv 1^8 \cdot 14 \equiv 14 \mod 77$.

Bemerkung 4. Eine abgefangene, verschlüsselte Nachricht c lässt sich trotz der öffentlichen Informationen e, N nicht bestimmen, da das Potenzieren mit e nur mit Hilfe von d aufgehoben werden kann. Doch um $\varphi(N) = (p-1)(q-1)$, und damit d, zu berechnen, müsste N faktorisiert werden. Dies ist für große Werte von p, q sehr aufwändig. Somit bleibt auch d geheim und das Verfahren ist sicher.

4.2. Pollards $p-1$-Methode

Die Sicherheit von RSA basiert darauf, dass N nicht effizient in p und q faktorisiert werden kann. Es gibt aber effizientere Faktorisierungsverfahren, als für alle Zahlen bis $\sqrt{N}$ eine Probedivision durchzuführen. Wir betrachten folgenden Ansatz: Sei $N = p \cdot q$ mit $p \in \mathbb{P}, q \in \mathbb{N}_{>1}$. Für $a \in \mathbb{N}_{>0}, \mathrm{ggT}(a, N) = 1$ und somit $\mathrm{ggT}(a, p) = 1$ gilt nach dem kleinen Satz von Fermat

$$a^{p-1} \equiv 1 \mod p.$$

Somit gilt auch für Vielfache k von $p-1$

$$a^k \equiv 1 \mod p, \text{ d.h. } p \mid a^k - 1.$$

Wegen $p \mid a^k - 1$ gilt

$$p \mid d = \mathrm{ggT}(a^k - 1, N).$$

Falls $d \neq N$ ist, haben wir einen echten Faktor von N gefunden. Die nun folgende $p-1$-Methode ist dann effektiv, wenn $p-1$ ein Produkt von kleinen Primzahlen ist. Wir können dies spezifizieren: Wir nennen eine Zahl *B-glatt*, wenn gilt $p \mid m \implies p \leq B$. Dies führt zu folgendem Algorithmus.

Algorithmus.

1. Wähle $B \in \mathbb{N}$ und bestimme alle Primzahlen $p_1, \ldots, p_n \leq B$.

2. Wähle $\nu_i \in \mathbb{N}$ maximal mit $p_i^{\nu_i} \leq B$ für alle $i \in \{1, \ldots, n\}$.

3. Wähle einen Startwert $a_0 \in \mathbb{N}_{>0}$ mit $\mathrm{ggT}(a_0, N) = 1$.

4. Berechne $a_i \equiv a_{i-1}^{p_i^{\nu_i}} \mod N$ und bestimme $d := \mathrm{ggT}(a_i - 1, N)$. Dies kann effizient mit dem euklidischen Algorithmus durchgeführt werden.

   - Falls $d = 1$: Fahre mit dem nächsten i fort.
   - Falls $d > 1$ und $d \neq N$ gilt, ist d Teiler von N. Falls $d = N$ wähle a_0 neu.

5. Falls $\mathrm{ggT}(a_n - 1, N) = 1$, dann gibt es keine B-glatten Primteiler von N. Man wählt dann ein größeres B.

Bemerkung 5. Falls $p-1$ aus kleinen Primteilern besteht, erhalten wir nach einigen Iterationen bei Schritt 4 die Gleichung $a_i \equiv {a_0}^k$, wobei k ein Vielfaches von $p-1$ ist. Deshalb gilt nach dem kleinen Satz von Fermat $d = \mathrm{ggT}(a_i - 1, N) \mid N$.