# Mathematische Betrachtung des Diffie-Hellman Algorithmus visualisiert mit einer selbst entwickelten Java-Anwendung

Von

Alexander Niedrig

Im Informatik LK 2017 bei

Herrn Rolf Faßbender

# Inhaltsverzeichnis

# Einleitung

Seit Jahrhunderten ist das übermitteln von Botschaften ohne das Mitwissen dritter vor allem im militärischen, aber auch im privaten Bereich, viel gefragt. Zunächst war die einzige Möglichkeit eine Nachricht sicher zu übermitteln, sie von einem Boten auswendig lernen zu lassen. Mit den ersten Hochkulturen kamen jedoch bald auch die ersten Verfahren, eine Nachricht nur für jene lesbar zu machen, für die sie bestimmt war. Die ersten Kryptosysteme-wie die Caesar Verschlüsslung waren einfache Substitutionsalgorithmen, deren Sicherheit auf der Unwissenheit Dritter beruhte. Jedoch hielt die Unwissenheit nicht an und über die Zeit wurden neue Verfahren entwickelt, jedoch ließen sich diese mittels Häufigkeitsanalyse knacken. Im Jahre 1586 veröffentlichte Blaise de Vigenère sein eigenes Verschlüsselungsverfahren, die Vigenère Verschlüsslung. Diese war das erste polyalphabetische Verfahren und als ein solches immun gegen Häufigkeitsanalysen. Jedoch ließ sich auch diese-bei ausreichend Text-knacken. 1854 gelang es Charles Babbage die Verschlüsslung zu knacken. Durch den Abstand zwischen Bigrammen (oder auch längeren Buchstabenkombinationen) ließ sich die Länge des Schlüsselwortes erschließen und damit der verschlüsselte Text so aufteilen, dass man erneut eine Häufigkeitsanalyse anwenden konnte. Mit der Zeit kamen weitere Verschlüsselungsverfahren auf und einige wurden bis heute nicht geknackt-wie zum Beispiel der Doppelwürfel. In den Jahren des ersten und zweiten Weltkrieges gewann das Verschlüsseln von Nachrichten-und das Knacken der Verschlüsselungen-an immenser Bedeutung. Die Kryptologie wurde zunächst mechanisiert, gegen Ende des zweiten Weltkrieges digitalisiert und sorgte für den Bau der ersten Computer. Ein Problem blieb jedoch seit dem ersten Tag der Kryptologie-der Schlüsselaustausch. Dieser musste immer im geheimen und ohne das Mitwissen anderer erfolgen, um eine sichere Übermittlung späterer Nachrichten zu gewährleisten. [1]

*„Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus abhängen. Die Sicherheit gründet sich nur auf die Geheimhaltung des Schlüssels.“ - Auguste Kerkhoffs, La Cryptographie militaire* [1]

Persönliches Interesse im Bereich Kryptologie bestand schon immer. Durch Vorschlage meines Informatiklehrers Herr Faßbender habe ich mich für Diffie-Hellman entschieden, da es aktuell ist. Die andere mögliche Wahl, RSA, ging an einen Mitschüler. Der Diffie-Hellman Algorithmus dient dem Austausch eines geheimen Schlüssels über ein öffentliches Netzwerk, welcher dann für eine symmetrische Verschlüsselung verwendet werden kann. Der Algorithmus wurde 1976 von Whitfield Diffie und Martin Hellman basierend auf der Arbeit von Ralph Merkle entwickelt und gründete somit die Public Key Kryptologie. Heutzutage wird der Diffie-Hellman Algorithmus z.B. noch für TLS/SSL verwendet.

# 1: Der Diffie-Hellman Schlüsselaustausch

## 1.1: Der Schlüsselaustausch in der analogen Welt

### 1.1.1: Das verschicken einer gesicherten Kiste

Vor der Betrachtung des digitalen Schlüsselaustausches muss erst die Idee hinter dem Schlüsselaustausch klarwerden. Zu Beginn stelle man sich zwei Personen vor-der Tradition halber hier Alice und Bob. Alice will Bob eine Box zuschicken, diese enthält jedoch private Dinge und soll somit gesichert verschickt werden. Alice verschließt die Kiste mit einem Schloss und schickt sie an Bob. Dieser hat jedoch keinen Schlüssel für das Schloss, kann jedoch sein eigenes anbringen. Nun schickt er die Kiste wieder zurück. Alice erhält die Kiste, entfernt ihr eigenes Schloss und schickt sie, nun nur noch mit Bobs Schloss versiegelt, wieder an Bob. Dieser kann das verbleibende Schloss mit seinem eigenen Schlüssel entfernen und gelangt an den Inhalt, während die Kiste zu keinem Zeitpunkt unverschlossen war. Somit ist das Mitwirken beider Seiten für die Übermittlung notwendig.[2]

### 1.1.2: Der Farbaustausch

Alice und Bob wollen eine geheime Farbe ausmachen. Sie können allerdings nicht privat kommunizieren und müssen sich irgendwie abhörsicher verständigen. Hierbei nehmen wir an, dass sich die Farben nicht trennen lassen. Sowohl Bob als auch Alice wählen im geheimen eine beliebige Farbe und machen öffentlich eine weitere Farbe aus. In diesem Beispiel die gemeinsame Farbe Gelb, Alice geheime Farbe Rot und Bobs geheime Farbe Blau. Sowohl Alice als auch Bob vermischen ihre geheime Farbe mit der öffentlichen und schicken ihr Ergebnis an den jeweilig anderen. Da wir davon ausgehen, dass sich die Farben nicht trennen lassen bleiben für alle Beteiligten die geheimen Farben unbekannt. Alice hat nun Bobs grünes Gemisch und Bob nun Alice orangenes. Die beiden mischen ihre eigenen geheimen Farben dazu und landen beide bei Grau. Diese gemeinsame Farbe hat die Formel Gelb+Rot+Blau. Sollte eine Person-der Tradition wegen hier Eve genannt-den Farbaustausch zwischen den beiden verfolgt haben, so kennt sie die gemeinsame Farbe Gelb und die bereits

gemischten Farben Orange und Grün, nicht jedoch die geheimen Farben Rot und Blau-zu mindestens solange sich die Kombination von Gelb und einer gemeinsamen Farbe nicht umkehren lassen. Unter dieser Voraussetzung lässt sich eine gemeinsame Kombination sicher über öffentliche Kommunikation austauschen. Allerdings muss eine nicht umkehrbare Funktion dafür verwendet werden.[3]

### 1.1.3: Die Anforderungen an eine mathematische Herangehensweise

Wie oben bereits bemerkt wird das Mitwirken beider Seiten in Form eines Schlüssels benötigt.
Sollte man versuchen die Idee aus 1.1.1 umzusetzen, so stößt man auf folgendes Problem: wenn ein Kryptoanalytiker alle Texte t abfängt so besitzt er die Kryptotexte $K_A$, $K_B$ und $K_{AB}$. Damit das System funktionieren kann, muss die Verschlüsselung symmetrisch sein, das bedeutet, dass $K_A$ = t verschlüsselt (im Folgenden mit v abgekürzt) $S_a$ ist, $K_B$ =t v $S_B$ und $K_{AB}$ = t v $S_a$ v $S_b$. Wenn man nun die Kryptotexte "addiert" bedeutet das:

$$K_A \text{ v } K_B \text{ v } K_{AB} = (t \text{ v } S_A) \text{ v } (t \text{ v } S_B) \text{ v } (t \text{ v } S_A \text{ v } S_B)$$

Damit das System überhaupt funktionieren kann muss das Assoziativgesetz und das Kommutativgesetzt gelten, da ansonsten das Entschlüsseln von Alice Verschlüsselung nicht $K_B$ liefern würde. Daher gilt:

$$(t \text{ v } S_A) \text{ v } (t \text{ v } S_B) \text{ v } (t \text{ v } S_A \text{ v } S_B) = t \text{ v } S_A \text{ v } t \text{ v } S_B \text{ v } t \text{ v } S_A \text{ v } S_B$$

$$= (S_A \text{ v } S_A) \text{ v } (S_B \text{ v } S_B) \text{ v } (t \text{ v } t) \text{ v } t$$

$$= 0 \text{ v } 0 \text{ v } 0 \text{ v } t$$

$$= t$$

Somit ist die Verschlüsslung nicht sicher. [2]

Wenn man nun allerdings wieder zur Idee aus 1.1.2 zurückkehrt so erkennt man folgendes: Beide Seiten erstellen einen eigenen, privaten, Schlüssel, kombinieren diesen mit einem gemeinsamen "Schlüssel" und schicken das Ergebnis an den jeweilig Anderen. Damit das Verfahren jedoch bis dahin sicher ist muss das Kombinieren des privaten Schlüssels mit der gemeinsamen Zahl g

nicht umkehrbar sein, da man nicht davon ausgehen kann, dass g etwaigen Analytikern unbekannt ist. Gesucht wird also eine Funktion die sich nicht umkehren lässt. Praktisch gesehen ist dies unmöglich, da sich alle Funktionen umkehren lassen, jedoch gilt eine Funktion schon als Einwegfunktion, wenn lediglich keine effiziente Umkehrfunktion existiert. So eine Funktion ist die Diskrete Exponentialfunktion deren Umkehrfunktion der diskrete Logarithmus ist. Somit sollte $g^{Sa}$ mod p (wobei Sa Alice' privater Schlüssel ist) nicht umkehrbar sein. Wenn nun beide Parteien dies durchführen und sich gegenseitig schicken besitzt Alice Sa und $g^{Sb}$ mod p während Bob Sb und $g^{Sa}$ mod p besitzt. Allerdings lässt sich nun schon erkennen, dass eine gemeinsame Zahl nicht ausreicht für eine Diskrete Exponentialfunktion, es wird noch eine weitere Zahl p benötigt.

## 1.2: Die Diskrete Exponentialfunktion

### 1.2.1: Das Lösen der diskreten Exponentialfunktion

Eine Möglichkeit die diskrete Exponentialgleichung zu lösen ist der Satz von Euler. Dieser lautet: $a^{\varphi(n)} \equiv 1\ (mod\ n)$ wobei $\varphi(n)$ die Anzahl der Zahlen ist die teilerfremd zu n sind. Allerdings muss dafür der größte gemeinsame Teiler(ggT) von a und n eins sein, d.h. sie müssen Teilerfremd sein. Dies ist genau dann der Fall, wenn n eine Primazahl ist und a kein Vielfaches von n ist. Da n von nun an eine Primzahl ist wird n von nun an mit p betitelt. Dies kann zur Vereinfachung einer diskreten Exponentialfunktion dienen. Die Basis die in dem Satz von Euler noch a genannt wird, wird nun schon einmal vorraugreifend g genannt, der Exponent heißt nun Sa und steht für den geheimen Schlüssel, welcher zufällig aus der Menge {1,…,p-1} erzeugt wird.

Beispiel: g=7, Sa=25, p=13

$7^{25} = 7 * 7^{24} = 7 * 7^{12^2}$ [Da $\varphi(13) = 12$ und $7^{\varphi(13)} \equiv 1\ (mod\ 13)$]

$= 7 * 1^2 = 7\ (mod\ 13)$

Abgesehen davon, dass die Wahl einer Primzahl für p das Ausrechnen von $\varphi(p)$ stark vereinfacht wird entsteht so auch die maximale Anzahl an Restgruppen für p, was bei gleichbleibendem Aufwand die Sicherheit massiv verstärkt. [18,19]

### 1.2.2: Das Erzielen eines gemeinsamen Schlüssels

Durch einfaches Anwenden der Diskreten Exponentialfunktion kommt man auf kein gemeinsames Ergebnis. Wie kommt man nun auf ein geteiltes Geheimnis? Ein erneutes Anwenden der diskreten Exponentialfunktion mit veränderter Basis ermöglicht dies. Wenn nun Alice Bobs öffentlichen Schlüssel SB anstatt g nimmt und Bob das gleiche mit Alice öffentlichem Schlüssel SA macht.

Alice Informationen=$SB^{Sa}\ mod\ p$

$$= (g^{Sb}\ mod\ p)^{Sa}\ mod\ p$$

$$= g^{Sb^{Sa}}\ mod\ p$$

$$= g^{Sa*Sb}\ mod\ p$$

$$= g^{Sa^{Sb}}\ mod\ p$$

$$= (g^{Sa}\ mod\ p)^{Sb}\ mod\ p$$

$= SA^{Sb}\ mod\ p$ = Bobs Informationen

Somit besitzen Alice und Bob beide am Ende $g^{ab}\ mod\ p$, ein geteiltes Geheimnis. Ist diese Übermittlung auch sicher? Übertragen werden lediglich g, p, SA und SB. Um an Sa oder Sb zu kommen muss die Gleichung $SA = g^{Sa}\ mod\ p$ umgekehrt werden, ansonsten ist es unmöglich das zweite g aus der Gleichung zu entfernen. Die Umkehrung der diskreten Exponentialfunktion, der diskrete Logarithmus ist jedoch sehr Aufwendig zu berechnen. Besonders bei guter Wahl von g und p. [2]

### 1.2.3: Die Wahl der Primzahl p

Ein weiterer Vorteil für die Wahl einer Primzahl als p ist, dass sie sich nicht durch Primfaktorzerlegung vereinfachen lässt. Allerdings ist die Primfaktorzerlegung von p-1 bereits eine Vereinfachung die z.B. mit dem Zahlkörpersieb einfach erreicht werden kann. Um dieser Bedrohung entgegenzuwirken wird für p am besten eine starke Primzahl gewählt. Da p-1 keine Primzahl sein kann (außer für p=3, was jedoch eine viel zu kleine Primzahl ist), muss p-1 mindestens durch 2 teilbar sein. Somit ist die starke Primzahl p = 2*q+1, wobei q eine weitere Primzahl ist. Somit entstehen bei der

Primfaktorzerlegung von p-1 lediglich die Faktoren 2 und q was eine minimale Vereinfachung ist. [20]

### 1.2.4: Die Wahl des Generators g

Damit möglichst viele mögliche Lösungen für den diskreten Logarithmus existieren, muss g Erzeuger der gesamten Gruppe $\mathbb{Z}_p;*$ sein, da sich alle Elemente, die nicht in der von g erzeugten Gruppe vorhanden sind, als Lösung des Diskreten Logarithmus ausschließen lassen. Die Zahl g muss demnach eine Primitivwurzel modulo p sein und zwischen 1 und p-1 liegen. Wobei man sowohl 1 als auch p-1 als exklusiv betrachten kann, da in beiden Fällen kein diskreter Logarithmus berechnet werden muss. Ein passendes g zu bestimmen ist simpel, wenn p wie oben bestimmt wurde und q bekannt ist. Nun wählt man ein g zufällig und rechnet $g^q = 1 \; mod \; p$. Sollte die Gleichung erfüllt sein ist g kein Erzeuger von $\mathbb{Z}_p;*$. Etwa die Hälfte aller Zahlen von 1 bis p sind solche Generatoren, daher lässt sich ein Generator schnell finden. [20]

## 1.3: Die Sicherheit des Diffie-Hellman Schlüsselaustausches

### 1.3.1: Man-in-the-Middle-Angriff

Der Man-in-the-Middle-Angriff basiert darauf, dass Alice' und Bobs Übertragung von einem Lauscher abgefangen werden. Dieser führt dann jeweils einen Schlüsselaustausch mit Alice' und Bob durch und steht somit zwischen ihnen. Wenn nun Alice eine Nachricht schreibt wird diese von dem Lauscher abgefangen, entschlüsselt und mit dem mit Bob geteilten Schlüssel wieder verschlüsselt. Anschließend wird die Nachricht wieder an Bob geschickt. Weder Bob noch Alice bekommen etwas davon mit, wenn dies mit jeder Nachricht geschieht. Allerdings kann dies leicht mittels Authentifizierung verhindert werden.

### 1.3.2: Logjam-Angriff

Da das Generieren von g und p zu aufwändig ist um bei jedem Schlüsselaustausch neue Zahlen zu verwenden, werden oft vorgegebene Zahlen verwendet. Dies ermöglicht jedoch, dass das Diffie-Hellman Problem durch

vorzeitige Berechnung stark vereinfacht wird. Somit kann der diskrete Logarithmus mit langwieriger Vorberechnung in wenigen Minuten gelöst werden. Mit ausreichend Ressourcen (wie sie z.B. die NSA besitzt) kann somit eine Datenbank mit den meist genutzten Wahlen für g und p angelegt werden und so fast jeder Schlüsselaustausch noch während der Übertragung geknackt werden. Somit können 512-, 768- und sogar 1024-bit Primzahlen geknackt werden. Dieses Verfahren ist, da es erst kürzlich als durchführbar bewiesen wurde (2015), im Moment die größte Bedrohung für Diffie-Hellman. [22]

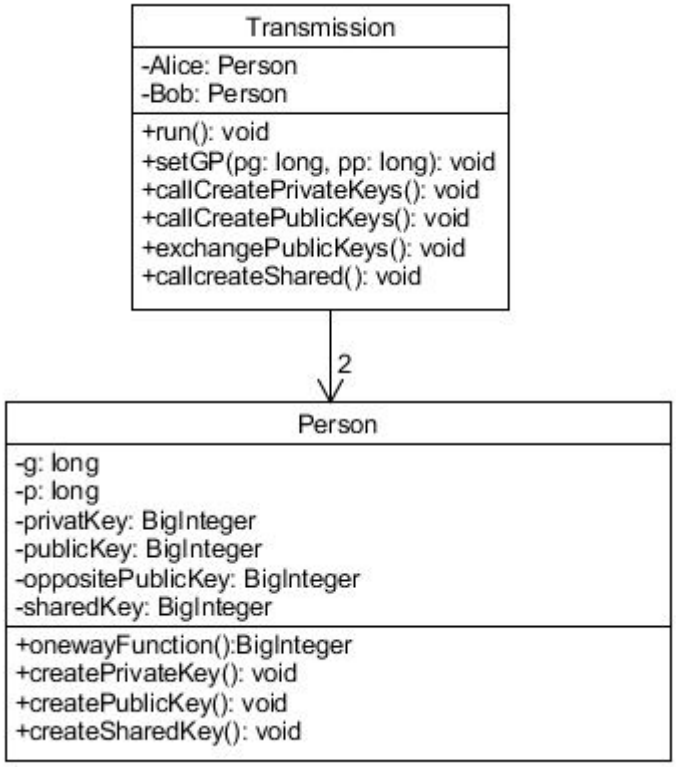
# 2: Die Implementierung in Java

## 2.1: Vormerkung

### 2.1.1: Vorüberlegungen bei der Implementierung

Es wird eine Klasse benötigt um die Personen, die einen Schlüssel austauschen wollen zu repräsentieren. Eine weitere wird als Kontrollklasse benötigt. Diese wird jedoch der Einfachheit halber in die GUI mit eingebunden.

Implementationsdiagramm:




Vorüberlegungen zur GUI: Die GUI sollte leicht zugänglich sein und zunächst nur das zeigen, was für einen Lauscher sichtbar ist, auf Knopfdruck jedoch vollständigen Zugriff gewähren.

### 2.1.2: Schwierigkeiten bei der Implementierung:

Die Implementierung von Diffie-Hellman ist größtenteils simpel, jedoch trat bereits sehr früh ein großes Problem auf. Das Erstellen von p. Die Klasse BigInteger kann zwar Primzahlen erzeugen, starke Primzahlen jedoch nicht. Aus

diesem Problem entstand auch das Problem des Generierens von g. Wenn p keine starke Primzahl ist es nicht möglich, schnell zu überprüfen ob g ein Generator von ganz $\mathbb{Z}_p;*$ ist. Bis zuletzt war es mir nicht möglich, dieses Problem zu lösen. Ein weiteres Problem ist es, g und die privaten Schlüssel so zu erzeugen, dass es Zahlen zwischen 1 und p-1 sind, da das Erzeugen von zufälligen BigIntegern keine Obergrenze zulässt, lediglich die Länge der Zahl in Bits. Um dieses Problem zu umgehen sind die betroffenen Zahlen ein Bit kürzer als p.

## 2.2: Klassen

### 2.2.1: Importierte Klassen

Importiert wurden lediglich die Klassen BigInteger und Random. Die Klasse BigInteger wird verwendet, da mit extrem großen Zahlen gerechnet wird und außerdem enthält die Klasse die Methode modPow. Diese löst die diskrete Exponentialfunktion weit aus effizienter als erst zu potenzieren und dann das Modul zu bilden. Die Klasse Random wird lediglich importiert, da sie bei zufälliger Erzeugung von g und p mittels der Klasse BigInteger benötigt wird.

### 2.2.2: Die selbstgeschriebene Klasse Person

Klassenattribute sind g, p, der eigene private Schlüssel, die öffentlichen Schlüssel und der geheime, geteilte Schlüssel, welcher das Ziel des Schlüsselaustausches ist sowie ein Integer um die in der GUI angegebene Bitlänge zu speichern. Dies wird beim Erstellen der Personen getan. Die Methode "createPrivateKey" erzeugt eine Zufallszahl, welche als privater Schlüssel verwendet wird (hierbei ist die Bitlänge um eins kleiner als die von p, damit der Schlüssel kleiner als p ist. "createPublicKey" und "createSharedKey" wenden beide jeweils nur die diskrete Exponentialfunktion mittels der von der Klasse BigInteger bereitgestellten Methode modPow und den entsprechenden Variablen an. Alle weiteren Methoden sind lediglich getter und setter Methoden.

### 2.2.3: Die selbstgeschriebene Klasse Transmission/GUI

Die Klasse, welche zunächst Transmission hieß, wurde bei dem Übertragen von BlueJ in Netbeans in die Klasse GUI mit eingebunden. Die Klasse besitzt zwei Objekte der Klasse Person, welche die Personen darstellen, zwischen denen ein Schlüsselaustausch stattfindet. Außerdem gibt es zwei Strings um die Ausgabe für Alice' und Bobs Textfeld zu speichern, sodass sie auf Knopfdruck angezeigt werden kann. Dies wird in den Methoden "jButtonAActionPerformed" und "jButtonBActionPerformed" getan. Sollte der Knopf jButtonT gedrückt werden, so werden zunächst alle Textfelder und Buttons zurückgesetzt. Dann wird, falls der Inhalt des Textfeldes "jTextField1" nicht zu klein oder kein Integer ist eine neue Übertragung mit der im "jTextField1" angegebenen Bitlänge gestartet, ansonsten wird für die Bitlänge 64 verwendet. Die Methode welche durch den Button "jButtonR" aufgerufen wird zeigt lediglich den geheimen Schlüssel welcher nach dem Austausch in Besitz von Alice bzw. Bob ist in den Textpanelen "jTextPaneAK" und "jTextPaneBK" an. Die Methode "setGP" überträgt g und p an die beiden Personen und gibt die Zahlen im mittleren Textpanel aus. Die Methoden "callCreatePrivateKeys", "callCreatePublicKeys" und callCreateSharedKey rufen lediglich die entsprechenden Methoden in der Klasse Person auf und geben dies aus. Die Methode "exchangePublicKeys" setzt die Attribute "oppositePublicKey" der beiden Instanzen der Klasse Person auf den publicKey der jeweilig anderen Instanz und gibt die publicKeys aus. Die Methode "run" started den Schlüsselaustausch und setzt gleichzeitig die Personen durch einfaches Überschreiben zurück. Nach dem Zurücksetzten werden neue Zahlen für p und g gewählt und die Methoden callCreatePrivateKeys, callCreatePublicKeys, exchangePublicKeys und callCreateSharedKey aufgerufen, sowie die Strings a und b erweitert.

# Fazit

Zusammenfassend lässt sich sagen, dass der Diffie-Hellman Algorithmus zum Schlüsselaustausch mit gegebenem g und p sehr einfach durchzuführen ist, jedoch mittels LogJam von z.B. der NSA einfach geknackt werden kann und somit nur Schutz vor kleineren Gruppen von Angreifern bietet. Diffie-Hellman mit eigenen Werten für g und p durchzuführen ist zwar weitaus sicherer aber auch um ein vielfaches zeitaufwändiger und für den praktischen Gebrauch nicht geeignet. Daher ist der Diffie-Hellman Schlüsselaustausch (sofern man nicht auf 2048-bit oder größere Primzahlen umsteigt) nicht vor größeren Bedrohungen sicher. Ein besserer Algorithmus ist jedoch nicht vorhanden, da auch RSA Schwächen hat. Somit ist der Diffie-Hellman Schlüsselaustausch unvermeidbar, das Einzige was zur Stärkung der Sicherheit getan werden kann, ist die Bitlänge zu erhöhen.

# Quellen:


| NR | Quelle | Datum |
|---|---|---|
| 1 | "The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography" von Simon Singh | |
| 2 | "Einführung in die Kryptologie" von Juraj Hromkovic, Karin Freiermuth, Lucia Keller und Björn Steffen | |
| 3 | https://de.wikipedia.org/wiki/Diffie-Hellman-Schl%C3%BCsselaustausch | 23.4.17 |
| 4 | https://de.wikipedia.org/wiki/Primitivwurzel | 23.4.17 |
| 5 | https://de.wikipedia.org/wiki/Satz_von_Euler | 23.4.17 |
| 6 | https://www.youtube.com/watch?v=SaDpNBZQSj0 | 23.4.17 |
| 7 | https://www.youtube.com/watch?v=WJBeSB41Rp4 | 23.4.17 |
| 8 | https://www.youtube.com/watch?v=XV-jT1VPmEc | 23.4.17 |
| 9 | https://www.youtube.com/watch?v=d2PU_vw7Eug | 23.4.17 |
| 10 | https://www.youtube.com/watch?v=Mzp0BoDSumo | 23.4.17 |
| 11 | https://www.youtube.com/watch?v=-ckiq1M-RYA | 23.4.17 |
| 12 | https://www.youtube.com/watch?v=91bTabKYDrE | 23.4.17 |
| 13 | https://www.youtube.com/watch?v=BqoO5HDdGAY | 23.4.17 |
| 14 | https://www.youtube.com/watch?v=WHyQsVqG7iQ | 23.4.17 |
| 15 | https://www.youtube.com/watch?v=LnzKQvEt1zs | 23.4.17 |
| 16 | https://www.youtube.com/watch?v=LnzKQvEt1zs | 23.4.17 |
| 17 | https://www.youtube.com/watch?v=2uYufusFEDQ | 23.4.17 |
| 18 | https://www.youtube.com/watch?v=I74D-g1ZbDQ | 23.4.17 |
| 19 | https://www.youtube.com/watch?v=DU082wcr40A | 23.4.17 |
| 20 | http://www.inf.fh-flensburg.de/lang/krypto/protokolle/diffie.htm | 23.4.17 |
| 21 | http://www.inf.fh-flensburg.de/lang/krypto/grund/gruppezn.htm#section4 | 23.4.17 |
| 22 | https://weakdh.org/ | 25.4.17 |

# Versicherung der selbständigen Erarbeitung

Ich versichere, dass ich die vorliegende Arbeit einschließlich evtl. beigefügter Zeichnungen, Kartenskizzen, Darstellungen u. ä. m. selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter genauer Angabe der Quelle deutlich als Entlehnung kenntlich gemacht.

_____, den _____

(Ort) (Datum)

_____

(Unterschrift)