JuniorAkademie Adelsheim

16. Science Academy Baden-Württemberg 2018

Biologie

Chemie/Technik

Informatik

Mathematik

Philosophie

Fotografie

HOPP FOUNDATION for computer literacy & informatics

Regierungspräsidium Karlsruhe
Abteilung 7 – Schule und Bildung

Stiftung Bildung und Jugend

Dokumentation der JuniorAkademie Adelsheim 2018

16. Science Academy Baden-Württemberg

Veranstalter der JuniorAkademie Adelsheim 2018:
Regierungspräsidium Karlsruhe
Abteilung 7 –Schule und Bildung–
Hebelstr. 2
76133 Karlsruhe
Tel.: (0721) 926 4245
Fax.: (0721) 933 40270
www.scienceacademy.de
E-Mail: joerg.richter@scienceacademy.de
monika.jakob@scienceacademy.de
rico.lippold@scienceacademy.de

Die in dieser Dokumentation enthaltenen Texte wurden von der Kurs- und Akademieleitung sowie den Teilnehmerinnen und Teilnehmern der 16. JuniorAkademie Adelsheim 2018 erstellt. Anschließend wurde das Dokument mit Hilfe von LaTeX gesetzt.

Gesamtredaktion und Layout: Jörg Richter

Vorwort

Meine Damen und Herren, wir begrüßen Sie recht herzlich an Bord unseres Fluges mit der JuniorAkademie2k18 über Adelsheim. Wir bitten Sie jetzt, Ihre Sitzplätze einzunehmen und möglicherweise ablenkende Objekte sicher außerhalb Ihrer Reichweite zu verstauen. Wir möchten Sie nun mit der Dokumentation der Science Academy 2018 vertraut machen!

Dear Ladies and Gentlemen ... auch in diesem Sommer haben sich wieder 72 Passagiere auf dem Gelände des Landesschulzentrums für Umwelterziehung, kurz: LSZU eingefunden, um mit ihrer 30-köpfigen Crew aus Akademie-, Kurs- und KüA-Leitenden die 16. Science Academy Baden-Württemberg zu erleben.

In jedem Jahr steht die Akademie unter einem besonderen Motto. Wie unschwer zu erraten ist, drehte sich dieses Jahr alles um das Thema „Fliegen“. Durch verschiedene Aktionen und Denkanstöße dazu konnten die Zeit in Adelsheim und die vielen Erlebnisse, die hier schnell einmal wie im Flug an einem vorbeirauschen, aus einer anderen Perspektive betrachtet und reflektiert werden.

In den sechs Kursen lernten die Teilnehmerinnen und Teilnehmer die Welt des wissenschaftlichen Arbeitens anhand verschiedener Themen kennen. Während die einen Tomaten gepflanzt, Schiffe versenkt oder Nachrichten verschlüsselt haben, wurden in anderen Kursen spannende Zaubertricks durchschaut, das Thema Zeit beleuchtet oder professionelle Fotos geknipst.

Neben den rein fachlichen Aspekten konnten die Teilnehmerinnen und Teilnehmer hierbei auch neue Arten zu Lernen und zu Arbeiten entdecken und Fähigkeiten wie beispielsweise ihre Präsentationstechnik verbessern.

Auch wenn alle mit unterschiedlichsten Erwartungen, Hoffnungen und Wünschen im Gepäck nach Adelsheim gereist sind, so saßen wir hier doch alle im selben Flieger und wuchsen schnell zu einer großen, bunten Gruppe zusammen. Die einzigartige Akademieatmosphäre, die entsteht, wenn so viele interessierte und motivierte Leute zusammenkommen, bringt viele spannende Gespräche, neue Interessen und häufig auch bereichernde Freundschaften mit sich.

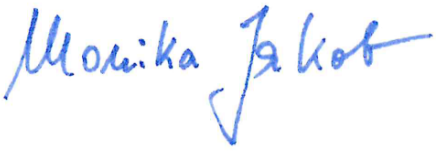
Auch wenn unsere Wege jetzt in verschiedene Richtungen gehen werden, wünschen wir euch alles Liebe und Gute, und dass Ihr noch lange vom Akademiefieber beflügelt seid. Wir freuen uns darauf, euch wiederzusehen (vielleicht ja sogar in Adelsheim?), und jetzt bleibt nur noch zu sagen: Sie können den Sicherheitsgurt nun wieder lösen.

Wir wünschen Euch und Ihnen viel Spaß und viele schöne Einblicke in unsere Akademiezeit beim Lesen der Dokumentation!

Eure/Ihre Akademieleitung

Johanna Kroll (Assistenz)

Johanna Rettenmeier (Assistenz)



Dr. Monika Jakob

Jörg Richter

Inhaltsverzeichnis

Kurs 3 – Informatik/Kryptographie

Vorwort

Antonia Münchenbach, Kevin Sommer, Michael Krüger

(EBG13) Zna zhff fpuba rva tnam orfbaqrere whatre Zrafpu frva, jraa zna zruerer Jbpura qre rvtrara Servmrvg qnmh irejraqrg, fvpu zvg xbzcyrkra grpuavfpura haq jvffrafpunsgyvpura Vaunygra mh orsnffra. Jrvgreuva jheqr ivry Fcbeg trznpug, zhfvmvreg, (zvg Erab) tronpxra haq abpu Ivryrf zrue. Jve serhra haf üore rvara resbytervpura Xhef haq ceäfragvrera fgbym qrffra Retroavffr.

Einleitung und Kursziel

Henry Wacker

Immer mehr Menschen nutzen für das Verschicken von Nachrichten das Internet. Doch Vielen ist dabei nicht bewusst, dass die gesendeten Nachrichten nicht direkt zum Empfänger übermittelt werden, sondern zuerst sehr viele verschiedene Stationen passieren müssen (siehe Abbildung 1). Jede dieser Stationen ist theoretisch dazu imstande, die gesendeten Nachrichten abzufangen. Damit die Kommunikation sicher und privat ist, dürfen diese Zwischenstationen nicht in der Lage sein, die Nachrichten zu lesen.

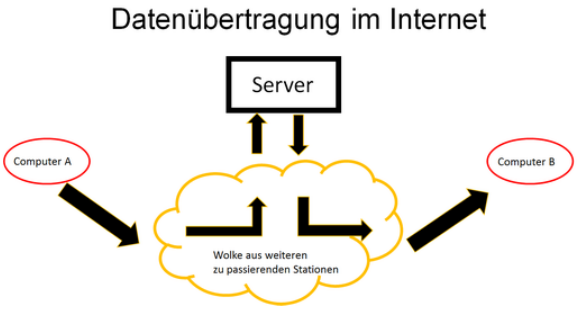


Abbildung 1: Datenübertragung im Internet

Wir, der Informatikkurs der Science Academy 2018, haben uns ausgiebig mit dem Problem der Verschlüsselung beschäftigt. Die Wissenschaft der Verschlüsselung von Informationen

nennt man Kryptographie. Unser Ziel war es, eine Chatanwendung im Web-Browser zu programmieren, bei der unsere gesendeten Nachrichten so verschlüsselt werden, dass Dritte die Nachrichten nicht mitlesen können. Am Anfang mussten wir uns überlegen, welche Eigenschaften unsere Website besitzen sollte. Schließlich einigten wir uns auf eine Chatanwendung, bei der man sich auf einer Website registrieren, andere Benutzer suchen und vor allem auch mit ihnen schreiben können sollte.

Um dies zu erreichen, haben wir uns des Öfteren in zwei Gruppen aufgeteilt. Die eine Gruppe befasste sich hauptsächlich mit den Programmiersprachen hinter unserer Website und dem Design. Die andere Gruppe beschäftigte sich mit komplizierter Mathematik, um für ausreichend Sicherheit beim Chatten zu sorgen.

Die Sorgen, die manche anfangs wegen der unterschiedlichen Vorkenntnisse hatten, waren im Endeffekt völlig unbegründet. Natürlich gab es hin und wieder einige Schwierigkeiten, wenn man etwas nicht verstand oder ein kleiner Fehler im Programmiercode das ganze Programm aussetzen lies. Aber es entstand ein gutes Team, das ein tolles Ergebnis erzielte, welches wir im Folgenden vorstellen werden.

Unser Kurs

Aileen ist eine stets gut gelaunte, motivierte Programmiererin. Bei Problemen gibt sie nie auf und arbeitet ständig weiter. Zudem hat sie ein sehr interessantes Hobby: Sie lernt π auswendig. Derzeit (13.10.2018) kann sie die ersten 606 Nachkommastellen von π. Ihr Hobby ist sehr ansteckend, inzwischen können mehrere Teilnehmer der Akademie mehr als 50 Nachkommastellen der Kreiszahl aufzählen.

Charlotte hilft uns oft bei den mathematischen Algorithmen und hat immer den Durchblick beim Mathe-Teil. Mit ihrer freundlichen Art trägt sie positiv zum Kursklima bei. Auch bei der Vorbereitung zur Rotations- und Abschlusspräsentation hat sie sich viel Mühe gegeben, z. B. bei Inhalt und Gestaltung der Plakate.

Aber nicht nur im mathematischen Teil findet sich Charlotte schnell zurecht. Auch beim Programmieren nimmt sie alle neuen Informationen sofort auf und setzt diese in praxisbezogenen Aufgaben um.

Helene ist eine tatkräftige Programmiererin und brachte entscheidende Ideen ein, die uns schließlich zum Ziel führten. Neben dem Programmieren bleibt aber auch immer noch Zeit zum Scherzen übrig. Aufgeweckt und immer anwesend folgte sie Kevins und Michaels Ausführungen, sodass sie einiges an Wissen mitnehmen konnte. Kleinere Scherze über ihren Namen nahm Helena mit Humor und konnte darüber lachen. So war sie eine unersetzliche Teilnehmerin in unserem Kurs.

Henry sorgt für ein gutes Klima im Kurs, da er immer einen Witz auf Lager hat, für jeden Spaß zu haben ist und gerne Schnellschreibtests absolviert. Außerdem war er so ambitioniert, dass er während der Kurszeit eine Website, die sogenannte „Läbens Website“, mit Jannis erstellte. Um das „Läbensgefühl“ perfekt zu machen, war er einer der zwei Gründer der sogenannten „Läbensgruppe“.

Jannis, der zweite Gründer der „Läbensgruppe“, hat von Anfang an zu allem einen guten Spruch parat. Obwohl er zunächst dachte, er wolle mehr programmieren, erkannte er schnell, wie ihn die Mathematik hinter unserer Website fesselt. Er wurde zu einem wichtigen Mathematiker unseres Kurses. Des Weiteren präsentiert er wie

kein Zweiter, und man kann in diesem Bereich viel von ihm lernen. Er unterhält seine Mitmenschen immer mit seiner lustigen Art und alles, auch in der Freizeit außerhalb des Kurses, machte mit ihm Spaß. Der einzige Ort, an dem er keinen Spaß verstand, war die Mensa.

Jascha ist uns allen eine sehr große Hilfe. Vorallen unseren Kursleitern, wenn es darum ging, das Whiteboard zu putzen. Er hat uns aber auch oft geholfen und uns mit seinem Wissen zum Staunen gebracht. Wegen seines fehlenden Namensschilds hat er den Spitznamen „Dr. Doofenschmerz“ bekommen, und wegen seiner verrückten Hobby-Projekte, wie beispielsweise Schuhe, mit denen er an der Decke laufen kann, wurde er zu unserem „verrückten Wissenschaftler“. Außerhalb des Kurses wurde er am meisten bei der Tischtennisplatte gesehen oder wie er in seinem Zimmer an weiteren Projekten arbeitete. Auch beim Sportfest war er eine große Hilfe für unser Team.

Lena ist sowohl in der Mathegruppe als auch in der Programmiergruppe mit guten Ideen und viel Engagement dabei. Wenn jemand Hilfe braucht, kann man sich immer auf sie verlassen. Auch außerhalb des Kurses war Lena sehr ambitioniert. So hat sie jeden Mittag im Orchester Trompete gespielt und war beim Sportfest die Beste im Teebeutelweitwurf.

Leontine sorgt mit ihrer offenen und fröhlichen Art stets für eine positive Atmosphäre. Egal ob beim Programmieren, in der Mathegruppe oder bei der Präsentationsvorbereitung – überall war sie mit großem Interesse und Einsatz dabei. Doch mit mindestens genauso großem Eifer verbrachte sie die meiste Freizeit mit den Proben für die Musik-KüA, was sich, wie wir beim Hausmusikabend alle feststellen durften, wirklich gelohnt hat.

Máté arbeitet konstruktiv am Kursthema und erklärte erfolgreich den Matheteil bei der Präsentation. Außerdem ist er der Meister im Browser „T-Rex Game“. Obwohl er beim Sportfest ins „giftige Wasser“ trat und starb, konnte er unsere Gruppe ansonsten immer mit guten Ideen voranbringen.

Moritz ist ein begabter Mathematiker, und mit seinen mathematischen Fähigkeiten glänzte er im Kurs des Öfteren. Aber er zeigt sich auch sehr geschickt im Umgang mit PowerPoint, sodass er unsere Abschlusspräsentation um eine Animation erweitern konnte. Auch bringt er in Diskussionen immer gewinnbringend seine Meinungen und Ideen ein.

Salome kann toll programmieren. Doch besonders in den Pausen ist sie bei lustigen Dingen immer dabei. So war es kein Wunder, dass ihr Namensschild schon beim Eröffnungswochenende in den Brunnen fiel. Auch bei dem Sportfest kamen wir durch ihren Einsatz fast alle sicher ans andere Ufer, obwohl sie den für sie viel zu schweren Hammer kurzerhand ins Wasser warf. Am Abschlussabend durften wir alle feststellen, dass wir mit ihr ein großes Theatertalent im Kurs hatten.

Simon ist stets ein sehr aufmerksamer Teilnehmer, was so manchen Fehler in seinem Keim erstickt. Braucht man bei der Umsetztung der mathematischen Theorie einen guten Programmierer, war Simon immer bereit, sein Wissen zu teilen und auszuhelfen, wo seine Hilfe benötigt wurde. Des Weiteren zeichnet ihn seine ruhige Art und die Bereitschaft aus, auf Kompromisse einzugehen. Doch er saß nicht den ganzen Tag hinter dem Schreibtisch und programmierte, nein, er ist ein leidenschaftlicher Basketballer, der gefühlt jeden Tag den selben Basketball-Pulli trägt.

Antonia ist eine äußerst enthusiastische und immer fröhliche Schülermentorin, der man ansehen konnte, dass sie gerne bei der Akademie dabei war. Sie hat uns jede Kurspause mit ihren heißgeliebten Kinderliedern und -spielen beschäftigt, obwohl ihre Begeisterung nicht immer hundertprozentig geteilt wurde. Auch wenn sie beim Sportfest Angst hatte, dass wir sie mitsamt dem „laufendem A“ fallen lassen, war sie immer für einen Spaß zu haben.

Kevin ist nicht nur ein leidenschaftlicher Informatiker, sondern auch Tänzer, der uns immer mit Süßigkeiten versorgte. Aufop-

ferungsvoll versuchte er, uns die Grundlagen der Informatik beizubringen und beteuerte oftmals, wie wichtig eine richtige Formatierung sei. Schlussendlich hat er uns das Programmierern trotz mancher schwieriger Themen erfolgreich beigebracht, und wir können uns jetzt stolz „Amateur-Programmierer“ nennen.

Michael hat die Fähigkeit, jedes noch so langweilige Thema durch unterschiedlichste, teils verrückte Ideen spannend zu machen. Als unser mathematik-spezialisierter Kursleiter wurden wir sowohl mit Informationen, als auch mit „Memes“ und sehr speziellen Gesängen versorgt. Seine große Leidenschaft ist sein Elektroauto.

Kryptographische Grundlagen

Salome Grote, Aileen Stroda

Kryptographie ist die Wissenschaft der verborgenen Übermittlung. Heute befasst sie sich auch mit Themen wie Informationssicherheit.

Klassische Kryptographie

Um die Kryptographie kennenzulernen, haben wir uns erst einmal mit klassischen Verschlüsselungstechniken wie z. B. der Cäsar-Chiffre und der Vigénère-Verschlüsselung beschäftigt

Cäsar-Chiffre

Die Cäsar-Chiffre oder auch Cäsar-Scheibe (siehe Abbildung 2) wurde von Julius Cäsar, zur Zeit der Römer, im militärischen Bereich zur Nachrichtenübermittlung genutzt. Sie basiert auf einem einfachen Prinzip, bei dem das Alphabet um eine beliebige Anzahl von x Buchstaben verschoben wird.

Dabei werden zwei Scheiben genutzt, von denen eine verdreht werden kann, sodass bei der Verschiebung um einen Buchstaben aus einem A ein B wird und bei einer Verschiebung um 2 Buchstaben aus einem A ein C wird. Um einen Buchstaben zu verschlüsseln, schaut man, wo er auf der äußeren Scheibe steht, und liest dann den zugehörigen Buchstaben auf der inneren Scheibe ab.

Beispiel: Wenn das Alphabet um 13 Zeichen verschoben wird, wird $A = N$. Bei dem Klartext „hallo“ ergibt sich folglich „unyyb“.

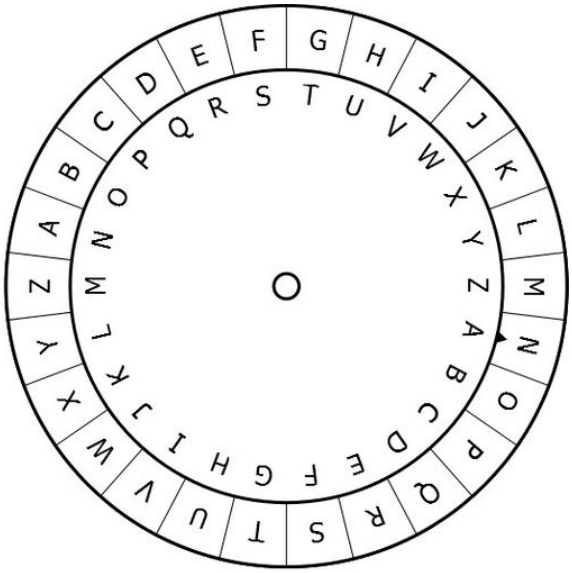


Abbildung 2: Cäsarscheibe

Nachdem aber das Prinzip hinter der Verschlüsselung durchschaut wurde, war sie aufgrund der 26 Buchstaben im Alphabet und den somit nur 25 Möglichkeiten der Verschlüsselung sehr leicht zu knacken.

Vigénère-Verschlüsselung

Etwas sicherer ist die Vigénère-Verschlüsselung. Sie ist eine Erweiterung der Cäsar-Verschlüsselung. Eine Buchstabenfolge bildet den Schlüssel, und so wird nicht jeder Buchstabe mit der gleichen Verschiebung verschlüsselt. Beispielsweise könnte man den ersten Buchstaben um 6 verschieben, den zweiten um 20, den dritten um 19, den vierten wieder um 6 und so weiter.

Beispiel: Das Wort „Schloss“ wird mit dem Schlüssel „gut“ verschlüsselt. Hierbei werden die Buchstaben von „gut“ immer wieder verwendet, bis jedem Buchstaben des Klartextes ein Buchstabe des Schlüsselwortes zugeordnet ist. Nun zählt man, an welcher Stelle im Alphabet das „g“ von „gut“ steht. In diesem Fall befindet es sich an der siebten Stelle im Alphabet. Nun verschiebt man das Alphabet für den ersten Buchstaben um 6 Stellen und kann dann den verschlüsselten Buchstaben ablesen.

Klartext	S	c	h	l	o	s	s
Verschiebung	g	u	t	g	u	t	g
Chiffre	Y	w	a	r	i	l	y

Beispiel einer Vigénère-Verschlüsselung

Knacken einfacher Verschlüsselungen

Zum Knacken der beiden Verschlüsselungsmethoden wurde ein Verfahren entwickelt, das heute noch eingesetzt wird und mit dem Computer einen Text in Sekundenbruchteilen entschlüsseln können: Die Buchstabenhäufigkeitsanalyse. In einem Text wird untersucht, wie häufig jeder Buchstabe vorkommt. Diese Buchstabenhäufigkeiten wird dann mit der Buchstabenhäufigkeit im Klartextalphabet verglichen.

Buchstabe	Häufigkeit
e	17,40 %
n	9,78 %
i	7,55 %
s	7,27 %
r	7,00 %
a	6,51 %
t	6,51 %

Häufigste Buchstaben im deutschen Sprachgebrauch

Wenn in einem mit der Cäsar-Verschlüsselung verschlüsselten Text der Buchstabe H am häufigsten vorkommt, so kann man davon ausgehen, dass er dem E entspricht, da das E der häufigste Buchstabe im deutschen Sprachgebrauch ist. Um von E zum H zu kommen, muss man das Alphabet um drei Buchstaben verschieben. Somit kann man jetzt jeden Buchstaben des verschlüsselten Textes um drei Buchstaben zurück verschieben und man hätte den gesamten Text entschlüsselt.

Am Besten funktioniert die Buchstabenhäufigkeitsanalyse mit einem langen Text, da dann die häufigen Buchstaben besser erkennbar und ihre Häufigkeit in Prozent deutlicher ist. Die Vigénère-Verschlüsselung ist an diesem Punkt sicherer, da sie die Häufigkeiten im Text verschleiert. Dennoch gibt es erweiterte Algorithmen, mit denen die Vigénère-Verschlüsselung einfach zu knacken ist.

Symmetrische und Asymmetrische Kryptographie

Die Cäsar- und Vigénère-Verschlüsselung sind Beispiele für die symmetrische Kryptographie. Bei der symmetrischen Kryptographie hat man nur einen Schlüssel, mit dem man den Text ver- und entschlüsseln kann.

Vorteil:
⇒ in der Regel schneller als asymmetrische Verschlüsselungen

Nachteile:
⇒ Schlüsselaustausch notwendig, sodass Dritte ihn abfangen könnten
⇒ Für vollständige Sicherheit muss pro Nachricht ein neuer langer Schlüssel ausgetauscht werden (One-Time-Pad)

Das Gegenstück dazu ist die asymmetrische Verschlüsselung. Jede Person hat zwei Schlüssel: einen öffentlichen Schlüssel zum Verschlüsseln und einen privaten Schlüssel zum Entschlüsseln. Eine Nachricht, die mit dem öffentlichen Schlüssel des Empfängers verschlüsselt wird, kann nur mit dem zugehörigen privaten Schlüssel entschlüsselt werden und umgekehrt (siehe Abbildung 3).

Wenn also eine Person A und eine Person B mithilfe einer asymmetrischen Verschlüsselung in Kontakt treten wollen, dann verschlüsselt Person A ihre Nachricht mit dem öffentlichen Schlüssel von Person B und schickt sie dann an diese. Wenn Person B die verschlüsselte Nachricht empfängt, kann sie diese nur mit ihrem eigenen privaten Schlüssel wieder entschlüsseln.

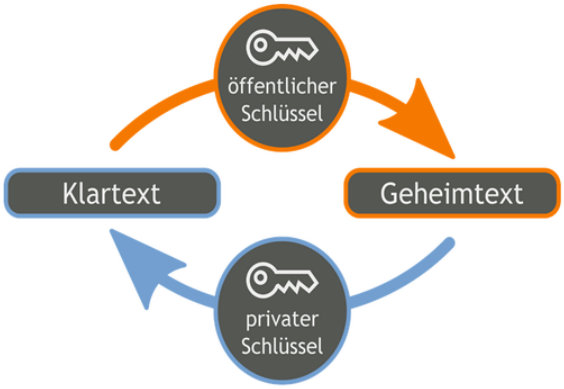


Abbildung 3: Asymmetrische Kryptographie

Vorteile:
⇒ bei ausreichender Schlüssellänge sicher
⇒ nur ein Schlüsselpaar pro Person nötig

Nachteil:
⇒ im Verhältnis langsamer als die symmetrische Kryptographie

Transportverschlüsselung

In unserem Kurs haben wir uns neben der Ende-zu-Ende-Verschlüsselung, die sicherstellt, dass unsere Nachrichten nur vom Chatpartner gelesen werden können, zusätzlich auch mit der Transportverschlüsselung beschäftigt. Die Transportverschlüsselung garantiert dem Benutzer, dass seine Daten ausschließlich vom Chatserver gelesen werden können werden. Sie sichert also die Informationen vor dem Abfangen oder Manipulieren durch Dritte.

Programmiergrundlagen

Moritz Klett, Henry Wacker, Máté Makszi

Um unsere Chatanwendung zu programmieren, benötigen wir jedoch nicht nur kryptographische Kenntnisse, sondern auch das passende Handwerkszeug. So mussten wir zusätzlich zu den kryptographischen Grundlagen auch noch Programmier- und Skriptsprachen erlernen, mit denen wir unsere Chatanwendung verwirklichen konnten. Da der Computer nicht mitdenkt, müssen die Befehle an den Computer eindeutig sein.

HTML

Eine dieser Sprachen ist HTML (Hypertext Markup Language), mit der sich die Struktur einer Webseite erstellen lässt. Sie ermöglicht zum Beispiel das Schreiben eines Textes oder einer Überschrift. Aber auch Buttons und Textfelder lassen sich hiermit erstellen.

Die Oberfläche in Abbildung 4 wird mittels folgendem HTML-Text programmiert:

```
Zahl 1 = <input [(ngModel)]= "Zahl 1"
   type= "number"/> <br>
Zahl 2 = <input [(ngModel)]= "Zahl 2"
   type= "number"/> <br>
<button (click) = "berechne()">
   berechne Summe</button>
Ergebnis= {{ergebnis}}
```

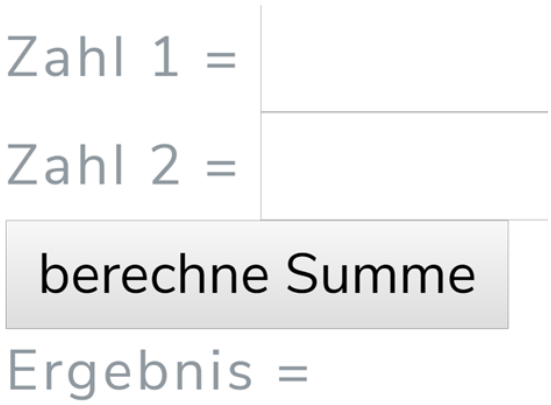


Abbildung 4: Ein mit HTML erstelltes Formular

CSS

Eine weitere Sprache, die wir im Laufe des Kurses kennengelernt haben, ist CSS (Cascading Style Sheets). Diese Stylesheet-Sprache ist für das Design der Webseite zuständig. So lässt sich mit ihr die Farbe oder Schriftgröße eines Textes ändern und wie gewünscht anpassen. Außerdem kann man die Hintergrundfarbe und den Abstand zum Rand und die Position der einzelnen Elemente auf der Webseite festlegen.

All diese Funktionen erleichtern die Programmierarbeit, da man durch CSS die Darstellungsformen von dem Inhalt trennen kann. So wird der gesamte Code übersichtlicher. Außerdem muss man zu Beginn den Code in CSS nur einmal in HTML einbinden. Dadurch kann man

auch nachträglich ganz einfach das Layout verändern.

Für einen grünen Button und eine weiße Schrift, wie in Abbildung 5, ist dieser CSS-Code nötig:

```
button {
    background-color: green;
    color: white;
}
```

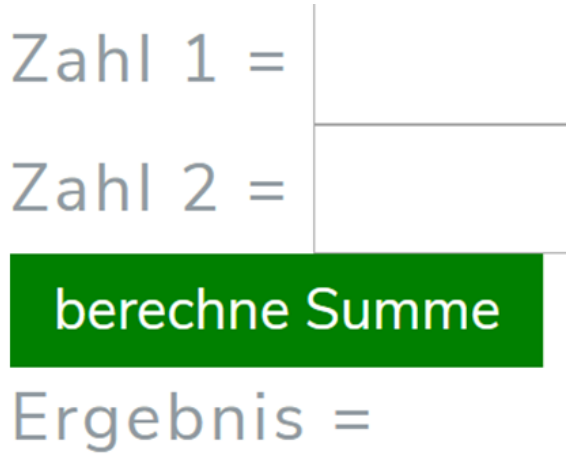


Abbildung 5: Das vorherige HTML Formular mit CSS verschönert

Durch den Gebrauch von CSS wird aus einer langweilig aussehenden eine schönere Webseite. Um nicht das ganze Design für unsere Seite selbst erstellen zu müssen, haben wir hierfür ein Framework namens Bootstrap eingesetzt. Dieses beinhaltet einige Grundeinstellungen wie Schriftarten und Anordnungsmöglichkeiten für Webseiten.

Im folgenden Code werden die drei Buttons als „Button“ definiert und benannt. Die Designvorlage dafür ist komplett in Bootstrap enthalten, sodass eine einheitliche „Designsprache“ wie in Abbildung 6 erkennbar wird.

```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
```

Mit HTML und CSS lassen sich nun statische Websiten umsetzten, jedoch ist die Webseite nach wie vor noch ohne Funktion.

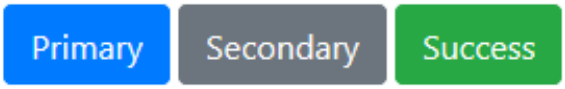


Abbildung 6: Verschiedene Button-Typen von Bootstrap

TypeScript

Um Funktionalität in unsere Chatanwendung einzubauen, mussten wir eine weitere Programmiersprache erlernen: TypeScript, eine von Microsoft entwickelte Programmiersprache, die auf Javascript basiert. Mit TypeScript ist es uns möglich, die Funktionalität unserer Website zu gewährleisten. Dabei benutzt man unter anderem Variablen und Funktionen. Mit Variablen ordnet man einem Wort eine bestimmte Zahl zu.

```
let ScienceAcademyTeilnehmer = 72;
```

Dies ist vor allem nützlich, wenn man einen Fehler gemacht hat. Hätte man nun in viele Befehlen immer 71 geschrieben, müsste man alle durchgehen und jede 71 in 72 ändern. Durch die Variable kann man jedoch nur diese eine definierte Zahl zu einer 72 ändern und jeder Befehl nimmt diese Änderung wahr.

Die Funktionen bringen, wie der Name schon sagt, Funktionalität ins Spiel. Ein Button, der vorher nur schön aussah, kann nun benutzt werden, um beispielsweise zwei Zahlen zu addieren.

So könnte eine Funktion zu diesem Beispiel aussehen:

```
berechne() {
 this.ergebnis = this.Zahl1 + this.Zahl2;
}
```

Dieser Term wird berechnet, sobald im HTML-Formular Zahl 1 und 2 definiert werden und der Button gedrückt wird.

Mit diesen drei Programmiersprachen kann man nun eine Webanwendung programmieren.

ASCII, Unicode, TextZuZahl

Da wir später unsere Textnachrichten verschlüsseln wollen und unsere Verschlüsselungsmethode nur mit Zahlen funktioniert, benötigen wir

einen Code, der jedes Zeichen in eine Zahlenfolge umwandelt. Der bekannteste Computer-Code ist ASCII (American Standard Code for Information Interchange), der jedem Zeichen eine Zahl zuordnet.

Zeichen	Zahl	Zeichen	Zahl
A	65	a	97
B	66	b	98
C	67	c	99
D	68	d	100
E	69	e	101
F	70	f	102
G	71	g	103

Ausschnitt aus der ASCII-Tabelle

Der Computer arbeitet im Binärsystem, das heißt nur mit Nullen und Einsen. Bei dem ASCII-Code wird jedem Buchstaben eine 7-stellige Ziffernfolge aus Binärzahlen zugeordnet. Deshalb gibt es 2^7, also 128 verschiedene Ziffernfolgen und damit können 128 verschiedene Zeichen dargestellt werden. Mit der Erweiterung ISO-8859-1 können doppelt so viele Zeichen, also 256 Stück, dargestellt werden. Da diese aber nicht immer reichen, werden andere Codes wie zum Beispiel Unicode verwendet.

In Unicode sind neben Buchstaben und Zahlen auch Umlaute, Emojis und alle weiteren bekannten Zeichen hinterlegt. Dafür werden deutlich mehr als 7 bzw. 8 Bit Speicherplatz benötigt.

Zeichen	Zahl
😃	128515
☺️	9786
😉	128521
🙈	128584
ポ	12509
ä	228
ö	246
ü	252

Ausschnitt aus der Unicode-Tabelle

Um einen ganzen Text in Zahlen zu übersetzen, reiht der Computer die Zahlen aus der Code-Tabelle aneinander. Die selbstprogrammierte Funktion „TextZuZahl“ wandelt die einzelnen Zahlenwerte der verschiedenen Zeichen eines ganzen Textes in eine einzelne große Zahl, die wir mit unseren Verschlüsselungsverfahren RSA verschlüsseln können. Der rechtmäßige Empfänger kann mit der Gegenfunktion „ZahlZuText“ die große Zahl wieder in einen Text wandeln.

```
textZuZahl(text: string): BigInteger {
 // Umwandeln der Nachricht Text in Zahl
 let summe: BigInteger = bigInt(0);
 const textBytes = toUTF8Array(text);
 for (let i = 0; i < textBytes.length; i++) {
  // Zahlenwert an aktueller Stelle holen
  const zeichen = bigInt(textBytes[i]);
  // Zahlenwert mit Wertigkeit der aktuellen
  // Stelle multiplizieren
  const zwischenergebnis = zeichen.multiply(
   bigInt(2).pow(
   (textBytes.length - 1 - i) * 8));
  // Zwischenergebnis zur Summe hinzufügen
  summe = summe.add(zwischenergebnis);
 }
 return summe;
}
```

Tools und Frameworks

Um nicht alle Komponenten in unserer Chatanwendung selbst programmieren zu müssen, haben wir ein Webframework namens Angular verwendet, das uns einige grundlegende Aufgaben abnimmt, wie zum Beispiel die Kommunikation mit unserem Chatserver, und uns eventuell schon vorgefertigte Programmteile zur Verfügung stellt. Angular ist ein Open-Source-Framework, das von Google entwickelt und kostenlos zur Verfügung gestellt wird.

Zum Programmieren unserer Chatanwendung haben wir die Open-Source-Software Visual Studio Code (siehe Abbildung 7) von Microsoft genutzt. Diese ist sehr gut geeignet, um Webseiten zu programmieren und auf dem eigenen Laptop zu testen.

Mathematische Grundlagen

Lena Stäble, Charlotte Bandke, Leontine Franz, Jannis Welsche, Simon Wessel

Bevor wir mit Verschlüsselungen arbeiten konnten, mussten wir uns zunächst mit einigen ma-

thematischen Grundlagen beschäftigen. Um die RSA-Verschlüsselung zu verstehen, mit der wir uns hauptsächlich beschäftigt haben, brauchen wir beispielsweise die Modulo-Rechnung sowie Primzahlen.

Abbildung 7: Visual Studio Code

Modulo-Rechnung

Die Modulo-Rechnung ist prinzipiell erweitertes Divisionsrechnen mit Rest, mit dem besonderen Merkmal, dass allein der Rest, der beim Teilen übrig bleibt, im Fokus liegt. Als Ergebnis schreibt man nur den Rest und den Teiler. Der Teiler wird bei einer Modulo Rechnung als Modulus bezeichnet und wird im Ergebnis mit „mod Teiler“ angegeben.

Beispiel: Die Rechnung lautet 72 geteilt durch 7. Das Ergebnis mit Rest lautet also 10 Rest 2. Man schreibt daher $72 \equiv 2 \mod 7$. Dies wird so ausgesprochen: „72 ist kongruent zu 2 Modulo 7“

Schnelle Exponentiation

Bei großen Zahlen dauert das Potenzieren sehr lange. Deshalb rechnet unser Programm mithilfe einer Technik namens „schnelle Exponentiation“. Hierbei wird die Potenz zerlegt. Eine Zahl a mit einer großen Zahl zu potenzieren dauert sonst sehr lange. Für das Beispiel nehmen wir die vergleichsweise kleine Zahl 400 und sehen bereits hier deutliche Unterschiede in der Rechenzeit.

Normales Potenzieren: $x^{400} = x{\cdot}x{\cdot}x{\cdot}x{\cdot}x{\cdot}\ldots$ (400 Rechenschritte)

Schnelle Exponentiation: Zunächst berechnet man x hoch alle 2-er-Potenzen, die relevant sind:

$$
\begin{aligned}
x^2 &= x^1{\cdot}x^1 \\
x^4 &= x^2 \cdot x^2 \\
x^8 &= x^4 \cdot x^4 \\
x^{16} &= x^8 \cdot x^8 \\
x^{32} &= x^{16} \cdot x^{16} \\
x^{64} &= x^{32} \cdot x^{32} \\
x^{128} &= x^{64} \cdot x^{64} \\
x^{256} &= x^{128} \cdot x^{128}
\end{aligned}
$$

Danach kombiniert man die korrekte Potenz aus den errechneten Zwischenwerten: $x^{400} = x^{256} \cdot x^{128} \cdot x^{16}$. Hier waren es insgesamt statt 400 Schritten nur 10 Schritte.

Wenn man statt x^{400} etwa $x^{1000000000}$ betrachtet, bemerkt man den riesigen Unterschied deutlich. Es sind etwa 50 Rechenschritte im Vergleich zu einer Milliarde!

Primzahlen

Primzahlen sind natürliche Zahlen > 1, die nur durch 1 und sich selbst ohne Rest teilbar sind. Die ersten Primzahlen sind $2, 3, 5, 7, 11, \ldots$

Es gibt unendlich viele Primzahlen. Da für die Verschlüsselung später große Primzahlen benötigt werden, brauchen wir eine Möglichkeit, Primzahlen zu finden. Eine davon ist das Sieb des Eratosthenes.

Sieb des Eratosthenes

Das Sieb des Eratosthenes ist nach dem Mathematiker Eratosthenes von Kyrene benannt, der es allerdings nicht erfand, sondern erstmals als „Sieb“ bezeichnete.

Um die Primzahlen zu finden, werden zuerst alle Zahlen von 2 bis n aufgeschrieben. Hierbei ist n die höchste Zahl, bei der geprüft wird, ob sie eine Primzahl ist.

Man beginnt damit, alle Vielfachen der 2 zu streichen: die $4, 6, 8, \ldots$ Nun ist die kleinste nicht durchgestrichene Zahl immer eine Primzahl. Die nächste Primzahl ist also 3. Danach streicht man alle Vielfachen der 3 und erhält

die nächst größere Primzahl, die 5. Dies macht man immer so weiter, bis man bei n ankommt.

2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19
~~20~~	21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~
29	~~30~~	31	~~32~~	33	~~34~~	35	...	n

Sieb des Eratosthenes, nachdem alle Vielfachen von 2 gestrichen wurden

2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19
~~20~~	~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~
29	~~30~~	31	~~32~~	~~33~~	~~34~~	~~35~~	...	n

Sieb des Eratosthenes, bei dem alle Vielfachen der vorhandenen Zahlen gestrichen wurden. Es bleiben die Primzahlen übrig.

Da wir für eine sichere RSA-Verschlüsselung sehr große Zahlen brauchen, würde es jedoch zu lange dauern, bis man erst einmal die Zahlen aufgeschrieben hat. Daher ist das Sieb des Eratosthenes für die Kryptographie in unserer Chatanwendung nicht geeignet.

Eine weitere Möglichkeit, Zahlen auf ihre Teilbarkeit und somit auf die Eigenschaft der Primalität zu testen, ist die Probedivision bis $\sqrt{n}$ bzw. die Primfaktorzerlegung. Doch auch die Primfaktorzerlegung dauert bei großen Zahlen sehr lange. Um diese Zahlen auf ihre Teilbarkeit zu prüfen, gibt es deshalb verschiedene Primzahltests. Zwei davon haben wir im Kurs behandelt.

Fermat-Primzahltest

Der kleine Satz von Fermat besagt: Wenn p eine Primzahl ist, gilt für alle $b \in N$: $b^p \equiv b \mod p$. Das heißt, wenn wir eine Zahl b finden, für die $b^p \not\equiv b \mod p$ gilt, wissen wir mit Sicherheit, dass p keine Primzahl ist. Falls $b^p \equiv b \mod p$, so gibt es zwei Möglichkeiten:

1. p ist eine Primzahl

2. p ist eine Fermat-Pseudoprimzahl zur Basis b. Hier ist p also nicht prim, besteht aber trotzdem diesen einzelnen Fermat-Test zur Basis b.

Beispiel: $5^6 \equiv 5 \mod 6$, aber da $5^6 = 15625$ und $15625 : 6 = 2604$ Rest 2 ($15625 \equiv 2 \mod 6$) ist 6 keine Primzahl. In dem Fall probiert man es weiter. Also: $5^7 \equiv 5 \mod 7$ und $5^7 = 78125$ und $78125 : 7 = 11160$ Rest 5 ($78125 \equiv 5 \mod 7$).
So haben wir die Vermutung, dass 7 eine Primzahl ist – falls nicht, ist 7 zumindest eine Fermat-Pseudoprimzahl zur Basis $b = 5$. Um sicher zu gehen, müssten wir den Test mit vielen Basen b wiederholen, da mit jeder erfolgreich bestandenen Basis die Wahrscheinlichkeit steigt, dass die getestete Zahl p prim ist.

Leider gibt es auch sogenannte Carmichael-Zahlen. Das sind natürliche, zusammengesetzte Zahlen, für die zu *jeder* Basis b gilt: $b^p \equiv b \mod p$.

Carmichael-Zahlen sind für die Kryptographie problematisch, da sie vom Fermat-Test fälschlicherweise als prim erkannt werden. Die gesamte Sicherheit der modernen Kryptographie basiert darauf, dass große zusammengesetzte Zahlen nicht faktorisiert werden können.

Zur Umgehung dieser Carmichael-Problematik haben wir noch einen zweiten Test im Kurs behandelt: den Miller-Selfridge-Rabin-Test. Dieser ist gegen die Schwächen des Fermat-Primzahltests immun. Allerdings ist der Miller-Selfridge-Rabin-Test im Rechenverfahren aufwändiger.

Größter gemeinsamer Teiler (ggT)

Der ggT ist der größte gemeinsame Teiler zweier Zahlen a und b. Er ist die größte natürliche Zahl, durch die sich zwei ganze Zahlen ohne Rest teilen lassen. Um den ggT zu berechnen, gibt es zwei Möglichkeiten, die im Folgenden vorgestellt werden – den euklidischen Algorithmus und die Primfaktorzerlegung.

a) Die Primfaktorzerlegung

Bei der Primfaktorzerlegung stellt man eine natürliche Zahl n als Produkt aus mehreren Primzahlen dar. Diese Primzahlen werden dann als Primfaktoren bezeichnet. Ein Primfaktor p muss also eine Primzahl sein und die natürliche Zahl n ohne Rest teilen.

Beispiel 1:

$$a = 1000 = 2^3 \cdot 3^0 \cdot 5^3$$
$$b = 6 = 2^1 \cdot 3^1 \cdot 5^0$$

Für den ggT verwendet man die Primfaktoren, die in beiden Zahlen vorkommen und als dazugehörigen Exponenten verwendet man jeweils den kleineren Ausgangsexponenten.

$$ggT(1000, 6) = 2^1 \cdot 3^0 \cdot 5^0 = 2$$

Beispiel 2:

$$a = 3528 = 2^3 \cdot 3^2 \cdot 7^2$$
$$b = 3780 = 2^2 \cdot 3^3 \cdot 5^1 \cdot 7^1$$
$$ggT(3528, 3780) = 2^2 \cdot 3^2 \cdot 7^1 = 252$$
$$ggT(3528, 3780) = 252$$

Dieses Verfahren ist allerdings sehr langsam, und je größer die Zahlen werden, desto schwieriger wird es, die Primfaktoren zu finden.

b) Der Euklidische Algorithmus

Der Euklidische Algorithmus ist ein Algorithmus aus der Zahlentheorie und nach dem griechischen Mathematiker Euklid von Alexandria benannt. Man rechnet hierbei in jedem Schritt a modulo b, wobei der dabei entstehende Rest immer das neue b wird und das alte b immer das neue a. Man rechnet jetzt so lange a modulo b, bis der Rest 0 beträgt. Dieses b, bei dem der Rest 0 beträgt, ist dann der größte gemeinsame Teiler von a und b.

a	b	Rechenschritt
99	78	99:78 = 1 Rest 21
78	21	78:21 = 3 Rest 15
21	15	21:15 = 1 Rest 6
15	6	15:6 = 2 Rest 3
6	3	6:3 = 2 Rest 0
3	0	fertig! – $ggT(99, 78) = 3$

Berechnung des ggT von 99 und 78 mit dem Euklidischen Algorithmus. Das Ergebnis ist 3, was mit dem Algorithmus schnell und einfach berechnet werden kann.

Der erweiterte Euklidische Algorithmus

Der erweiterte Euklidische Algorithmus ist eine Ergänzung des Euklidischen Algorithmus, welcher neben dem ggT zweier Zahlen a und b auch noch zwei Zahlen x und y berechnen kann. x und y müssen hierbei die sogenannte Euler-Identität $x \cdot a + y \cdot b = ggT(a, b)$ erfüllen.

Beispiel:

a	b	q	ggT	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	-	3	1	0

Tabelle des Erweiterten Euklidischen Algorithmus

a und b werden wie beim Euklidischen Algorithmus berechnet, also a modulo b bis der Rest 0 ist.

Die Zahl in der Spalte mit der Variablen q zeigt uns, wie oft b in a hinein passt. Zum Beispiel passt 21 dreimal in 78 hinein.

Die Spalte mit dem ggT sagt uns den größten gemeinsamen Teiler der Zahlen a und b. Wenn man nur x und y berechnen will, kann man diese Spalte auch weglassen.

x und y sind die Zahlen, die wir mit dem erweiterten Euklidischen Algorithmus berechnen wollen. Hierfür beginnen wir in der untersten Zeile und schauen, mit welcher Zahl wir a und b multiplizieren müssen, damit die Gleichung erfüllt wird. Wenn wir die unterste Zeile eingetragen haben, können wir die nächste Zeile berechnen. Das alte y wird zu dem neuen x. $x_{\text{neu}} = y_{\text{alt}}$. Um unser neues y zu erhalten, rechnen wir:

$$y_{\text{neu}} = x_{\text{alt}} - q_{\text{neu}} \cdot y_{\text{alt}}$$

Im Beispiel, in der zweiten Zeile von unten, wäre das:

$$1 = 1 - 2 \cdot 0$$

So geht man von unten nach oben vor, bis die Tabelle fertig ausgefüllt ist. Den erweiterten Euklidischen Algorithmus verwendet man beispielsweise, um ein Schlüsselpaar der asymmetrischen Verschlüsselung zu berechnen.

Phi-Funktion

Die eulersche Phi-Funktion φ gibt für eine natürliche Zahl n die Anzahl aller zu n teilerfremden Zahlen an, die nicht größer als n sind. Teilerfremd bedeutet, dass der ggT von zwei natürlichen Zahlen a und b die Zahl 1 beträgt.

Beispiel: Wir möchten $\varphi(6)$ bestimmen. Dazu müssen wir jeweils den ggT(1,6), ggT(2,6), ... ggT(6,6) bestimmen.

$$ggT(1,6) = 1$$
$$ggT(2,6) = 2$$
$$ggT(3,6) = 3$$
$$ggT(4,6) = 2$$
$$ggT(5,6) = 1$$
$$ggT(6,6) = 6$$

Jetzt zählt man, wie viele von diesen „Paaren“ teilerfremd, also ggT(x,6) = 1, sind. Ergebnis: $\varphi(6) = 2$ (1 und 5)

Phi-Funktion und Primzahlen

Eine Primzahl p ist nur durch sich selbst und 1 teilbar, also ist sie zu allen Zahlen von 1 bis p teilerfremd. Folglich gilt: $\varphi(p) = p - 1$

Beispiel: Wir möchten $\varphi(5)$ bestimmen. Dazu müssen wir wieder jeweils den ggT(1,5), ggT(2,5), ... ggT(5,5) bestimmen.

$$ggT(1,5) = 1$$
$$ggT(2,5) = 1$$
$$ggT(3,5) = 1$$
$$ggT(4,5) = 1$$
$$ggT(5,5) = 5$$

Auch hier zählt man wieder, wie viele „Paare” teilerfremd sind. Ergebnis: $\varphi(5) = 4$ (1 bis 4)

Dies könnten wir auch so berechnen:

$$\varphi(5) = 5 - 1 = 4$$

Multiplikativität

Die φ-Funktion ist multiplikativ, es gilt also:

$$\varphi(a \cdot b) = \varphi(a) \cdot \varphi(b)$$

Beispiel:

$$\varphi(30) = \varphi(5) \cdot \varphi(6) = 4 \cdot 2 = 8$$

Die 8 zu 30 teilerfremden Zahlen sind $1, 7, 11, 13, 17, 19, 23, 29$. Alle anderen Zahlen haben mit 30 einen ggT > 1.

Satz von Euler

Der Satz von Euler sagt aus: Wenn m und n teilerfremd sind, dann gilt:

$$m^{\varphi(n)} \equiv 1 \mod n$$

Wir benötigen diese Aussage später, um zeigen zu können, dass die von uns verwendete RSA-Verschlüsselung tatsächlich funktioniert.

RSA

Helene Hellstern, Leontine Franz, Jannis Welsche

Das RSA-Kryptosystem, benannt nach den Mathematikern Rivest, Shamir und Adleman, ist ein in den 1970er Jahren entwickeltes asymmetrisches Verschlüsselungsverfahren. Asymmetrisch heißt, man hat zwei Schlüssel, die sich gegenseitig aufheben – ein Schlüsselpaar.
Damit später auch niemand unsere Nachrichten lesen kann, haben wir uns entschieden, dieses Verfahren in unserer Chatanwendung einzusetzen.

RSA-Schlüsselerzeugung

Um mit RSA arbeiten zu können, braucht man ein Schlüsselpaar. Der öffentliche Schlüssel wird e, der private Schlüssel wird d genannt. Um diese Schlüssel zu erzeugen, müssen die im Folgenden vorgestellten Schritte durchgeführt werden.

Zunächst benötigen wir zwei beliebige, aber unterschiedliche Primzahlen p und q. Wir nehmen zum Beispiel die Primzahlen $p = 5$ und $q = 13$. Aus diesen berechnen wir den Modulus N und $\varphi(N)$.

N berechnen wir, indem wir p mit q multiplizieren. Ohne unseren Modulus N wäre unsere verschlüsselte Zahl viel zu groß und unsere Schlüssel e und d würden sich nicht gegenseitig aufheben, wodurch unsere Verschlüsselung nicht mehr funktionieren würde. Mit unseren

Primzahlen aus dem Beispiel gerechnet ergibt das: $N = 5 \cdot 13 = 65$.

Da p und q Primzahlen sind, lässt sich $\varphi(N)$ mit Hilfe der Multiplikativität von φ leicht berechnen. Es gilt:

$$\begin{aligned}\varphi(N) &= \varphi(p \cdot q) \\ &= \varphi(p) \cdot \varphi(q) \\ &= (p-1) \cdot (q-1).\end{aligned}$$

In unserem Beispiel:

$$\varphi(N) = (5-1) \cdot (13-1) = 48.$$

Unseren öffentlichen Schlüssel e können wir frei wählen, mit der einen Bedingung, dass der ggT von e und $\varphi(N)$ gleich 1 sein muss, das heißt, dass e teilerfremd zu $\varphi(N)$ sein muss.

In unserem Beispiel wählen wir 5, da 5 eine der Zahlen ist, die teilerfremd zu 48 sind. Man könnte aber beispielsweise auch 11 oder 13 nehmen.

Nachdem wir e gewählt haben, können wir mit dem erweiterten Euklidischen Algorithmus den Schlüssel d bestimmen, sodass am Ende gilt: $e \cdot d = 1 \mod \varphi(N)$. Dabei ist d das Moduloinverse zu e. d wird so bestimmt, dass $e \cdot d$ mod $\varphi(N)$ gerechnet 1 ergibt. Das Moduloinverse von 5 ist in diesem Fall 29. Also wenn man $5 \cdot 29 \mod 48$ rechnet, erhält man 1. Zusammengefasst ergibt das in unserem Beispiel:

$$\begin{aligned}p &= 5 \\ q &= 13 \\ N &= 65 \\ \varphi(N) &= 48 \\ e &= 5 \\ d &= 29\end{aligned}$$

Für die eigentliche Verschlüsselung benötigen wir aber die Primzahlen p und q und $\varphi(N)$ nicht, daher können wir sie löschen. Das sollten wir unbedingt tun, denn wenn jemand diese Zahlen wissen würde, könnte man damit unsere Verschlüsselung knacken. Also bleibt übrig:

$$\begin{aligned}N &= 65 \\ e &= 5 \\ d &= 29\end{aligned}$$

Jetzt haben wir unseren öffentlichen Schlüssel e, unseren privaten Schlüssel d und unseren Modulus N. Dieser ist sowohl Teil des öffentlichen als auch des privaten Schlüssels. Nun können wir mit der eigentlichen Verschlüsselung beginnen.

RSA-Verschlüsselung

Bevor man einen Text überhaupt verschlüsseln kann, muss man ihn erst in eine Zahl umwandeln. Das haben wir bereits in den Grundlagen erklärt. Wir gehen hier von einer zu verschlüsselnden Zahl m aus. Diese verschlüsselt man mit dem öffentlichen Schlüssel e und dem Modulus N folgendermaßen: $m^e \equiv c \mod N$.

Die dabei berechnete Zahl c stellt die verschlüsselte Nachricht dar. Diese kann nun gefahrlos über Dritte übertragen werden, da der Klartext ohne den privaten Schlüssel d nicht berechnet werden kann.

Beim Empfänger angekommen, wird die verschlüsselte Nachricht mithilfe des privaten Schlüssels d und dem Modulus N wie folgt entschlüsselt: $c^d \equiv m \mod N$

Beispiel: Der zu verschlüsselnde Text ist ein ?, als Zahl ist das eine 63. Die Schlüssel und der Modulus sind dieselben wie im Beispiel der Schlüsselerzeugung:

$$\begin{aligned}e &= 5 \\ d &= 29 \\ N &= 65\end{aligned}$$

Jetzt rechnet man $m^e \mod N$, also $63^5 \equiv 33 \mod 65$. Das ergibt unsere verschlüsselte Nachricht $c = 33$.

Um m wieder zu erhalten, rechnet man $c^d \mod N$, also $33^{29} \mod 65$. Das Ergebnis ist 63, also haben wir wieder unsere ursprüngliche Nachricht m.

Die Verschlüsselung beruht darauf, dass sich e und d gegenseitig aufheben. Dies kann man mit dem Satz von Euler und der Euler-Identität beweisen.

Die Euler-Identität besagt, dass der ggT zweier Zahlen a und b durch die Addition ganzzahliger Vielfacher von a und b berechnet werden kann: $x \cdot a + y \cdot b = ggT(a, b)$. x und y können durch

den erweiterten Euklidischen Algorithmus berechnet werden.

Zum Beweis der Funktion von RSA nutzt man diese Euler-Identität und setzt dabei $a = e$ sowie $b = \varphi(N)$. Außerdem benennt man die anderen Bestandteile der Euler-Identität um: $x = d$, $y = -k$. Die veränderte Euler-Identität lautet also $e{\cdot}d - k{\cdot}\varphi(N) = ggT(e, \varphi(N))$. Da wir e so gewählt haben, dass es teilerfremd zu $\varphi(N)$ ist, gilt: $e{\cdot}d - k{\cdot}\varphi(N) = 1$. k und d lassen sich mit dem Erweiterten Euklidischen Algorithmus berechnen, und wir können die Gleichung umformen zu $e{\cdot}d = 1 + k{\cdot}\varphi(N)$.

Damit können wir den Beweis beginnen. Wir möchten zeigen, dass $m^{e^d} \equiv m \mod N$. Die Verwendung der Euler-Identität ist mit (EI) gekennzeichnet, der Satz von Euler mit (SvE).

$$
\begin{aligned}
(m^e)^d &\equiv m^{e\cdot d} \mod N \\
&\overset{(EI)}{\equiv} m^{1+k\cdot\varphi(N)} \mod N \\
&\equiv m^1 \cdot m^{k\cdot\varphi(N)} \mod N \\
&\equiv m \cdot m^{\varphi(N)\cdot k} \mod N \\
&\equiv m \cdot (m^{\varphi(N)})^k \mod N \\
&\overset{(SvE)}{\equiv} m \cdot 1^k \mod N \\
&\equiv m \cdot 1 \mod N \\
&\equiv m \mod N
\end{aligned}
$$

$q.\,e.\,d.$

Sicherheit von RSA

Nun stellt sich natürlich die Frage, wieso die RSA-Verschlüsselung überhaupt sicher ist. Wird eine Nachricht mit einem der zwei Schlüssel verschlüsselt, ist sie nur mit dem Gegenschlüssel entschlüsselbar. Nur wie erzeugt man überhaupt den Gegenschlüssel? Zuerst braucht man den erweiterten euklidischen Algorithmus, für den man wiederum die zwei Ursprungsprimzahlen der Verschlüsselung benötigt. Sie sind das Kettenglied, das für eine tatsächliche Verschlüsselung benötigt wird. Der Vorteil hierbei ist, dass p und q aus dem Modulus N mit Hilfe der Primfaktorzerlegung nur sehr schwer zurückrechenbar sind.

Die Primfaktorzerlegung ist zwar langwierig, bei „kleinen“ Zahlen bis 20 Stellen schafft ein normaler Computer das allerdings noch in absehbarer Zeit. Sind die Primzahlen jetzt aber zum Beispiel 75 Stellen lang und der Modulus N deshalb 150 Stellen oder noch größer, dann würde die Primfaktorzerlegung unheimlich lange dauern. Hintergrund ist, dass man für die Faktorisierung alle Zahlen bis $\sqrt{N}$ durchprobieren muss.

N	$\sqrt{N}$	Dauer
10^{12}	10^6	1 Sekunde
10^{20}	10^{10}	10^4 Sek. = 3 Stunden
10^{40}	10^{20}	10^{14} Sek. = 3 Mio. Jahre
10^{150}	10^{75}	10^{69} Sek. = 10^{61} Jahre

Rechendauer für die Primfaktorzerlegung bei 1.000.000 Tests pro Sekunde

Es gibt aber natürlich auch Schwachstellen. Wenn beispielsweise die Ursprungszahlen p und q keine Primzahlen sind, kann die Primfaktorzerlegung enorm verkürzt werden. Außerdem dauert die RSA-Verschlüsselung mit zunehmender Stellenzahl zunehmend länger, was dazu führt, dass die Stellenanzahl von p und q teilweise gezielt „kurz“ gehalten wird, um eine schnelle Kommunikation zu ermöglichen.

Alles in allem ist die RSA-Verschlüsselung sehr sicher, da eine Entschlüsselung ohne den passenden Schlüssel praktisch unmöglich ist.

Chat

Jascha Fricker, Máté Makszi

Unser Kursziel bestand darin, einen funktionierenden Chat zu programmieren. Dazu benötigten wir alle beschriebenen Grundlagen und Verschlüsselungensarten. Der Chat sollte vergleichbar mit Vorbildern aus dem Internet sein. Deshalb lag unser Augenmerk darauf, unsere Anwendung möglichst sicher, aber auch einfach zu gestalten. Komplex sind bei unserer Chatanwendung eher die Hintergrundprozesse, so haben wir zum Beispiel eine sichere Verschlüsselung programmiert.

Registrierung

Bevor man unsere Chatanwendung benutzen kann, muss man sich zunächst registrieren (sie-

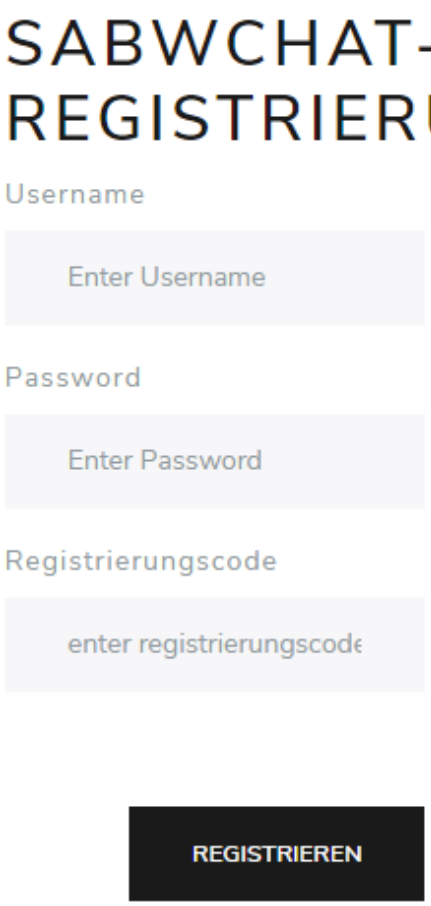


Abbildung 8: Registrierungsseite

he Abbildung 8), damit die Daten auf dem Server hinterlegt werden können.

Bei der Registrierung gibt der Benutzer einen Benutzernamen und ein Passwort an. Der Benutzername wird im Klartext auf dem Server gespeichert. Damit dieser Server nicht das Passwort des Benutzers kennt, wird das Passwort vor dem Hinterlegen im Server mit einem sogenannten Hashing-Algorithmus gehasht. Beim Hashing wird das Passwort des Benutzers durch einen speziellen Algorithmus in eine Zeichenkette umgewandelt. Diese Umwandlung hat die Eigenschaft, dass man sie nicht zurück rechnen kann. Aus dem Hash kann also nicht das Passwort im Klartext errechnet werden.

Der folgende TypeScript-Code zeigt Passwortverhashung mit dem MD5-Algorithmus.

```
this.passwordhash
= Md5.hashStr(this.password)
```

Außerdem wird beim Registrieren für den Benutzer ein öffentlicher und ein privater Schlüssel für die RSA-Verschlüsselung erzeugt. Der öffentliche Schlüssel, bestehend aus e und N, kann direkt auf dem Server gespeichert werden, da dieser öffentlich bekannt sein darf.

Doch wie der Name schon sagt, darf der private Schlüssel d nur dem Benutzer bekannt sein. Trotzdem muss er irgendwo hinterlegt werden. Da wir uns den privaten Schlüssel nicht persönlich merken können, muss er auf dem Server gespeichert werden.

Dies darf jedoch nicht unverschlüsselt passieren, da der Server sonst alle privaten Schlüssel kennt und die RSA-Verschlüsselung sinnlos wäre. Daher haben wir uns entschieden, den privaten Schlüssel mit dem Passwort des Benutzers mithilfe der symmetrischen AES-Verschlüsselung zu verschlüsseln.

Somit wird der private Schlüssel d beim Registrieren mit dem eingegebenen Passwort verschlüsselt und kann gefahrlos auf dem Server gespeichert werden. Durch diese Methode kann sich der Benutzer von unterschiedlichen Geräten anmelden, ohne dass Dritte den privaten Schlüssel kennen.

Der folgende Code zeigt die Verschlüsselung des eigenen Passwortes mit AES

```
this.verschluesseltesd =
this.aesVerschlüsseln(d.toString(),
this.password)
```

Zudem benötigt man bei der Registrierung für unsere Chatanwendung einen bestimmten Registrierungscode. Dieser ist erforderlich, da die Chatanwendung online gestellt wurde und wir nicht möchten, dass sich jeder in unserer Chatanwendung registrieren kann.

Login

Beim Login gibt der Benutzer seinen bereits gewählten Benutzernamen und sein Passwort ein (siehe Abbildung 9). Das eingegebene Passwort wird mit MD5 verhasht und in Kombination mit dem Benutzernamen an den Server geschickt. Der Server gleicht nun den Benutzernamen und das verhashte Passwort mit den bereits hinterlegten Daten ab. Bei Übereinstimmung erhält der Benutzer Zugang zur Chatanwendung.

Der private Schlüssel, der verschlüsselt auf dem Server hinterlegt ist, wird dem Benutzer gesendet, welcher ihn mit seinem eigenen Passwort und der AES-Verschlüsselung wieder errechnen kann. Stimmen die eingegebenen Zugangsdaten nicht überein, wird kein Zugang erteilt und es wird eine Fehlermeldung angezeigt.

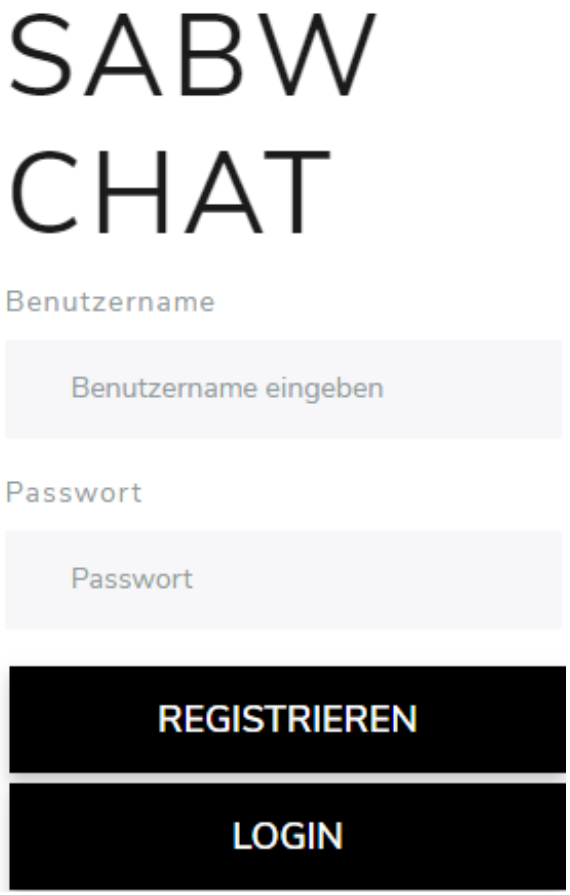


Abbildung 9: Login-Seite

Benutzerübersicht

Nachdem der Zugang vom Server erteilt wurde, wird der Benutzer auf die Benutzerübersicht geleitet (siehe Abbildung 10). Hier werden alle in der Chatanwendung registrierten Benutzer angezeigt. Bei unserer Chatanwendung werden die Benutzer nach Alphabet sortiert und dem Benutzer gegliedert angezeigt.

In diesem Code (Angular-HTML) wird mithilfe einer for-Schleife alle Benutzer über einen jeweiligen Button in der Benutzerliste angezeigt:

```
<div class="col-sm-12 col-md-4"
 *ngFor="let b of benutzer">
```

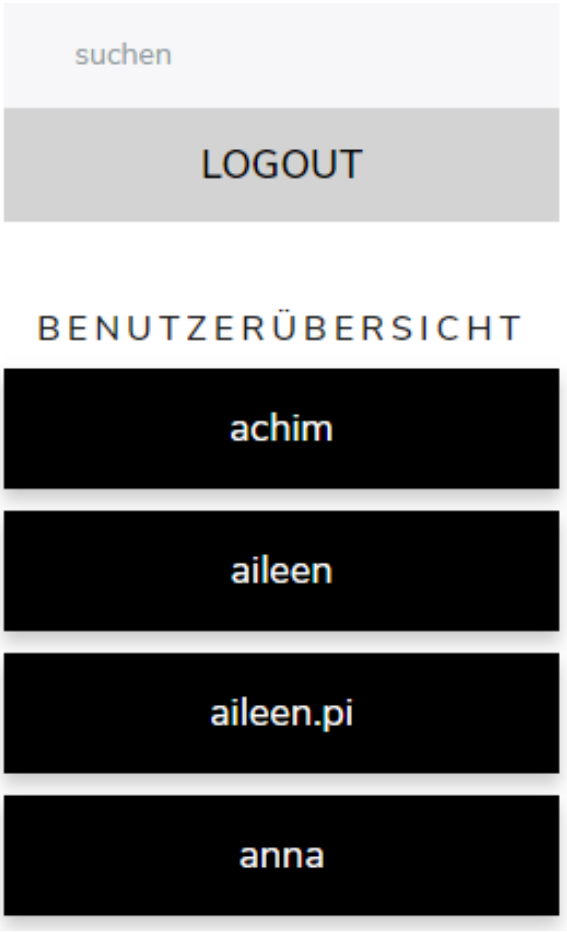


Abbildung 10: Benutzerübersicht

```
  <button(click)="benutzerAngeklickt(b)"
   class="benutzerButton">
      {{b.username}}
  </button>
</div>
```

Auch ist ein Suchfeld integriert, bei dem nach Übereinstimmungen zwischen den Namen der verschiedenen Benutzer und den eingegebenen Buchstaben gesucht wird, damit einem die Suche nach einem bestimmten Chatpartner erleichtert wird und man nicht ewig mit Suchen beschäftigt ist, falls es sehr viele Benutzer geben sollte. Bei Übereinstimmung werden die passenden Ergebnisse angezeigt, wobei die restlichen, unpassenden Benutzer ausgeblendet werden.

Beim Anklicken eines Benutzers wird man in das jeweilige Chatfenster weitergeleitet (siehe Abbildung 11), in dem Nachrichten ausgetauscht werden können. Außerdem befindet sich in der Benutzerübersicht noch ein Logout-Button, dieser löscht die zwischengespeicherten

Daten des Benutzers und leitet zur Login-Seite zurück.

Chat

Abbildung 11: Chat-Seite

In unserem Chat werden bereits gesendete Nachrichten des Benutzers rechts und bereits empfangene Nachrichten links angezeigt. Wechselt man in den Chat, werden die letzten 20 Nachrichten aus der Konversation vom Server geladen und angezeigt.

Beim Schreiben einer neuen Nachricht hat der Benutzer die Möglichkeit, Buchstaben und Sonderzeichen, sowie Smileys zu schreiben. Hierbei hat der Benutzer aber eine Begrenzung von 63 Byte. Ein Buchstabe belegt ein Byte, ein Sonderzeichen zwei und ein Smiley vier. Da bei einer zu großen Byteanzahl die Länge des Entschlüsselungprozesses zu lang wäre, haben wir eine Begrenzung von 63 Byte gesetzt. Deshalb ist es bei einer größeren Byteanzahl nicht möglich, die Nachricht abzusenden.

Zunächst muss, wie in den Grundlagen beschrieben, der Text in Zahlen umgewandelt werden. Dieser TypeScript-Code wurde für die Umwandlung von Nachrichten in Zahlen programmiert. Er ist das Gegenstück zur Funktion „TextZuZahl“.

```
zahlZuText(zahl: BigInteger): string {
    let zeichen: number[] = [];
    let i = 0;
    while (zahl.notEquals(0)) {
      const s = zahl.mod(256);
      zeichen[i] = s.toJSNumber();
      zahl = zahl.divide(256);
      i = i + 1; }
    zeichen = zeichen.reverse();
    return FromUTF8Array(zeichen);
}
```

Vor dem Versenden wird diese Zahl mit dem öffentlichen Schlüssel des Empfängers, den wir vom Server erhalten haben, verschlüsselt. So verhindern wir das Mitlesen, da nur der Empfänger den Schlüssel zum Entschlüsseln der Nachricht besitzt.

Der Empfänger erhält diese Nachricht mit einer kleinen Verzögerung, da der Server nur in bestimmten Zeitintervallen, in unserem Fall 1 Mal pro Sekunde, nach neuen Nachrichten abgefragt wird. Nun entschlüsselt der Empfänger die erhaltene Zahlenkombination nur mit seinem privaten Schlüssel und kann den erhaltenen Klartext im Chatfenster anzeigen.

Falls wir ausschließlich mit dem öffentlichen Schlüssel des Empfängers die Nachricht verschlüsseln würden, könnte der Benutzer nicht mehr auf seine bereits geschriebenen Nachrichten zugreifen. Deshalb wird die Nachricht zusätzlich mit dem öffentlichen Schlüssel des Senders verschlüsselt und auf dem Server hinterlegt. Nun kann der Sender diese Nachrichten beim Einloggen abrufen, indem er sie mit seinem privaten Schlüssel entschlüsselt. Die Nachricht wird also zweimal verschlüsselt und in beiden Versionen gespeichert.

Insider aus der Kursarbeit

- Jerôme, das Dreieck
- „Wir haben eine Endlosschleife programmiert“ – „Ja, aber nur eine kurze“
- „Das Alphabet hat 25 Buchstaben“
- „Caesar im Mittelalter“
- Siegestanz Uh uh uh uh
- Dackel Waldemar, ich bin ein Chicken

- „Es war zwar kein Wettbewerb, aber wir haben gewonnen“
- „Das ist ja toll! Das Skript hat eine Spannungskurve!“ – Michael begeistert über das Skript über Primzahltests
- „Raus, Treppe runter – das einzige was man sich zum Fluchtweg merken muss“
- „Oh Antonia, kannst du mal runter gehen? Ich habe ein Buch aus dem Fenster fallen lassen“ – Kevin, nachdem das erste von zwei Büchern dem Gebrauch als Fensterhalter nicht standgehalten hat
- „Wir sind Informatiker! Wir kopieren!“ – nachdem wir den Schülermentor eines anderen Kurses mit allen Namensschildern beim Sportfest entdeckten und einen kurzen Moment später Antonia auch alle Namensschilder am Arm hatte
- Jannis schreibt schräg und lang gezogen an der Tafel. Michael zu Kevin: „Wieso ist die Schrift an der Tafel so lang gezogen?“ Kevin: „Er hat halt die falsche Schriftart eingestellt“.
- „ng serve!“
- „kdoor“
- „Jascha, kannst du mal bitte das Whiteboard putzen?“

Unsere „Auffahrt“ zum 2. Platz im Sportfest

Danksagung

Wir möchten uns an dieser Stelle bei denjenigen herzlich bedanken, die die 16. JuniorAkademie Adelsheim / Science Academy Baden-Württemberg überhaupt möglich gemacht haben.

Finanziell wurde die Akademie in erster Linie durch die Stiftung Bildung und Jugend, die Hopp-Foundation, den Förderverein der Science Academy sowie durch den Fonds der Chemischen Industrie unterstützt. Dafür möchten wir an dieser Stelle allen Unterstützern ganz herzlich danken.

Die Science Academy Baden-Württemberg ist ein Projekt des Regierungspräsidiums Karlsruhe, das im Auftrag des Ministeriums für Kultus, Jugend und Sport Baden-Württemberg und mit Unterstützung der Bildung & Begabung gGmbH Bonn für Jugendliche aus dem ganzen Bundesland realisiert wird. Wir danken daher Frau Anja Bauer, Abteilungspräsidentin der Abteilung 7 – Schule und Bildung des Regierungspräsidiums Karlsruhe, der Leiterin des Referats 75 – allgemein bildende Gymnasien, Frau Leitende Regierungsschuldirektorin Dagmar Ruder-Aichelin, Herrn Jan Wohlgemuth vom Ministerium für Kultus, Jugend und Sport Baden-Württemberg sowie dem Koordinator der Deutschen Schüler- und JuniorAkademien in Bonn, Herrn Volker Brandt, mit seinem Team.

Wie in jedem Jahr fanden die etwas über einhundert Gäste sowohl während des Eröffnungswochenendes und des Dokumentationswochenendes als auch während der zwei Wochen im Sommer eine liebevolle Rundumversorgung am Eckenberg-Gymnasium mit dem Landesschulzentrum für Umwelterziehung (LSZU) in Adelsheim. Stellvertretend für alle Mitarbeiterinnen und Mitarbeiter möchten wir uns für die Mühen, den freundlichen Empfang und den offenen Umgang mit allen bei Herrn Oberstudiendirektor Meinolf Stendebach, dem Schulleiter des Eckenberg-Gymnasiums, besonders bedanken.

Ein herzliches Dankeschön geht auch an Frau Oberstudiendirektorin Dr. Andrea Merger vom Hölderlin-Gymnasium in Heidelberg, wo wir bei vielfältiger Gelegenheit zu Gast sein durften.

Zuletzt sind aber auch die Kurs- und KüA-Leiter gemeinsam mit den Schülermentoren und der Assistenz des Leitungsteams diejenigen, die mit ihrer hingebungsvollen Arbeit das Fundament der Akademie bilden.

Diejenigen aber, die die Akademie in jedem Jahr einzigartig werden lassen und die sie zum Leben erwecken, sind die Teilnehmerinnen und Teilnehmer. Deshalb möchten wir uns bei ihnen und ihren Eltern für ihr Engagement und Vertrauen ganz herzlich bedanken.

Bildnachweis

Seite 30, Abbildung Gasgenerator:
kfztech.de (mit freundlicher Genehmigung)

Seite 101, Abbildung 1:
https://commons.wikimedia.org/wiki/File:E-30-Cutmodel.jpg
Wikimedia-User: Hanabi123, Bearbeitungen: Mika Alkabetz
CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/legalcode)

Seite 103, Abbildung 5:
https://commons.wiki media.org/wiki/File:Shutter_priority_mode.svg
Wikimedia-User: Athepan, Bearbeitungen: Mehdi
CC BY-SA 2.5 (https://creativecommons.org/licenses/by-sa/2.5/legalcode)

Seite 108, Abbildung 12:
https://commons.wikimedia.org/wiki/File:CMY_ideal_version_rotated.svg
Gemeinfrei
https://commons.wikimedia.org/wiki/File:Synthese+.svg
Wikimedia-User: Quark67
CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/legalcode)

Seite 109, Abbildung 13:
https://commons.wikimedia.org/wiki/File:HSV_cone.png
Wikimedia-User: (3ucky(3all
CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/legalcode)

Seite 114, Abbildung 23:
https://commons.wikimedia.org/wiki/File:Hexacyanidoferrat(II).svg
Wikimedia-User: Ilgom und Muskid
Gemeinfrei

Alle anderen Abbildungen sind entweder gemeinfrei oder eigene Werke.