Datensicherheit – Zusammenfassung

Malte L. Jakob

17. Juli 2021

Inhaltsverzeichnis

1 Einführung

1.1 Was ist Datensicherheit?

Es gibt mehrere Begriffe, die alle gleich klingen und sich allesamt um verwandte Thematiken drehen, dennoch bedeuten sie unterschiedliche Dinge. Eine kleine Übersicht, was für Verwechslungen entstehen können, und worin sich die Maßnahmen zum Erreichen der Ziele unterscheiden, ist in Tabelle 1.1 aufgeführt.

Nachfolgend wird etwas genauer auf die unterschiedlichen Ziele und Maßnahmen eingegangen.

1.1.1 Datensicherung

Die Datensicherung dient dem Schutz vor dem Verlust der Daten – sei es durch einen Unfall oder durch böswillige Handlungen. Unternehmen, die keinen Zugriff mehr auf ihre Daten haben, stehen in der Regel nach drei Tagen vor dem Ruin.

Um das zu verhindern ist Datensicherheit, wie z.B. ein Backup an einem andern Ort, sehr wichtig. Beim Backup gibt es zwei Faktoren, die wichtig sind:

1. Wie viel Zeit ist seit dem letzten Backup vergangen?
2. Wie viel Zeit wird benötigt, um das Backup wieder einzuspielen?

Zudem gibt es unterschiedliche Backup-Arten, die jeweils ihre eigenen Vor- und Nachteile haben:

Full-Disk Backups speichern bei jedem Backup alle Daten.

Inkrementelle Backups speichern nur beim ersten Mal alle Daten, danach nur noch die, die sich seit dem letzten Backup verändert haben.

Journaling und Atomare Transaktionen spielen bei Datenbanken eine wichtige Rolle. Hier wird aufgeschrieben, was die Datenbank vorhatte, sollte sie mal abstürzen. Was auch immer sie aufschreibt muss atomar sein, und wird somit entweder ganz oder gar nicht ausgeführt.

1.1.2 Datenschutz

Datenschutz beschreibt das Recht auf informationelle Selbstbestimmung.

Informationelle Selbstbestimmung heißt wiederum, dass man weiß, welche personenbezogenen Daten von wem und zu welchem Zweck verarbeitet werden, sowie die Möglichkeit, diese Verarbeitung zu unterbinden.

Personenbezogene Daten sind Daten, die es ermöglichen eine Person innerhalb eines gewissen Personenkreises eindeutig zu identifizieren.

Das erste Gerichtsurteil, das einem Bürger dieses Recht zuspricht ist das Volkszählungsurteil von 1938. Durch diese Informationelle Selbstbestimmung soll vermieden werden, dass jemand zum „gläsernen Bürger“ wird, ohne dass er/sie das möchte.

Deutscher Begriff	Englischer Begriff	Maßnahmen
Datensicherheit	Data Security	Mathematisch-logische Maßnahmen
Datensicherung	Data Safety	Physikalische Maßnahmen
Datenschutz	Data Privacy	Juristische Maßnahmen

Tabelle 1.1: Verwandte, aber verschiedene Ziele und deren Maßnahmen

Ziel	Maßnahmen
Vertraulichkeit	Verschlüsselung, Steganographie
Integrität	Digitale Signatur, Message Authentication Code (MAC)
Authentizität	Digitale Signatur, Message Authentication Code (MAC)
Verbindlichkeit	Digitale Signatur (*Nicht* MAC)

Tabelle 1.2: Ziele der Datensicherheit und Maßnahmen, um diese zu erreichen

1.1.3 Datensicherheit

Datensicherheit hat vier grundlegende Zielsetzungen:

Vertraulichkeit: Nur die Menschen haben Zugriff, die auch Zugriff haben sollten

Integrität: Die Daten sind vor versehentlichen und absichtlichen Veränderungen geschützt, bzw. es ist erkennbar, wenn dies passiert.

Authentizität: Man kann sich darauf verlassen, dass Gesprächspartner auch diejenigen sind, die sie vorgeben zu sein.

Verbindlichkeit: *Auch non-repudiation (zu dt. Nicht-Abstreitbarkeit) genannt.* Es kann nicht abgestritten werden, dass man für gewisse Dinge verantwortlich ist bzw. eingewilligt hat.

Eine Übersicht entsprechender Maßnahmen, um diese zu erreichen ist in Tabelle 1.2 dargestellt

1.1.3.1 Verschlüsselung und Codierung

Oft werden die Worte „Verschlüsselt“ und „Codiert“ synonym verwendet, doch in Wahrheit gibt es einen entscheidenden Unterschied.

Zwar sind beides Abbildungen aus einem Definitionsbereich $\mathbb{D}$ in einen Bildbereich $\mathbb{B}$, deren Abbildung mithilfe einer Abbildungsfunktion $f(x)$ durchgeführt wird.

Bei einer *Codierung* ist diese Abbildungsfunktion immer gleich und nur von dem zu codierenden Element x aus der Definitionsmenge $\mathbb{D}$ abhängig. Bei einer Abbildungsfunktion $f(x) = 3x$ ergibt das Codieren des Elements $x = 3$ immer $f(3) = 3 \cdot 3 = 9$. Dementsprechend ist auch das zurückrechnen vom Bildbereich $\mathbb{B}$ in den Definitionsbereich $\mathbb{D}$ durch die Funktion $f^{-1}(x) = \frac{x}{3}$ immer einfach durchzuführen.

Ist der Definitionsbereich sehr klein, kann auch einfach in einer Tabelle darstellen, welcher Wert aus $\mathbb{D}$ zu welchem Wert aus $\mathbb{B}$ gehört. So geschieht es zum Beispiel bei den gängigen Codierungen ASCII oder Unicode.

Bei einer *Verschlüsselung* hingegen, hängt die Abbildungsfunktion nicht nur von x, sondern auch noch von mindestens einem weiteren Parameter – zum Beispiel s – ab. Somit entstehen Funktionen wie $f_s(x) = 3x - s$. Nun muss man zur eindeutigen Umkehrung dieser Abbildungsfunktion bereits das geheime s kennen, oder sehr viel raten, um eine sinnvolle Lösung zu erhalten.

2 Grundlagen

2.1 Terminologie

Nachfolgend werden einige grundlegende Begriffe geklärt.

Kryptographie ist die Lehre des Absicherns von Nachrichten durch Verschlüsselung

Kryptoanalyse ist das Reproduzieren des Klartextes aus dem Chiffretext ohne die Kenntnis über den Schlüssel

Kryptologie ist die Verbindung aus Kryptographie und Kryptoanalyse

Alphabet: Ein Alphabet A ist eine Endliche Menge von Zeichen $n = |A|$ ist die Mächtigkeit des Alphabets.

Klartext ist der lesbare Text einer Nachricht M und wird als Zeichenkette über das Alphabet A abgebildet. Auch Schlüssel sind Zeichenketten.

Chiffretext ist der verschlüsselte Text über dem gleichen, oder einem anderen Alphabet.

Verschlüsselung bezeichnet den Vorgang, die Nachricht unverständlich zu machen

Chiffre E ist eine invertierbare (=umkehrbare) Abbildung, welche aus dem Klartext M und dem Schlüssel S den Chiffretext C erzeugt.

Entschlüsselung D ist die Umkehrung der Chiffre E. Es gilt

$$E(M) = C \wedge D(C) = M \wedge D(E(M)) = M$$

2.2 Kryptographische Algorithmen

Kryptographische Algorithmen sind mathematische Funktionen zur Ver- und Entschlüsselung

Symmetrische Algorithmen verwenden denselben Schlüssel K zur Ver- und Entschlüsselung:

$$E_K(M) = C \wedge D_K(C) = M \wedge D_K(E_K(M)) = M$$

Asymmetrische Algorithmen verwenden für das Verschlüsseln einen Schlüssel K_1 und zum Entschlüsseln einen anderen Schlüssel K_2

$$E_{K_1}(M) = C \wedge D_{K_2}(C) = M \wedge D_{K_2}(E_{K_1}(M)) = M$$

Stromchiffren verschlüsseln ein Zeichen nach dem Anderen

Blockhiffren verschlüsseln ganze Blöcke (z.B. 64 Bit) auf einmal, und dann Block für Block

Kryptosysteme sind die Kombination aus Algorithmus, Schlüssel und verschlüsselten Nachrichten

Manche Organisationen setzen beim Verschlüsseln auch auf die Geheimhaltung der Verschlüsselungsalgorithmen (sogenannten eingeschränkte Algorithmen), um noch mehr Sicherheit zu gewährleisten. Diese Methode ist jedoch nicht sehr sinnvoll, da

- jemand, der die Organisation verlässt, den Algorithmus trotzdem kennt,
- der geheime Algorithmus nicht zur Verschlüsselung der Kommunikation mit Dritten verwendet werden kann, und

- die Qualitätskontrolle durch den kleinen Personenkreis sehr erschwert wird.

Die Sicherheit eines Verschlüsselungsverfahrens sollte daher nicht von der Einschränkung eines Algorithmus abhängen, sonder nur von der des Schlüssels. Wenn ein Algorithmus öffentlich ist und dennoch nicht geknackt werden kann, wird dies *Starke Kryptographie* genannt.

2.3 Kryptographische Protokolle

Kryptographische Protokolle sind ebenso wichtig, wie die Verschlüsselung selbst. Sie sind Verfahren zur Steuerung des Ablaufs von Transaktionen für bestimmte Anwendungen. Ohne ein geregeltes Verfahren kann es zu erheblichem Mehraufwand kommen.

Nehmen wir beispielsweise das Problem der *Key Distribution*. Zwischen beliebig vielen Mitarbeitern einer Organisation soll die Kommunikation verschlüsselt werden. Nutzt man hier, ohne besonderes Verfahren, eine symmetrische Verschlüsselung, muss für jede einzelne Beziehung zwischen m Mitarbeitern ein eigener Schlüssel erstellt werden. Diese Schlüsselanzahl, die mit der Formel $\sum_{i=1}^{m}(i-1)$ berechnet werden kann, nimmt schnell überhand.

Stattdessen können Public-Key-Algorithmen (siehe Punkt 2.4) verwendet werden, um für jeden Mitarbeiter einen privaten und einen öffentlichen Schlüssel zu generieren. Die öffentlichen Schlüssel werden auf einen Keyserver geladen, auf den jeder Zugriff hat. Möchte nun jemand kommunizieren, so muss nur der Schlüssel auf dem Keyserver kopiert und die Nachricht damit verschlüsselt werden. Hier beträgt die Anzahl an Schlüsseln bei m Mitarbeitern also nur $2m$ – das Verfahren ist wichtig!

2.4 Public-Key-Algorithmen

Public-Key-Algorithmen sind asymmetrische Algorithmen. Es gibt zwei Schlüssel, die mathematisch voneinander abhängig sind; Ein Schlüssel ist privat (secret key S), der andere ist öffentlich (public key P). Es ist mathematisch nicht möglich vom public key auf den private key zurückzuschließen. Möchte eine Person A eine Nachricht an Person B verschlüsseln, so nutzt sie zum Verschlüsseln den öffentlichen Schlüssel von P_B. Die Verschlüsselte Nachricht wird an B übermittelt und diese kann sie mittels des privaten Schlüssels S_B wieder entschlüsseln. Es gilt also:

$$E_{P_B}(M) = C \wedge D_{S_B}(C) = M \wedge D_{S_B}(E_{P_B}(M)) = M$$

Gleichzeitig gilt es aber auch anders herum: Was mit dem privaten Schlüssel verschlüsselt wurde, kann mit dem öffentlichen Schlüssel wieder entschlüsselt werden:

$$E_{S_B}(M) = C \wedge D_{P_B}(C) = M \wedge D_{P_B}(E_{S_B}(M)) = M$$

Das kann zum *Signieren* von Nachrichten verwendet werden. Der Nachrichteninhalt M wird in eine Hashfunktion H gegeben und man erhält den Hash $H(M)$. Dieser Hash wird nun mit dem privaten Schlüssel des Absenders (hier B) verschlüsselt: $E_{S_B}(H(M))$ und an die Originalnachricht angehängt. Der Empfänger der Nachricht kann mit derselben Hashfunktion den Inhalt der Nachricht nachrechnen. Nun entschlüsselt er die mitgesendete Signatur: $D_{P_B}(E_{S_B}(H(M))) = H$; Die beiden Hashs sollten gleich sein. Ist dies nicht der Fall, so wurde die Nachricht entweder modifiziert oder nicht von der Person erstellt, die sie vorgibt zu sein. So sind zugleich Integrität, Verbindlichkeit und Authentifikation einer Nachricht gesichert.

2.5 Kryptoanalyse

Die Kryptoanalyse, auch Kryptanalyse genannt, versucht die Verschlüsselung zu durchbrechen. Hierbei gibt es verschiedene Arten von Angriffen:

Cyphertext-Only-Angriff: Der Kryptoanalytiker kennt nur den Chiffretext

Known-Plaintext-Angriff Der Kryptoanalytiker kennt Chiffre- und Klartext

Chosen-Plaintext-Angriff Der Kryptoanalytiker kann sich den Klartext auswählen, zu dem er den Chiffretext erhält (z.B. Möglich bei PIN-Eingaben von Bankautomaten)

Chosen Ciphertext-Angriff Der Angreifer sucht sich einen verschlüsselten Text aus, bei dem er die Möglichkeit hat an den Klartext heranzukommen

Angriff mit Gewalt: Der Kryptoanalytiker bedroht oder Foltert eine Person mit Zugang zum Schlüssel

Angriff mit gekauftem Schlüssel der Schlüssel wird mittels Bestechung „gekauft“

Die effektivsten Methoden, um starke Kryptographie zu überwinden sind diejenigen, die auf den Menschen abziehen, wie z.B. Social Engineering.

Angriffe, bei denen einfach alle möglichen Schlüssel ausprobiert werden, werden *Brute-Force-Angriffe* genannt.

Definition 2.1: Sicherheit eines Algorithmus

Ein Algorithmus gilt dann als sicher, wenn

- der zum Aufbrechen nötige Geldaufwand den Wert der verschlüsselten Daten übersteigt, oder
- die zum Knacken erforderliche Zeit größer ist als die Zeit, die die Daten geheim bleiben müssen
- das mit einem bestimmten Schlüssel chiffrierte Datenvolumen kleiner ist als die zum Knacken erforderliche Datenmenge (= Es gibt nicht genug „Testmaterial“, um auf den richtigen Schlüssel zu kommen)

Ein Algorithmus ist uneingeschränkt sicher, wenn der Klartext auch dann nicht ermittelt werden kann, wenn Chiffretext in beliebigem Umfang vorhanden ist.

Weitere wichtige Größen sind die Berechnungskomplexität (also die Rechenzeit in Abhängigkeit von der Schlüsselgröße), der Speicherplatzbedarf und die Datenkomplexität (vgl. „Testmaterial“)

2.6 Sicherheit von Schlüsseln

Schlüssel können unterschiedlich sicher sein, je nachdem wie groß der *Schlüsselraum*, also die Anzahl aller möglichen Schlüssel ist. Nimmt man z.B. ein herkömmliches Fahrrad-Zahlenschloss, mit vier Ziffern von 0 bis 9, so ist der Schlüsselraum $10^4 = 10.000$ – für einen Computer nicht viel.

So hat der 56-Bit Schlüssel des Data-Enncryption-Standard (DES) einen Schlüsselraum von $2^{56} \approx 7 \cdot 10^{16}$; Die beste Hardwareimplementierung (eine Platine, die genau so gelötet ist, dass sie nur das berechnet), kann ca. $3.9 \cdot 10^5$ Schlüssel pro Sekunde berechnen. So benötigt die Maschine im Schnitt nur $3.9 \cdot 10^5$ Sekunden (ca. 4,5 Tage), um den richtigen Schlüssel zu berechnen; Somit ist dieser Schlüssel unsicher.

Betrachtet man hingegen einen 100-Bit Schlüssel, so steigt die Rechenzeit auf eine Zeit an, die die bisherige Existenzdauer dieses Universums übersteigt.

3 Klassische Chiffren

Definition 3.1: Klassische Chiffren

Bei einer *Dispositionschiffre* wird der Geheimtext durch eine Permutation der Klartextzeichen erzeugt. Bei einer *Substitutionschiffre* wird jeder Zeichen des Klartextes durch ein anderes ersetzt, die Position bleibt jedoch gleich. Eine Substitutionschiffre heißt *monoalphabetisch*, wenn jedes Klartextzeichen (bei selbem Schlüssel) immer auf das gleiche Geheimtextzeichen abgebildet wird (z.B. beim Ceasarcipher wird beim Schlüssel 2 jedes e immer zu eine g); Sie heißt *polyalphabetisch*, wenn sie nicht monoalphabetisch ist (und ein e bei selbem Schlüssel nicht immer auf ein g abgebildet wird).

Ältere und bekannte Chiffren, ersetzen nach bestimmten Regeln alle Buchstaben durch andere desselben Alphabets. Über unserem natürlichen Alphabet von a bis z gibt es $26! \approx 4{\cdot}10^{26}$ Möglichkeiten, die Buchstaben einander zuzuordnen. Bei Binärdaten dagegen gibt es nur zwei: Die Identität (jede 1 bleibt eine 1 und jede 0 bleibt eine 0), und die Inverse (Die Werte werden jeweils umgekehrt).

Allerdings stellt sich bei jedem endlichen Alphabet das Problem, dass man beim Rechnen mit den Buchstaben, sei es durch Addition oder Multiplikation, irgendwann ans Ende des Alphabetes kommt und das Ergebnis außerhalb liegt. Um dieses Problem zu beheben, wird der Modulo verwendet, um die Ergebnisse innerhalb des existierenden Bereiches zu beschränken.

3.1 Monoalphabetische Chiffren

3.1.1 Verschiebechiffren

Definition 3.2: Verschiebechiffren

Bei einer Verschiebechiffre wird jedes Klartextzeichen z durch ein um k Zeichen im Alphabet verschobenes Zeichen ersetzt. Bei einem Alphabet mit n Zeichen seien die Zeichen durchnummeriert von 0 bis $n-1$. Dann gilt für die Verschiebechiffre:

$$z \mapsto (z + k) \mod n$$

Eine der bekanntesten Verschiebechiffren ist der Ceasar-Cipher, der genau so funktioniert, wie in obiger Definition beschrieben. Allerdings ist diese Verschlüsselung heutzutage sehr einfach zu knacken.

Da es insgesamt nur 25 Möglichkeiten gibt, die Buchstaben zu verschieben, dauert ein Brute-Force-Angriff nicht lange.

Zum anderen, da Zeichen für Zeichen verschlüsselt und ersetzt wird, bleibt die *Buchstabenhäufigkeit* bestehen. In der deutschen Sprache ist der häufigste Buchstabe das e mit 17,40%. Hat man nun einen so verschlüsselten Text, kann man der Hypothese folgen, dass der häufigste Buchstabe das e ist. Ist der häufigste Geheimtextbuchstabe z.B. das g, so können wir probieren, ob der Schlüssel 2 ist (g (6) - e (4) = 2). Somit ist diese Chiffre selbst bei einer Ciphertext-Only-Attacke mit minimaler Datenkomplexität einfach umkehrbar.

3.1.2 Multiplikative Chiffren

Definition 3.3: Multiplikative Chiffren

Bei einer multiplikativen Chiffre über dem Alphabet A wird jedes Klartextzeichen z mit einer Zahl $t \in 0, \ldots, n-1$ multipliziert. t und $n = |A|$ müssen teilerfremd sein, und der größte gemeinsame Teiler somit 1. Die Chiffriervorschrift lautet:

$$z \mapsto (z \cdot t) \mod n$$

Wegen $(z \cdot t) \mod n = (z \mod n) \cdot (t \mod n) \mod n$ müssen, wie bei der Verschiebechiffre auch, für t nur Werte von 0 bis $n-1$ betrachtet werden, da größere Werte durch die Modulo-Funktion nur wieder auf bereits vorhandene, kleinere Werte abgebildet werden.

Der Schlüsselraum für unser natürliches Alphabet, also Zahlen, die mit 26 teilerfremd sind, sind folgende zwölf Zahlen: $1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25$

3.1.3 Tauschchiffren (Affine Chiffren)

Definition 3.4: Tauschchiffre

Sei $ggT(t, n) = 1$. Dann wird jede Chiffre E mit $z \mapsto (zt + k) \mod n$ affine Chiffre oder Tauschchiffre genannt.

Die Umkehrfunktion der Tauschchiffre ist dementsprechend $f^{-1}(z') = \frac{z'-k}{t}$. Die Schwierigkeit hierbei ist, das multiplikative Inverse von t herauszufinden, am besten funktioniert dies mit dem erweiterten Euklidischen Algorithmus (siehe 10.4). Ohne diesen müssen viele verschiedene Zahlen getestet werde, bis das Inverse gefunden wurde.

Da hier multiplikative und Verschiebechiffren kombiniert werden, so werden auch deren Schlüsselräume verknüpft. Über unser gebräuchliches Alphabet $A = \{a, \ldots, z\}$ gibt es $12 \cdot 26 = 312$ Möglichkeiten, die verschiedenen Schlüssel zu kombinieren.

3.1.4 Kryptanalyse monoalphabetiscer Chiffren

Bei jeder monoalphabetischen Chiffre ist ihre schwäche dennoch, dass die Buchstaben- und Buchstabenpaarhäufigkeiten auch verschlüsselt gleich bleiben. So können durch Häufigkeitsanalysen gewisse Buchstaben identifiziert werden, sodass ein Lückentext entsteht, in den man durch einiges manuelles Rumprobieren die fehlenden Buchstaben einsetzen kann, bis man den kompletten Text erhält.

Diese statistische Abhängigkeit zwischen Chiffretext und Klartext wird auch *information leak* genannt, die das Knacken der Verschlüsselung bei ausreichend Wissen zur Sprache des Klartextes stark vereinfachen.

3.2 Polyalphabetische Chiffren

Um die Korrelation monoalphabetischer Chiffren zu umgehen, wurden polyalphabetische Chiffren entwickelt, bei denen ein Klartextbuchstabe nicht immer auf denselben Chiffrebuchstaben abgebildet wurde.

Hierbei gibt es. z.B. *homophone Chiffren*, bei denen jedes Zeichen im Chiffretext in etwa gleich oft vorkommt, was das Knacken durch statistische Analysen erheblich erschwert. Dennoch gibt es gewisse Buchstabenpaare, die häufiger vorkommen als andere; Um dieses Muster zu analysieren, sind aber weitaus mehr Daten benötigt als bei einer monoalphabetischen Ciffre.

3.2.1 Vigenère-Chiffre

Ein Beispiel einer polyalphabetischen Chiffre wurde im 16. Jahrhundert von Blaise de Vigenère entwickelt. Diese Chiffre ist eine simple Verschiebechiffre; doch anstatt jeden Buchstaben immer um dieselbe Anzahl an Stellen zu verschieben, gibt es nun eine *Liste von Verschiebe-Werten*, die auch entsprechend als Wort dargestellt werden kann (z.B. passwort). Nun wird der erste Buchstabe des Klartextes um den Wert des

ersten Passwort-Buchstabens verschoben, der zweite Buchstabe des Klartextes um den Wert des zweiten Passwort-Buchstabens usw. Ist man am Ende des Passwortes angelangt, so beginnt man wieder an dessen Anfang.

Somit lässt sich für die Werte
z =Nummer des zu codierenden Zeichens im Alphabet,
i =Position von z im Klartext,
n =die Länge des Alphabets,
K =das Schlüsselwort,
k =die Länge des Schlüsselwortes, und
K_x =das Zeichen in K an der Stelle x
folgende Abbildung festlegen:

$$E(z,i) = (z + K_{i \mod k}) \mod n = V_{z,K_{i \mod k}}$$

$V_{x,y}$ ist hierbei der Wert einer Matrix V, die abbildet, auf welches Geheimtextzeichen ($V_{x,y}$) ein Klartextzeichen x unter Addition des Zeichens y abgebildet wird.

Bei ständiger Wiederholung des Passwortes kommt es jedoch vor, dass ein Klartextzeichen dennoch mit demselben Passwort-Zeichen verschlüsselt wird und die Chiffrezeichen somit identisch sind.

Zu wissen, wie lang das Passwort ist kann das Dechiffrieren des Textes wieder vereinfachen. Somit können nämlich alle Buchstaben, die mit dem ersten Buchstaben des Passwortes verschlüsselt wurden, in einen Topf geworfen werden, alle Buchstaben, die mit dem zweiten Buchstaben des Passwortes verschlüsselt wurden, kommen ebenfalls in einen Topf usw.. Für eine Passwortklänge k gibt es also k Töpfe mit je $\frac{m}{k}$ Buchstaben (m = Länge des Chiffretextes).

Innerhalb dieser Töpfe gilt wieder die jeweilige Buchstabenhäufigkeit der Sprache, da hier alle Buchstaben um denselben Wert verschoben wurden. So kann also für jeden Topf (also für jeden Buchstaben des Passworts) der häufigste Buchstabe ermittelt werden und somit der Abstand zum häufigsten Buchstaben der Sprache berechnet werden. Topf für Topf bekommt man also die Werte des Passwortes. Je nach Länge des Textes m kann es natürlich zu einigen Ausreißern kommen, bei denen die Buchstabenhäufigkeiten durch zu kleine Probengröße nicht den normalen Häufigkeiten der Sprache passt.

3.2.1.1 Kasiski-Test

Um die Länge k des Passwortes zu ermitteln gibt es verschiedene Ansätze; Einer davon ist der Kasiski-Test. Dieser sucht nach Wiederholungen von Buchstabensequenzen, die mindestens drei Zeichen lang sind, und misst deren Abstände voneinander (jeweils die Abstände der ersten Buchstaben).

Diese Abstände werden nun in ihre Primfaktoren zerlegt; Die Faktoren, die bei allen Abständen am häufigsten vorkommen, sind möglicherweise das k. Hier muss nun mittels der oben genannten Topf-Methode ausprobiert werden, ob die Schlüsselwortlänge einen halbwegs sinnigen Klartext ergibt.

3.2.1.2 Friedmann-Test

Der Friedmann-Test liefert einen Näherungswert an k, indem er eine Art gewichteten Durchschnitt berechnet. Dieser basiert auf dem sogenannten *Koinzidenzindex*, der besagt, wie groß die Wahrscheinlichkeit ist, dass zwei zufällig ausgewählte Buchstaben eines Textes gleich sind.

Für einen Durchschnitt muss natürlich erst einmal eine Gesamtanzahl an Paaren berechnet werden. Mit der Ausnahme von einer Textlänge $m = 1$ entstehen für jeden hinzukommenden Buchstaben m neue Paare; Also für 2 Buchstaben entsteht 1 Paar, beim nächsten Buchstaben entstehen 2 zusätzliche Paare, beim darauffolgenden (vierten) Buchstaben 3 Paare usw. Dies entspricht also der Summenformel $\sum_{i=1}^{m} i$. Da wir allerdings noch berücksichtigen müssen,dass bei nur einem Buchstaben kein Paar entsteht, ergibt sich die Summenformel

$$\text{Anzahl aller Paare eines Textes } = \frac{m \cdot (m-1)}{2}$$

Die Anzahl jedes Buchstaben m_i eines Textes (z.B. m_1 = Anzahl von a, m_2 = Anzahl von b, $\ldots$, m_{26} = Anzahl von z) kann natürlich aufsummiert werden und ergibt die gesamte Anzahl an Buchstaben eines Textes: $\sum_{i=1}^{26} m_i = m$

Sind die Buchstabenhäufigkeiten abgezählt, so lässt sich natürlich auch die Anzahl aller Paare gleicher Buchstaben berechnen:

$$\text{Anzahl der Paare eines Buchstabens } m_i \text{ im Text } = \frac{m_i \cdot (m_i - 1)}{2}$$

Die Anzahl aller Paare gleicher Buchstaben ist entsprechend:

$$\text{Anzahl aller Paare gleicher Buchstaben eines Textes } = \sum_{i=1}^{26} \frac{m_i \cdot (m_i - 1)}{2}$$

Nun lässt sich der Koinzidenzindex I berechnen, der wie oben beschrieben angibt, wie groß die Wahrscheinlichkeit ist, aus allen Buchstabenpaaren eines auszuwählen, dessen Buchstaben gleich sind. Diese Berechnung basiert auf der statistischen Formel $\frac{\text{Anzahl gleiche Paare}}{\text{Anzahl alle Paare}}$:

$$I = \frac{\sum_{i=1}^{26} \frac{m_i \cdot (m_i - 1)}{2}}{\frac{m \cdot (m-1)}{2}} = \sum_{i=1}^{26} \frac{m_i \cdot (m_i - 1)}{2} \cdot \frac{2}{m \cdot (m-1)} = \frac{\sum_{i=1}^{26} m_i \cdot (m_i - 1)}{m \cdot (m-1)}$$

Für lange deutsche Texte wurde dieser Wert bestimmt als $I_d := 0,0762$.

Bei zufällig generierten Texten, in dem alle Buchstaben entsprechend gleich oft vorliegen, ergibt sich logischerweise ein Koinzidenzindex $I_r := \frac{1}{26} \approx 0,0385$.

Nun wird der Anfangs erwähnte gewichtete Durchschnitt berechnet: Dieser Durchschnitt ergibt sich aus der Anzahl der gleichen Paare mit der Sprachinzidenz und der Anzahl der gleichen Paare mit der Zufallsinzidenz.

Die Anzahl aller Buchstabenpaare innerhalb eines Topfes wird gleich berechnet, wie die Anzahl aller Paare des Textes, nur muss hier statt m, $\frac{m}{k}$, verwendet werden, da es ja nur so viele Buchstaben pro Topf gibt. Es gilt also

$$\text{Anzahl aller Paaare innerhalb eines Topfes} = \frac{\frac{m}{k} \cdot \left(\frac{m}{k} - 1\right)}{2}$$

Möchte man die Anzahl der Paare von allen k Töpfen wissen, muss man das obige Ergebnis entsprechend mit k multiplizieren:

$$\begin{aligned}
\text{Anzahl aller Paaare innerhalb aller Töpfe} &= \frac{\frac{m}{k} \cdot \left(\frac{m}{k} - 1\right)}{2} \cdot k \\
&= \frac{k \cdot \frac{m}{k} \cdot \left(\frac{m}{k} - 1\right)}{2} \\
&= \frac{m \cdot \left(\frac{m}{k} - 1\right)}{2} \\
&= \frac{\frac{m^2}{k} - m}{2} \\
&= \frac{1}{2} \cdot \left(\frac{m^2}{k} - m\right) \\
&= \frac{m^2}{2k} - \frac{m}{2} \\
&= \frac{m^2}{2k} - \frac{mk}{2k} \\
&= \frac{m^2 - mk}{2k} \\
&= \frac{m \cdot (m-k)}{2k}
\end{aligned}$$

Innerhalb dieser Töpfe, gilt wie bereits erwähnt die Sprachinzidenz, da dort jeweils alle Buchstaben um dieselbe Anzahl verschoben wurden.

Möchte man Paare zwischen den unterschiedlich Töpfen bilden, zwischen denen aufgrund der unterschiedlichen Verschiebung die Zufallsinzidenz gilt, so muss man die Anzahl aller Paare von der Anzahl aller inner-

topflichen Paare abziehen:

$$\begin{aligned}
\text{Anzahl aller Paare zwischen verschiedenen Töpfen} &= \frac{m\cdot(m-1)}{2} - \frac{m\cdot(m-k)}{2k} \\
&= \tfrac{1}{2}\cdot\left(m\cdot(m-1) - \tfrac{m\cdot(m-k)}{k}\right) \\
&= \tfrac{1}{2}\cdot\left(m^2 - m - \tfrac{m^2-mk}{k}\right) \\
&= \tfrac{1}{2}\cdot\left(m^2 - m - \tfrac{1}{k}\cdot(m^2-mk)\right) \\
&= \tfrac{1}{2}\cdot\left(m^2 - m - \tfrac{m^2}{k} + m\right) \\
&= \tfrac{1}{2}\cdot\left(m^2 - \tfrac{m^2}{k}\right) \\
&= \tfrac{1}{2}\cdot\left(\tfrac{m^2k}{k} - \tfrac{m^2}{k}\right) \\
&= \tfrac{1}{2}\cdot\left(\tfrac{m^2k-m^2}{k}\right) \\
&= \tfrac{1}{2}\cdot\left(\tfrac{m^2(k-1)}{k}\right) \\
&= \tfrac{m^2(k-1)}{2k}
\end{aligned}$$

Somit ergibt sich für die Zahl der Paare aus gleichen Buchstaben folgende Formel:

$$\frac{m(m-k)}{2k}\cdot I_d + \frac{m^2(k-1)}{2k}\cdot I_r$$

Möchte man nun die Wahrscheinlichkeit für Paare aus gleichen Buchstaben, so muss man noch durch die Anzahl aller Paare teilen:

$$\begin{aligned}
\frac{\frac{m(m-k)}{2k}\cdot I_d + \frac{m^2(k-1)}{2k}\cdot I_r}{\frac{m(m-1)}{2}} &= \left(\tfrac{m(m-k)}{2k}\cdot I_d + \tfrac{m^2(k-1)}{2k}\cdot I_r\right)\cdot\tfrac{2}{m(m-1)} \\
&= \tfrac{m(m-k)}{2k}\cdot\tfrac{2}{m(m-1)}\cdot I_d + \tfrac{m^2(k-1)}{2k}\cdot\tfrac{2}{m(m-1)}\cdot I_r \\
&= \tfrac{m(m-k)}{k}\cdot\tfrac{1}{m(m-1)}\cdot I_d + \tfrac{m^2(k-1)}{k}\cdot\tfrac{1}{m(m-1)}\cdot I_r \\
&= \tfrac{m(m-k)}{m(m-1)k}\cdot I_d + \tfrac{m^2(k-1)}{m(m-1)k}\cdot I_r
\end{aligned}$$

Dieses Ergebnis entspricht ungefähr der gemessenen Inzidenz I im gesamten Text.

Berechnen können, oder anderweitig kennen wir die Parameter I_d, I_r, I, sowie m – die einzige Unbekannte dieser Gleichung ist das k, also die Länge des Passwortes. Also lösen wir die Formel dementsprechend auf:

$$k \approx \frac{(I_d - I_r)m}{(m-1)I - I_r m + I_d}$$

Somit haben wir nur durch abzählen der Häufigkeiten aller Geheimtextzeichen eine Annäherung für k. Dese ist jedoch meist eine Dezimalzahl, was keiner tatsächlichen Länge eines Passwortes entspricht, aber dieser Annäherungswert bietet eine Entscheidungshilfe, welcher der Primfaktoren aus dem Kasiski-Test der richtige sein könnte.

4 Die Enigma

Die Enigma war eine der ersten moderneren Verschlüsselungen, die großflächig zum Einsatz kam. Ursprünglich zur Geheimhaltung von Geschäftsgeheimnissen konzipiert, wurde sie schnell von der Armee übernommen und für den Krieg eingesetzt.

Die Funktionsweise der Enigma basiert im Kern auf drei Walzen und einem Reflektor, diese sind Intern verdrahtet, so dass jede der 26 Öffnungen auf der einen Seite der Walze ein entsprechendes Pendant auf der anderen Seite besitzt. Wird ein Buchstabe eingegeben, so durchfließt der Strom die verschiedenen Verkabelungen der unterschiedlichen Walzen, wird vom Reflektor auf eine vorbestimmte andere Öffnung zurück-gelenkt und nimmt einen anderen Weg zurück, bis man schließlich wieder beim Eingabegerät angekommen ist. Nachdem ein Buchstabe verschlüsselt wurde, wird eine Walze um $\frac{1}{26}$ weiterbewegt. Hat sich die erste Walze ein mal komplett um sich selbst gedreht, wird die zweite Walze ebenfalls gedreht – diese Funktionsweise ist vergleichbar mit einer Uhr und ihren Sekunden-, Minuten- und Stundenzeigern. Somit entspricht die Verschlüsselung der Enigma einer Vigenère-Chiffre mit einer Passwortlänge von $26^3 = 17.576$ – bei dieser Passwortlänge sind Kasiski- und Friedmann-Test nutzlos, da es kaum zu Wiederholungen des Passwortes kommt. Zudem sind die Walzen austauschbar, sodass es 6 verschiedene Anordnungsmöglichkeiten gibt, was den Schlüsselraum entsprechend versechsfacht. Noch später gab es insgesamt fünf Walzen, von denen man drei auswählen konnte, die dann auf sechs verschiedene Arten angeordnet werden konnten.

Des weiteren konnten die Eingabebuchstaben über ein Steckbrett „vertauscht“ werden. Verdrahtete man also B und Q miteinander, wurde erst der Strom von B zu Q geleitet und erst dort mit dem Verschlüsseln begonnen. So konnten bis zu 13 zusätzliche Drähte gesteckt werden. Jeder gesteckte Draht entspricht einem Ziehen ohne Zurücklegen von zwei Buchstaben aus allen übriggebliebenen Buchstaben. Beim Ersten Draht sind dies 26, beim zweiten 24 us. Somit ergibt sich die Formel der Steckmöglichkeiten k_s in Abhängigkeit von der Kabelanzahl p: $k_s(p) = \binom{26}{2} \cdot \binom{24}{2} \cdot \dots \cdot \binom{26-2\cdot(p-1)}{2}$. Dies sind allerdings geordnete Paare; Da der Enigma egal ist, welches Kabel zuerst gesteckt wurde, sondern sich nur dafür interessiert, welche Kabel am Ende stecken, muss man also die Anzahl der Permutationen dieser Steckreihenfolgen wieder herausrechnen, indem man durch diese dividiert. Somit ergibt sich die Formel:

$$k_s(p) = \frac{1}{p!} \prod_{i=1}^{p} \binom{26 - 2(i-1)}{2}$$

Möchte man alle möglichen Steckmöglichkeiten für jede beliebige Anzahl an Drähten, so muss man über diese Summieren und es ergibt sich folgende Anzahl an Möglichkeiten:

$$k_s = \sum_{p=0}^{13} k_s(p) \approx 5 \cdot 10^{14}$$

Diese Anzahl erweitert den bestehenden Schlüsselraum zusätzlich, sodass sich insgesamt $k = 6 \cdot 17.576 \cdot 5 \cdot 10^{14} \approx 6 \cdot 10^{19}$ Möglichkeiten gab, ohne dass man die Auswahl aus fünf Walzen hatte.

4.1 Kryptanalyse

Zwei bedeutende Schwächen der Enigma waren, dass sie aufgrund ihrer Verdrahtung

1. niemals ein Zeichen auf sich selbst abbildete, und
2. bei gleicher Walzenstellung Buchstabenpaare bildet, die sich „ineinander verschlüsseln“, es also gilt $f \mapsto p \Rightarrow p \mapsto f$.

Beide Eigenschaften schränkten die Möglichkeiten enorm ein.

zudem haben gewisse Fehlentscheidungen der Heeresleitung (Standardisierte Nachrichten, erbeutete Codebücher etc.) das Knacken der Enigma glücklicherweise zusätzlich erleichtert.

Zudem hatte jeder Sprecher die Anweisung, pro Nachricht einen anderen Schlüssel zu verenden, den er vor Beginn der eigentlichen Übertragung zwei Mal (verschlüsselt mit dem aktuellen Tagesschlüssel aus dem Codebuch) ansagte, damit Zuhörer die Walzenstellung ganz sicher mitbekommen und umstellen können. Durch das Wiederholen der drei Walzenstellungen wissen Kryptanalytiker, dass der 1. und der 4. Buchstabe, der 2. und der 5., sowie der 3. und der 6. Buchstabe gleich sein müssen. Somit können hier die Buchstabenabstände zwischen den jeweiligen verschlüsselten Zeichen gemessen werden.

Nun wird jeweils nach Zyklen gesucht. Ein A wird z.B. auf ein F abgebildet, ein F auf ein W, ein W auf ein P und das P wieder auf ein A. Somit hatte der Zyklus eine Länge von 4. Nun wird der Erste Buchstabe aus dem Alphabet gewählt, der noch nicht in diesem Zyklus vorhanden ist (hier B) und dort erneut der Zyklus ermittelt. Dies wird so lange ermittelt, bis alle Buchstaben in einem Zyklus auftauchen. Von diesen Zyklen hat man nun entsprechend die Längen (z.B. 4,4,9,9). Diesen Vorgang wiederholt man für alle (1-4, 2-5,3-6) Buchstabenpaare, sodass man drei Listen an Zyklenlängen hat.

Diese Listen sind so etwas wie ein Fingerabdruck für Walzenstellungen und -anordnungen, da nur sehr wenige Kombinationen dieselben Zyklen reproduzieren. Welche Buchstaben in den Zyklen enthalten sind, ist hierbei unwichtig, da diese durch das Steckbrett immer wieder verändert werden können.

Marian Rejewski erstellte eine Tabelle von allen Möglichen Anordnungen und Einstellungen der Walzen und den entsprechenden Zyklen-Listen. Somit musste man nur noch in der Tabelle nach den Zyklen-Listen-Paaren suchen und die wenigen Treffer ausprobieren. Schon war die Verschlüsselung geknackt.

5 Das One-Time-Pad

Das One-Time-Pad ist im Grunde nichts anderes als eine Vigenère-Chiffre mit unendlich langem Passwort. Heutzutage wird hauptsächlich binär mittels XOR ($z \mapsto (z + r) \mod 2 = z \oplus r$) verschlüsselt, dessen Ergebnistabelle in Tabelle 5.1 dargestellt ist.

Somit werden (echt) zufällig Nullen und Einsen generiert und diese Bit für Bit mit dem Klartext verknüpft (Das One-Time-Pad ist somit also eine Stromchiffre). Mit demselben Schlüssel wird der Text dann entsprechend wieder entschlüsselt.

Definition 5.1: Perfektes Chiffriersystem

Ein Chiffriersystem heißt *perfekt*, wenn bei beliebigem Klartext M und beliebigem Chiffretext C die a-priori-Wahrscheinlichkeit $p(M)$ (Wie groß ist die Wahrscheinlichkeit, dass ein gewisser Buchstabe b im Klartext M vorkommt?) gleich der bedingten Wahrscheinlichkeit (a-posteriori-Wahrscheinlichkeit) $p(M|C)$ (Wie groß ist die Wahrscheinlichkeit, dass der Buchstabe $b \in M$ der verschlüsselte Buchstabe in C ist?) ist. Das heißt

$$p(M|C) = p(M)$$

Also: Die Wahrscheinlichkeiten, wie oft ein Chiffretext-Buchstabe und wie oft ein Klartext-Buchstabe vorkommen, sollen voneinander unabhängig sein. (Man soll also nicht wissen „Wenn der Chiffretext-Buchstabe ein A ist, dann ist der Klartextbuchstabe mit höherer Wahrscheinlichkeit ein q“)

Aus obiger Formel lässt sich ableiten:

$$p(M|C) = \frac{p(M \wedge C)}{p(C)} = p(M) \Leftrightarrow p(M \wedge C) = p(M) \cdot p(C)$$

Definition 5.2: (echte) Zufallsbitfolge

Eine Zahlenfolge a_n mit $a_i \in \{0, 1\}$ heißt *(echte) Zahlenbitfolge*, wenn die Werte 0 und 1 jeweils mit einer Wahrscheinlichkeit von $\frac{1}{2}$ vorkommen, und wenn es keine Möglichkeit gibt, aus der Kenntnis eines beliebig langen Anfangsstücks der Folge Informationen über den Rest der Folge abzuleiten.

Sei X eine Zufallsvariable zur Erzeugung einer echten Zufallsbitfolge, dann gilt

$$\begin{aligned} & P(x \oplus 0 = 0) = P(x \oplus 0 = 1) = \tfrac{1}{2} \\ \text{und} \quad & P(x \oplus 1 = 0) = P(x \oplus 1 = 1) = \tfrac{1}{2} \end{aligned}$$

Denn $x \oplus 0$ ist immer x ($1 \oplus 0 = 1; 0 \oplus 0 = 0$). Somit muss nur noch die Wahrscheinlichkeit für $x \oplus 1$ bewiesen werden. Hier sehen wir in der Ergebnistabelle 5.1, dass der Wert von x einfach umgekehrt wird – somit bleiben die Wahrscheinlichkeiten gleich.

Bei einer echten Zufallsbitfolge bietet das One-Time-Pad also perfekte Sicherheit. Dies liegt daran, dass bei einer Länge n alle Klartexte $\mathcal{M}$, alle Chiffretexte $\mathcal{C}$ und alle Schlüssel $\mathcal{K}$ bei einem One-Time-Pad dieselbe

$z \oplus r$		r	
		1	0
z	1	0	1
	0	1	0

Tabelle 5.1: Ergebnistabelle für $z \oplus r$

Mächtigkeit haben:

$$|\mathcal{M}| = |\mathcal{C}| = |\mathcal{K}| =: m = 2^n$$

da alle Mengen über einem vorgegebenen Alphabet die Länge n haben. Da der Schlüssel Zufällig aus m Möglichkeiten generiert wird, gibt es entsprechend m verschiedene Chiffretexte – diese entstehen alle Aufgrund der zufallsgenerierten Schlüsselwörter mit einer Wahrscheinlichkeit von $p(C) = \frac{1}{m}$.Somit gilt auch $p(C|M) = \frac{1}{m}$.

Nach Definition der bedingten Wahrscheinlichkeiten gilt

$$p(C|M) \cdot p(M) = p(M|C) \cdot p(C) \Rightarrow \frac{1}{m} \cdot p(M) = p(M|C) \cdot \frac{1}{m} \Rightarrow p(M) = p(M|C)$$

Somit gilt, dass die Texte statistisch voneinander unabhängig sind, was die Voraussetzung für eine perfekte Chiffre ist.

So sicher das One-Time-Pad auch sein mag, so ist es nur schwer möglich, echte Zufallszahlen zu generieren, Um dies zu bewerkstelligen sind zufällige physikalische Prozesse notwendig, wie z.B. ein Geigerzähler oder thermisches Rauschen. Zudem muss der Kommunikationspartner exakt dieselbe Schlüsselabfolge besitzen wie der Absender, dies benötigt einen sichern Schlüsselaustausch, welcher sich durchaus kompliziert gestalten kann.

Eine Möglichkeit, dies zu umgehen ist es einen zentralen Strom an Zufallsbits bereitzustellen (z.B. über einen Satelliten), von dem jeder dieselben zufällig generierten Zahlen beziehen kann. Nun kann im Vornherein mittels normaler Verschlüsselung ein Zeitpunkt vereinbart werden, welche gesendeten Bits zum Verschlüsseln der Nachricht verwendet werden (z.B. Die Bits ab 4:26.09). Beide Partner lesen die Bits zu diesem Zeitpunkt aus und haben somit dieselben zufällig generierten Schlüssel.

Ein Angreifer kennt diesen Zeitpunkt nicht – Er müsste die Bits aus einem beliebigen Zeitraum aufzeichnen und ausprobieren, ob beim Entschlüsseln mit diesen ein sinnvoller Text herauskommt, oder die ursprüngliche Nachricht mit dem Zeitpunkt knacken. Bei genügend hoher Datenrate des Generators übersteigt die Anzahl der zu speichernden Bits schnell die üblicherweise zur Verfügung stehenden Speicherkapazitäten.

Somit wäre ein One-Time-Pad fast ohne Schlüsseltausch realisiert. Das Problem hierbei ist jedoch der Zufallsgenerator, welcher von irgendjemandem betrieben werden muss. Ist diese Person oder Organisation nicht vollkommen vertrauenswürdig (wie z.B. die NSA), so kann es sein, dass die Zufallszahlen manipuliert werden, und der Betreiber des Generators die Zahlen immer nachrechnen und somit jede Nachricht entschlüsseln kann.

6 Moderne Blockchiffren

Alle bisher vorgestellten Chiffren waren *Stromchiffren*, also Chiffren, die einen Strom an Daten als solchen verschlüsseln – Zeichen nach Zeichen wird nacheinander verschlüsselt. *Blockchiffren* hingegen verschlüsseln, wie der Name vermuten lässt, ganze Zeichenblöcke (z.B. 64 Bit) auf einmal.

6.1 Data Encryption Standard (DES)

DES verschlüsselt immer in 64-Bit-Blöcken, die es in eine Rechte und eine Linke Hälfte mit je 32 Bit aufteilt. Zu Beginn und am Ende werden sie jeweils einer Permutation unterzogen – diese hat keinen Kryptographischen Wert; Es müssen bei der hardwarebasierten Implementierung lediglich die Register befüllt werden, und um potentielle Angreifer zu verwirren, wurden die Permutationen eingebaut.

Die Anfang- und Abschlusspermutation sind invers – heben sich also gegenseitig auf. Dies wurde mit einer Tabelle realisiert. Zu Beginn wird in der Tabelle mit 64 Einträgen für jedes der 64 Zeichen im Block nachgeschaut, an welche Stelle dieses denn verschoben werden soll (.B. Zeichen 5 wird an Stelle 17 gespeichert). Für die Abschlusspermutation gibt es ebenfalls so eine Tabelle, bei der die Werte aber entsprechend komplementär sint (z.B. Zeichen 17 wird an Stelle 5 gespeichert).

Zwischen diesen beiden Permutationen, verschlüsselt DES den Block in 16 Runden. Jede Runde verläuft wie folgt:

Die Linke Blockhälfte ist die Reche Blockhälfte der Vorrunde. Die rechte Blockhälfte ist das Ergebnis aus der linken Blockhälfte XOR der Rechten Blockhälfte, welche zuvor durch eine Funktion modifiziert wurde. Diese Funkion arbeitet zudem mit einem Teil des Schlüssels, von dem ebenfalls jede Runde ein neuer Teil gewählt wird. Es gilt also:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

Die Funktion f funktioniert wie folgt: Zuerst wird eine *Expansionspermutation* durchgeführt, bei der die 32-Bit-Hälfte auf 48 Bit verlängert wird. Auch hier wird wieder eine Tabelle verwendet, diesmal mit 48 Einträgen. Jeder dieser Einträge sagt, welches Bit aus der Originalkette an der entsprechenden Stelle eingetragen werden soll. Logischerweise werden hier einige Bits doppelt eingetragen, damit die 48 Zeichen erreicht werden.

Diese Expandierte Folge wird nun mit einem Teilschlüssel XORt. Der Teilschlüssel wird aus zwei gleichgroßen 28-Bit teilen des Schlüssels generiert. Eigentlich ist er 64 Bit lang, 8 Bits sind aber lediglich Kontrollbits und haben somit keinen kryptographischen Wert. Diese beiden Teile werden nun noch einem Bitshift unterzogen. Um welchen Wert geshifted wird, ist von Runde zu Runde unterschiedlich – dies steht ebenfalls in einer Tabelle definiert. In jeder Runde werden alle Shifts der vorherigen Runden und der neue Shift durchgeführt. Dann wird der 56 Bit lange Teilschlüssel noch durch eine Kompressionsmutation ebenfalls auf 48 Bit verkürzt. Auch hier wird eine Tabelle ähnlich der Expansionsmutation eingesetzt, bei der allerdings entsprechend einige Bits herausfallen, anstatt sie zu duplizieren.

Nun kommt es zum Kernstück des Algorithmus: Der *S-Box-Substitution*. Hierbei wird der Block in acht 6-Zeichen-Blöcke aufgeteilt, die jeweils einer S-Box Zugewiesen werden. Jede der 8 verschiedenen S-Boxen ist eine Tabelle mit drei Zeilen und 16 Spalten und enthält jeweils Werte zwischen 0 und 15 (je inklusive). Das erste und letzte Bit des 6-Bit-Blocks geben binär gelesen die Zeile an, in der nachgeschlagen werden soll, die anderen 4 Bits geben die entsprechende Spalte an.

Dies wird für jeden Block gemacht. Ergebnis sind acht 4-Bit-Blöcke, da der Intervall [0;15] mit nur 4 Bit dargestellt werden kann. Somit gehen die 2 Bits, die zuvor hinzugefügt wurden nun wieder verloren und wir haben wieder 32 Bit.

Zu guter Letzt wird noch eine sogenannte *P-Box-Permutation* durchgeführt, die einfach wieder die Einträge anhand einer Tabelle vertauscht. Nun ist f fertig, und das Ergebnis kann gemäß der Runden jeweils mit der linken Hälfte XORt werden.

Zurück funktioniert der Algorithmus fast gleich. Man hat den Chiffretext, wendet die Anfangspermuation an und teilt den Block in zwei Hälften. Nun besteht allerdings das Problem, das XOR rückgängig zu machen, bzw. auf die Linke Hälfte der Vorrunde zu schließen. Dies kann allerdings recht einfach gelöst werden. Die aktuelle linke Hälfte ist die rechte Hälfte der Vorrunde. Diese kann man nun in f hineingeben – hierbei werden die Schlüsselteile aber genau in die andere Richtung rotiert, um dieselben Teilschlüssel zu erhalten. Nun hat man die eine Hälfte für das XOR, aber die Linke fehlt. Aufgrund der Eigenschaften der XOR-Operation ist es allerdings möglich, die vorherige Rechte Hälfte (inkl. f) mit der aktuellen rechten Hälfte zu verknüpfen, und so die linke Hälfte zu erhalten. So kann man nun also die ganzen Runden rückwärts ablaufen, bis man wieder bei Runde 1 angelangt ist und die Abschlusspermutation anwendet.

Dieses „Spiralartige“ vorgehen ist Merkmal von sogenannten *Feistelchiffren*.

6.1.1 Sicherheit von DES

Ein entscheidender Faktor bei der Sicherheit von DES war die Konstruktion der S-Boxen. Wichtig ist es hierbei, dass eine kleine Veränderung des Klartextes zu einer großen Veränderung des Chiffretextes führen muss; Dies wird auch als *Lawineneffekt* bezeichnet.

Dies bedeutet, dass die S-Boxen nicht linear sein dürfen, denn lineare Funktionen sind ähnlichkeitserhaltend – kleine Änderungen im Klartext führen also zu kleinen Änderungen im Chiffretext. Dies ist z.B. bei den Permutationen der Fall; Diese sind sehr leicht umkehrbar und somit Kryptographisch unbrauchbar.

Ob eine Funktion linear ist lässt sich auch durch ihre Definition berechnen, welche besagt, dass der Funktionswert zweier verknüpfter Werte derselbe sein muss, wie die verknüpften Funktionswerte dieser Werte:

$$f(x \circ y) = f(x) \circ f(y)$$

Wählen wir als Verknüpfung das XOR und testen es mit zwei unterschiedlichen Bitfolgen bei den S-Boxen, so sehen wir, dass die Werte nicht übereinstimmen und die S-Box-Substitution somit nicht linear ist.

1996 konnte ein Rechner ca. 196.000 DES-Blöcke pro Sekunde ausprobieren; Heutzutage sind es über eine Million Blöcke pro Sekunde. Trotz dieser Geschwindigkeit beträgt die Rechenzeit für den Kompletten Schlüsselraum von 2^{56} noch 1.142 Jahre. Dennoch gelang es in 1999 mit einem Netz an 100.000 Rechnern und einem Spezialrechner einen Chiffretext innerhalb von 22 Stunden zu knacken.

6.1.2 Triple-DES

Der Schwachpunkt von DES war der kleine Schlüsselraum, welcher durch die NSA so vorgegeben wurde. Ein Workaround, um DES dennoch weiterhin sicher einsetzen zu können ist der Triple-DES, bei dem eine Nachricht drei Mal hintereinander mit zwei verschiedenen Schlüsseln verschlüsselt wurde. Damit hatte sich der Schlüsselraum auf 2^{112} verlängert. Streng genommen, wird beim Triple-DES zuerst Mit Schlüssel 1 Verschlüsselt, dann mit Schlüssel 2 entschlüsselt, und dann wieder mit Schlüssel 1 verschlüsselt, aber da die Entschlüsselung nicht mit demselben Schlüssel passiert wie die vorherige Verschlüsselung, kommt es einer weiteren Verschlüsselung gleich. Es gilt also

$$C = E_{K_1}(D_{K_2}(E_{K_1}(M))) \text{ und } M = D_{K_1}(E_{K_2}(D_{K_1}(C)))$$

Viel intuitiver wäre es allerdings, DES einfach nur zwei mal hintereinander anzuwenden. Diese Vorgehensweise ist allerdings anfällig für sogenannte „Meet in the middle“Angriffe. Das sind Known-Plaintext-Angriffe, bei denen der Hacker den Plaintext mit allen 2^{56} möglichen Schlüsseln ein Mal verschlüsselt, und den Chiffretext mit allen 2^{56} möglichen Schlüsseln entschlüsselt. Alle Ergebnisse werden jeweils in einer eigenen Tabelle/Datenbank gespeichert und dann verglichen. Diese Ergebnisse sind der Text, der zwischen der 1. und der 2. Verschlüsselung entsteht. Findet man nun in den beiden Tabellen dasselbe Zwischenergebnis, hat man das passende Schlüsselpaar für die Verschlüsselung. Es kann sein, dass mehrere Schlüsselpaare passen, die den Text entschlüsseln – aber nur genau diesen Text. Nimmt man ein Zweites Plaintext-Chiffretext-Paar her, kann man dort die bisher gefundenen Paare durchprobieren: Es wird nur ein Schlüsselpaar geben, das beide Texte ver- und entschlüsseln kann.

Somit ist die Komplexität beim Double-DES nicht wie erwartet 2^{112} sondern selbst im Worst Case nur $2 \cdot 2^{56}$. Im Average Case findet man, nachdem man von der ersten Richtung alle Möglichkeiten berechnet hat, schon nach der Hälfte aller Kombinationen der anderen Seite die richtige, sodass sich ergibt: $2^{56} + (2^{56} : 2) = 2^{56} + 2^{55}$ Triple DES löst dieses Problem, da wir uns beim Angriff von vorne und von

r	b = 128	b = 192	b = 256
k = 128	10	12	14
k = 192	12	12	14
k = 256	14	14	14

Tabelle 6.1: Die Anzahl der Verschlüsselungsrunden r von AES in Abhängigkeit von Schlüssel- und Blockgröße k und b

	b = 128	b = 192	b = 256
Reihe 1	1	1	1
Reihe 2	2	2	3
Reihe 3	3	3	4

Tabelle 6.2: Die Anzahl der Shifts von AES in Abhängigkeit der Blockgröße b

hinten nicht mehr in der Mitte treffen, sondern jeweils links und rechts von Schritt 2 stehen. Möchte man trotzdem diese Vorgehensweise verfolgen, so muss von einer Seite weitergerechnet werden und für alle 2^{56} Schlüsselkombinationen weitere 2^{56} Kombinationen ausprobiert werden. Erst hier hat sich der Aufwand zum Verdoppeln also im Worst Case auf $2^{56} \cdot 2^{56} + 2^{56} = 2^{112} + 2^{56}$ erhöht. Im Average Case muss zum endgültigen finden der richtigen Kombination im letzten Schritt wieder nur noch ca. die Hälfte aller Kombinationen ausprobieren, sodass sich ergibt $2^{56} \cdot 2^{55} + 2^{56} = 2^{111} + 2^{56}$

6.2 Advanced Encryption Standard (AES)

Als sichere Weiterentwicklung des DES wurde der *Advanced Encryption Standard (AES)* entwickelt. Dieser konnte mit verschiedenen Block- und Schlüssellängen von jeweils 128, 192 oder 256 Bit arbeiten. Auch der AES arbeitet in „Verschlüsselungsrunden“; Die Anzahl der Runden hängt jeweils von der Länge ab, wie in Tabelle 6.1 zu sehen ist. Zu Beginn werden sowohl der zu verschlüsselnde Block und der Schlüssel in 8-Bit-Blöcke aufgeteilt, welche dann gleichmäßig in 4 Reihen bzw. Zeilen einer Matrix aufgeteilt werden (Die Anzahl der Spalten hängt entsprechend von der Block-/Schlüssellänge ab). Die Matrix des Blocks wird auch *Zustand* genannt.

Dann wird der Klartextblock mit dem ersten Rundenschlüssel XORt und liefert den Input für die erste Verschlüsselungsrunde. Das Ergebnis wird mit dem zweiten Rundenschlüssel XORt und fließt in die Dritte Runde ein, usw.

Innerhalb einer Runde werden Drei Operationen Durchgeführt: Die Byte-Substitution, einen RowShift und einen Column-Mix. In der Byte-Substitution werden die einzelnen Bytes gemäß eines im Standard festgelegten Galois-Körpers (Siehe Punkt 10.6) als Polynome betrachtet und ihr Inverses berechnet. Dann wird jedes invertierte Byte noch mit einer vorgegebenen Matrix (bzw. einem anderen Polynom) multipliziert und ein weiterer Vektor wird hinzuaddiert. Dies lässt sich mit einer linearen Abbildung vergleichen: $\vec{y} = A \cdot \vec{x}^{-1} + \vec{b}$. Umkehren lässt sich die Gleichung entsprechend, indem $\vec{b}$ wieder abgezogen wird, der Rest mit dem Inversen von A multipliziert wird und dann wieder das Inverse der einzelnen Bytes berechnet wird.

Die nachfolgende Shift-Row-Operation verschiebt die Zahlen der Zeilen 1 bis 3 (0 bleibt unberührt) zyklisch nach links. In welchen Reihen wie viel nach Links verschoben wird, ist in Tabelle 6.2 zu sehen. Die Umkehrung sind entsprechend viele Shifts nach rechts.

Zuletzt werden auch noch die Spalten durch die Multiplikation mit einer Vorgegebenen Matrix vertauscht. Die Umkehrung ist entsprechend das Multiplizieren mit der Inverse.

Dieses Ergebnis wird nun wieder mit dem Rundenschlüssel XORt – doch wie kommt dieser Zustande? Hierfür werden der Reihe nach immer 4 32-Bit-Worte aus dem Schlüssel entnommen (in der Ersten Runde also Worte 0 bis 4). Bei einem 192-Bit langen Schlüssel gibt es allerdings nur 6 solche Worte. Es müssen also weitere Worte erzeugt werden; Dies geschieht, indem das vorhergehende Wort mit dem Wort XORt wird, das so weit entfernt ist, wie der Originalschlüssel Worte lang ist (z.B. $W_6 = W_5 \oplus W_0$ bei dem 6-Wort langen Schlüssel). Ist das vorhergehende Wort (in unserem Beispiel W_5) ein vielfaches der Schlüsselwortanzahl, so wird dieses erst noch einer nicht-linearen Substitution unterworfen (u.A. durch das Byte-Substitution Verfahren)

Nun sind wir mit dem Algorithmus fertig. Invertieren lässt er sich ganz einfach, indem wir einfach mit den Inversen der festgelegten Werte rechnen.

6.2.1 Geschwindigkeit, Sicherheit und Sonstiges

Durch effiziente Implementierung (z.B. durch Lookup-Tables für Multiplikationsergebnisse, anstatt sie jedes Mal aufs neue zu berechnen) ist der AES trotz seiner Komplexität bei einer Blocklänge von 192 Bit und einer Schlüssellänge von 192 Bit etwa 3 Mal so schnell wie der DES (und entsprechend 9 Mal schneller als der Triple-DES).

Der AES wurde auf verschiedene Angriffsmöglichkeiten untersucht: Eine Input-Output Korrelation (die z.B. den Kasiski-Test ermöglicht) ist sehr schwach, und er ist auch gegen differentielle und lineare Kryptanalyse resistent. Auch alle Tests des Konkurrenten während der Entwicklung waren erfolglos.

Zudem lässt sich der AES auch als Message-Authentication-Code, Einweg-Hash-Funktion und (aufgrund der geringen Input-Output-Korrelation) als Pseudozufallszahlengenerator verwenden.

6.3 Betriebsmodi von Blockchiffren

6.3.1 Electronic Codebook (ECB) Modus

Blockhiffren können auf verschiedene Art und Weise angewandt werden. Die naivste Weise ist der ECP-Modus, der jeden Block entsprechend mit demselben Schlüssel verschlüsselt.

Sieht man die Blöcke als eigene Zeichen, so entspricht dieser Modus einer monoalphabetischen Chiffre, da jeder Plaintext-Block immer auf denselben Chiffre-Block abgebildet wird. So kann ein Angreifer sich wiederholende Textblöcke leicht identifizieren und kann auch gezielt Blöcke einfügen, löschen oder ersetzen.

6.3.2 Cipher Block Chaining (CBC) Modus

Um diese Schwachstelle zu beheben gibt es den CBC-Modus, der den Klartext jedes Blocks (außer dem ersten) mit dem Chiffretext der vorhergehenden Blocks XORt, bevor er verschlüsselt wird. Somit ist jeder Block abhängig von allen Blöcken, die davor kommen. Somit ist es also nicht möglich, Blöcke nach belieben zu manipulieren, da dies bei der (nicht mehr funktionierenden) Entschlüsselung auffällt.

6.3.2.1 Initialisierungsvektor (IV)

Bei gleichem Klartext sieht der entsprechende Chiffretext immer noch gleich aus. Um diesen information leak ebenfalls zu verhindern, wird oft ein sogenannter *Initialisierungsvektor* als erster Block eingefügt. Dies ist ein Block, der zufällig mit Bits gefüllt wurde. So ist auch bei selbem Klartext der Chiffretext durch die verschiedenen IVs unterschiedlich.

6.3.3 Padding

Zuletst ergibt sich noch das Problem des Paddings. Ist der Text nicht ein vielfaches der Blockgröße lang, so muss der letzte Block noch aufgefüllt werden, damit die Blockchiffre damit arbeiten kann.

Nimmt man allerdings eine festgelegte Bitfolge (z.B. nur nullen), so ermöglicht man Kryptoanaysten einen Known-Plaintext-Angriff. Generiert man Zufallsbits, so muss man dem Empfänger mitteilen, wann die tatsächliche Nachricht aufhört und wann die Zufallsbits anfangen; Dies generiert wiederum Overhead.

Hier gilt es, das kleinere der beiden Übel auszuwählen.

7 Public-Key-Kryptographie

Wie in Punkt 2.3 beschrieben, kann es mit symmetrischen Verschlüsselungsverfahren schnell zu einer massiven Ansammlung verschiedener Schlüsselpaare kommen. Um diesem Problem, wie auch der Komplexität des Schlüsseltausches Herr zu werden, gibt es die *Public-Key-Kryptographie*, bei dem zwei unterschiedliche Schlüssel verwendet werden, um eine Nachricht zu ver- und entschlüsseln.

7.1 Merkles Rätsel

Vor Erfindung der Public-Key-Kryptographie gab es verschiedene Ansätze zum Austauschen von symmetrischen Schlüsseln; Einer davon war *Merkles Rätsel*.

Hierbei erzeugt eine Person zufällig n Nummern $\{x_1, \ldots, x_n\}$, sowie zwei mal n Schlüssel $\{y_1, \ldots, y_n\}$ und $\{k_1, \ldots, k_n\}$.

Nun verschlüsselt er alle zusammengehörenden Paare (x_i, y_1) mit dem entsprechenden Schlüssel k_i und speichert diese im sogenannten Rätsel und verschickt dieses über den unsicheren Kanal an seinen Kommunikationspartner. Die unverschlüsselten x, y-Paare speichert er ebenfalls ab, behält sie aber bei sich.

Der Kommunikationspartner wählt nun ein $j \in \{1, \ldots, n\}$ und knackt den Schlüssel k_j per Brute-Force und kann dann den Eintrag an der Stelle j lesen – er erhält also einen weiteren Schlüssel y_j sowie die zugehörige Zufallszahl x_j. Nun kann er eine Nachricht (meist ein weiterer Schlüssel, unter dem in Zukunft kommuniziert werden soll) mittel, y_j verschlüsseln, und sendet diesen Chiffretext, gemeinsam mit x_j wieder an den Ursprung zurück. Dieser sucht in seinen lokal gespeicherten Paaren nach der Nummer x_j und kann den Chiffretext mit dem entsprechenden Text entschlüsseln.

Dieses Verfahren scheint recht aufwändig, da beide Kommunikationspartner für n Schlüssel jeweils n Mal verschlüsseln bzw. Brute-Forcen müssen. Eine dritte Person muss hingegen alle Rätsel, also $n \cdot n$ lösen (bzw. im Schnitt nur die Hälfte), da sie nicht weiß, welches Rätsel die angegebene Zahl enthält.

7.2 Der RSA-Algorithmus

Der RSA-Algorithmus wurde 1970 als sicherer Nachfolger des DES entwickelt. Die Funktionsweise ist wie folgt:

Definition 7.1: Anwendung von RSA

Die Nachricht $M \in \mathbb{Z}_n$ wird codiert durch

$$E(M) = M^e \mod n$$

Und der entsprechende Chiffretext $C \in \mathbb{Z}_n$ wird entschlüsselt durch

$$D(C) = C^d \mod n$$

Definition 7.2: RSA-Schlüsselerzeugung

1. Wähle zufällig zwei große (zwischen 384 und 512 Bit) Primzalen p und q.
2. Berechne $n = p \cdot q$ (n ist dann 512 bis 1024 Bit lang)
3. Wähle eine kleine ungerade natürliche Zahl e, die zu $\varphi(n) = (p-1)(q-1)$ relativ prim ist (also $ggT(e, \varphi(n)) = 1$)
4. Berechne mittels des Euklidischen Algorithmus d als Lösung von $e \cdot d = 1 \mod \varphi(n)$
5. Gib das Paar $P = (e, n)$ als *öffentlichen Schlüssel* weiter
6. Halte das Paar $S = (d, n)$ als *geheimen Schlüssel* für dich

$\varphi(n)$ ist eine Funktion, die berechnet, wie viele zu n relativ Prime Zahlen es in $\mathbb{Z}_n$ gibt.

Definition 7.3: Korrektheit von RSA

Der RSA-Algorithmus ist korrekt. Das bedeutet $\forall M \in \mathbb{Z}_n$

$$D(E(M)) = E(D(M)) = M$$

Wie in der Definition über die Anwendung von RSA bereits erwähnt, erfolgt die Ver- und Entschlüsselung durch die Potenzierung mit d und e. Es gilt also

$$D(E(M)) = E(D(M)) = M^{ed} \mod n$$

Um die Korrektheit des RSA-Algorithmus zu beweisen, git es zu zeigen, dass $M^{ed} \equiv M \mod n$ ist. Da e und d per Konstruktion multiplikative inverse bezüglich $\varphi(n) = (p-1)(q-1)$, muss es also ein $k \in \mathbb{N}$ geben„ für das gilt

$$ed = 1 + k(p-1)(q-1)$$

In Werten ausgedrückt: Da e und d multiplikativ invers zu $\varphi(n)$ sind, muss das Produkt $\varphi(n)$ oder ein Vielfaches davon $+1$ sein.

Somit können wir (für $M \not\equiv 0 \mod p$) sagen

$$M^{ed} \equiv M^{1+k(p-1)(q-1)} \mod p$$

Aufgrund der Potenzgesetze ($a^{b+c} = a^b \cdot a^c$ und $a^{bc} = (a^b)^c$) Können wir die Gleichung umformen nach

$$M(M^{p-1})^{k(q-1)} \mod p$$

Da p eine Primzahl ist können wir aufgrund des Satzes von Fermat aus Punkt 10.4 sagen, dass $M^{p-1} \mod p = 1$. Somit gelangen wir zu der Gleichung

$$M \cdot 1^{k(q-1)} \mod p$$

Und da eine 1 potenziert immer 1 bleibt ergibt sich schließlich

$$M \cdot 1^{k(q-1)} \mod p = M \mod p$$

Dieselbe Schlussfolgerung gilt auch für q.

Nun gibt es den chinesischen Restsatz, der besagt, dass für zwei Primzahlen p und q gilt

$$x \equiv a \mod p \wedge x \equiv a \mod q \Leftrightarrow x \equiv a \mod pq$$

Daraus folgt $M^{ed} \equiv M \mod pq = M \mod n$, was zu beweisen war.

7.2.1 Sicherheit von RSA

Die Sicherheit von RSA hängt stark von der Zerlegung der Primfaktoren ab. Wie in Punkt 10.5 beschrieben, muss n erst in seine Primfaktoren zerlegt werden, bevor man d berechnen kann. Diese Faktorisierung ist allerdings sehr zeitaufwändig und steigt nach bisherigem Wissen exponentiell (z.B. muss n durch alle Primzahlen $\leq \sqrt{n}$ geteilt werden, und herauszufinden, ob diese Teiler von n sind). Experten glauben nicht, dass es eine effizientere (polynomielle) Methode gibt, dies zu erreichen (, bewiesen ist es aber nicht). Sofern diese Annahme wahr ist, beruht die Sicherheit von RSA auf dem großen Aufwand der Faktorisierung.

7.2.2 Effiziente Primzahltests

Eines der Probleme von RSA ist es, dass zur Generierung der Schlüssel zwei Primzahlen benötigt werden. Primzahlen zu finden, ist allerdings ebenfalls sehr aufwändig. Um diese Suche einfacher zu gestalten, gibt es den *Test von Miller und Rabin*.

Hierfür wird zufällig eine erlaubte Zahl n ausgewählt, Erlaubt sind Zahlen, die ungerade sind und bei denen auch $\frac{n-1}{2}$ ungerade ist. Nun wird eine zufällige Zahl $a \in \mathbb{Z}_n \backslash \{0\}$ gewählt und mit $\frac{n-1}{2}$ potenziert. Ist n eine Primzahl, so ist das Ergebnis dieser Potenz ± 1.

Dies liegt am Satz von Fermat, der besagt, dass $a^{n-1} = 1$ ist. Dies kann man mittels der dritten Binomischen Formel entsprechend umformen, sodass sich ergibt

$$(a^{\frac{n-1}{2}} + 1)(a^{\frac{n-1}{2}} - 1) = a^{n-1} - 1 = 0$$

Die Aufspaltung in diese geringeren Potenzen spart sehr viel Berechnungszeit.

Ist das gewählte n hingegen keine Primzahl, so ergibt sich für die gewählten $a \in \mathbb{Z}_n \backslash \{0\}$ nur in weniger der Hälfte aller Fälle ± 1. Wählt man also einfach genug as und testet diese, so kann man sich recht sicher sein, dass das gewählte n eine Primzahl ist, wenn man nur genug as erfolgreich testet.

7.2.3 Implementierung von RSA

Das Potenzieren von Zahlen ist sehr rechenaufwändig, da bei einer Potenz von z.B. 19, 18 Mal potenziert werden muss. Mit der sogenannten *square-multiply*-Methode lässt sich diese Zahl besonders bei großen Exponenten drastisch verringern.

Zuerst wird der Exponent binär dargestellt (z.B. $19 = 10011$), und dann wird der zu Verschlüsselnde Wert a für jede Stelle der Binärzahl ein Mal quadriert. Ist die Stelle 1, so wird sie davor nochmals mit a multipliziert. Der ursprüngliche Wert für die Berechnung ist 1.

$$\left(\left(\left((1 \cdot a)^2\right)^2\right)^2 \cdot a\right)^2 \cdot a = a^{19}$$

Diese Methode ist zwar effizienter, da sie nicht so viele Multiplikationen benötigt, allerdings hat sie nach wie vor eine Laufzeit von $\mathcal{O}(k^3)$, wobei k die Bitlänge der Werte ist. Zudem kann z.B. auf Smartcards der geheime Schlüssel ausgelesen werden, wenn man den Stromverbrauch während der Potenzierung betrachtet; Denn für Potenzieren und Multiplizieren wird mehr Strom Benötigt, als für die Potenzierung allein. So lässt sich an den längeren Stromspitzen eine 1 ablesen, an den kürzeren eine 0. Um diese Schwachstelle zu beheben, kann bei jeder Runde multipliziert werden; Die nicht benötigten Ergebnisse werden danach wieder verworfen. Damit ist zwar das Problem behoben, es gibt aber auch keinen Gewinn bei der Anzahl der Multiplikationen mehr.

Ein anderes Vorgehen ist das *blenden* des Angreifers, indem man nicht den normalen verschlüsselten Text übermittelt, sondern

1. Eine Zufallszahl $0 < r < n, ggT(r, n) = 1$ auswählt
2. $c' = cr^e \mod n$ berechnet, wobei e der öffentliche Exponent ist. c' wird schließlich übermittelt.
3. Die geblendete Nachricht erhält der Empfänger dann über $m' = (c')^d \mod n$
4. Schließlich wird das Blenden mittels $m = m'r^{-1} \mod n$ rückgängig gemacht.

Setzt man die Formeln ineinander ein, so ergibt sich, dass dieses Verfahren gültig ist:

$$\begin{aligned} m &= m'r^{-1} \mod n \\ &= ((c')^d)r^{-1} \mod n \\ &= ((cr^e)^d)r^{-1} \mod n \\ &= (c^d r^{ed})r^{-1} \mod n \\ &= c^d r r^{-1} \mod n \\ &= c^d \\ &= m \end{aligned}$$

7.2.4 Schnellere Implementierung von RSA

Das Verschlüsseln kann erneut (um den Faktor 4) beschleunigt werden, indem man den Zahlenraum verringert, in dem Potenziert wird. Hierfür wird der geheime Schlüssel verändert: Er speichert nun $(d, n, p, q, \overline{p})$, wobei d, n, p, q bereits zu Beginn erklärt wurden, und es muss gelten $p < q$ und $\overline{p} = p^{-1} \mod q$. Der öffentliche Schlüssel bleibt unverändert.

Das Verschlüsseln mit dem geheimen Schlüssel (also das Signieren von Dokumenten), verläuft nun in folgenden Schritten:

$$\begin{aligned} S_1 &= M^d \mod p \\ S_2 &= M^d \mod q \\ h &= \overline{p} \cdot (S_2 - S_1) \mod q \\ S &= (S_1 + p \cdot h) \mod n \end{aligned}$$

Die so berechnete Signatur entspricht der normal berechneten Signatur, denn

$$(S_1 + p \cdot h) \mod p = (S_1 + p(\overline{p} \cdot (S_2 - S_1) \mod q)) \mod p$$

Da wir innerhalb von $\mod p$ etwas mit p multiplizieren, ist das Ergebnis auf jeden Fall 0, sodass sich ph aus der Gleichung eliminiert. Deshalb gilt

$$(S_1 + p \cdot h) \mod p = S_1 \mod p = M^d \mod p$$

was unserer ursprünglichen Konstruktion entspricht. Für S_2 gilt dies ebenfalls zu zeigen, diesmal nehmen wir allerdings $\mod q$:

$$\begin{aligned} (S_1 + p \cdot h) \mod q &= (S_1 + p(\overline{p} \cdot (S_2 - S_1) \mod q)) \mod q \\ &= (S_1 + (p\overline{p}) \mod q \cdot (S_2 - S_1) \mod q) \mod q \end{aligned}$$

Da $\overline{p}$ und p multiplikativ invers bezüglich q sind, ergibt sich bei der Multiplikation eine 1, welche den Term $S_2 - S_1$ somit nicht beeinflusst. Also haben wir

$$\begin{aligned} (S_1 + p \cdot h) \mod q &= S_1 \mod q + S_2 \mod q - S_1 \mod q \\ &= S_2 \mod q = M^d \mod q \end{aligned}$$

Auch hier entspricht das Ergebnis unserem Konstruierten Wert. Somit haben wir für p und für q bewiesen, dass die Gleichung stimmt. Gemäß des chinesischen Restsatzes muss er also auch für pq und somit für n wahr sein.

Wichtig ist bei diesem Verfahren allerdings, p und q ebenfalls, so wie d geheimzuhalten, da damit alle benötigten Werte nachgerechnet werden können.

7.3 Schlüsseltausch

Obwohl asymmetrische Verschlüsselungen wie RSA sicher sind, ist es trotz effizienten Implementierungen schneller, symmetrische Verschlüsselung zu verwenden. Hier besteht jedoch das Problem der Schlüsselverwaltung.

7.3.1 Symmetrischer Schlüsseltausch

Ein Ansatz, der auch heute noch verwendet wird, ist es ein Trustcenter aufzubauen, dass für seine entsprechende Organisation alle symmetrischen Schlüssel aufbewahrt. Möchte nun Person A mit Person B kommunizieren, so bittet sie das Trustcenter um einen Sitzungsschlüssel. Das Trustcenter generiert einen Schlüssel und sendet ihn zwei mal an Person A. Ein mal symmetrisch verschlüsselt mit dem Schlüssel von A, und das andere mal symmetrisch verschlüsselt mit dem Schlüssel von B. A entschlüsselt das erste Paket und leitet das zweite an B weiter. Nun haben beide denselben Schlüssel sicher übermittelt bekommen und können sicher zwischen einander kommunizieren.

Allerdings ist das Trustcenter der *single point of failure*: Ist es nicht erreichbar, so können auch die anderen Teilnehmer nicht sicher miteinander kommunizieren, auch wenn die Verbindung steht. Zum anderen speichert es alle Schlüssel; wird es also kompromittiert, ist die Kommunikation der gesamten Organisation unsicher.

7.3.2 Man in the Middle Angriff

Auch bei asymmetrischen Verfahren muss beim Schlüsseltausch aufgepasst werden. Die Übermittlung des öffentlichen Schlüssels von A zu B kann von einer dritten Person M in der Mitte abgefangen werden. Diese sendet stattdessen ihren eigenen public key an B weiter; Entsprechend geht es auch von B zu A. Nun denken beide, sie hätten jeweils den Public Key ihres Gesprächspartners, obwohl sie den von M haben. Somit kann M die ausgetauschten Nachrichten mit seinem Schlüssel entschlüsseln, mitlesen, und dann den Klartext in den entsprechend richtigen Schlüssel an die eigentlichen Empfänger weiterleiten. Es muss beim Schlüsselaustausch also auch stets die *Authentizität* der Public Keys überprüft werden.

7.3.3 Das Interlock-Protokoll

Das Interlock-Protokoll möchte diesen Man in the Middle Angriff unmöglich machen. Zwar können die Public-Keys beim anfänglichen Schlüsseltausch noch ersetzt werden, ab da werden alle versandten Nachrichten allerdings in zwei Hälften geteilt. A verschlüsselt die Nachricht und sendet die Hälfte des verschlüsselten Blocks an B. M kann mit nur der Hälfte eines Blocks allerdings nicht entschlüsseln und den Klartext dann neu verschlüsseln. Die Zweite Hälfte sendet A allerdings erst, wenn sie von B ebenfalls einen halben Chiffreblock erhält. Somit kann sich keine Person mehr so einfach dazwischen setzen.

Dieses Verfahren ist in der Praxis aber sehr aufwändig und wird deshalb selten eingesetzt.

7.4 Der Diffie-Hellmann-Algorithmus

Oft wird heutzutage mit einer hybriden Verschlüsselung gearbeitet; Per asymmetrischem Verfahren wird ein Schlüssel übermittelt, mit dem in Zukunft symmetrsich verschlüsselt kommuniziert wird.

Hierbei möchte man allerdings auch eine sogenannte *Perfect Forward Secrecy* erreichen. RSA ist aktuell sicher, weil angenommen wird, dass die Faktorisierung zu aufwändig ist. Ist dies irgendwann nicht mehr der Fall (oder es gibt andere Gründe, warum die ursprüngliche Nachricht geknackt wurde), so kann eine Nachricht, die früher aufgezeichnet wurde (evtl. auch schon vor Jahren) jetzt entschlüsselt werden, um den symmetrischen Schlüssel zu erhalten . Mit diesem kann dann jegliche Kommunikation von damals bis heute entschlüsselt werden. Um diese Situation zu verhindern gibt es den Diffie-Hellmann-Algorithmus. Dieser funktioniert wie folgt:

1. A und B einigen sich auf eine große Primzahl n und eine Zahl g. Beides darf öffentlich bekannt sein.
2. A wählt eine große Zufallszahl x und sendet $X = g^x \mod n$ an B. Der Öffentliche Schlüssel von A ist (X, g, n) und der geheime Schlüssel ist (x, g, n)
3. B wählt eine große Zufallszahl y und sendet $Y = g^y \mod n$ an B. Der Öffentliche Schlüssel von B ist (Y, g, n) und der geheime Schlüssel ist (y, g, n)
4. A berechnet den geheimen Sitzungsschlüssel $k = Y^x \mod n$
5. B berechnet den geheimen Sitzungsschlüssel $k' = X^y \mod n$

Es gilt $k = k'$, da

$$k = Y^x \mod n = g^{xy} \mod n = X^y \mod n = k'$$

Die Sicherheit dieses Algorithmus liegt hierbei darin, dass aus X und Y nicht berechnet werden kann, was x und y ist. Löst man z.B. die Gleichung $Y = g^y \mod n$ nach y auf, so erhält man $y = \frac{\log(Y)}{\log(g)}$. Hier kommt das Problem des *diskreten Logarithmus* zu tragen. Während im normalen Zahlenraum der Logarithmus dem Gesetz der Ähnlichkeit (kleine Änderungen in x führen zu kleinen Änderungen in y) folgt, und man daher daraus schließen kann, dass der wert für $\ln(2,1)$ zwischen $\ln(2,0)$ und $\ln(2,2)$ liegt, so gilt dies in einem beschränkten Zahlenraum nicht, da aufgrund der Modulorechnung komplett unterschiedliche Ergebnisse herauskommen können. Somit kann man den Logarithmus von g im Zahlenraum von n nur lösen, indem man eine komplette Multiplikationstabelle aufstellt und nachschaut. Bei hinreichend großen Zahlen wird dies sehr aufwändig.

Wichtig ist hierbei die Wahl von n und g, da sich sonst im Zahlenraum die Ergebnisse der Potenzen schnell wiederholen (z.B. ist 2^x im Zahlenraum 7 für $x = 1$ 2, und für jedes höhere x ist das Ergebnis der Potenz abwechselnd 4 oder 1).

Auch hier muss aber auf die Authentizität der Schlüssel geachtet werden, da ein Man in the Middle (MITM) Angriff nicht ausgeschlossen ist.

7.5 El Gamal Algorithmus

Unter Verwendung des Diffie-Hellmann-Algorithmus wird der Schlüssel aus den zufällig gewählten Zahlen errechnet. Möchte man hingegen den Schlüssel k frei wählen, so ist dies durch den El Gamal Algorithmus möglich. Das Generieren der öffentlichen und privaten Schlüssel funktioniert genau gleich:

1. A und B einigen sich auf eine große Primzahl n und eine Zahl g. Beides darf öffentlich bekannt sein.
2. A wählt eine große Zufallszahl x und sendet $X = g^x \mod n$ an B. Der öffentliche Schlüssel von A ist (X, g, n) und der geheime Schlüssel ist (x, g, n)
3. B wählt eine große Zufallszahl y und sendet $Y = g^y \mod n$ an B. Der öffentliche Schlüssel von B ist (Y, g, n) und der geheime Schlüssel ist (y, g, n)

Nun kann $k \in \{1, \dots n-1\}$ frei gewählt werden. Zudem wird eine Zufallszahl $r \in \{1, \dots n-1\}$ erzeugt. A übermittelt dann die verdeckte Zufallszahl $m_1 = g^r \mod n$ und den verdeckten Schlüssel $m_2 = k \cdot Y^r \mod n$ an B. B berechnet schließlich den Schlüssel mittels

$$\begin{aligned} k &= m_2 \cdot (m_1^y)^{-1} \mod n \\ &= k \cdot Y^r \cdot ((g^r)^y)^{-1} \mod n \\ &= k \cdot g^{y^r} \cdot (g^{r^y})^{-1} \mod n \\ &= k \cdot g^{ry} \cdot (g^{ry})^{-1} \mod n \\ &= k \cdot 1 \mod n \end{aligned}$$

Auch hier muss aber auf die Authentizität der Schlüssel geachtet werden, da ein Man in the Middle (MITM) Angriff nicht ausgeschlossen ist.

7.6 Elliptische Kurven

Eine elliptische Kurve ist eine Kurve, die typischerweise der Formel

$$y^2 = x^3 + ax + b \tag{7.1}$$

folgt. Durch die quadratische Komponente in y ist es streng genommen keine Funktion, da sie für einen x-Wert 2 y-Werte besitzt. Diese werden an der x-Achse gespiegelt.

So eine Funktion kann in einem Körper (z.B. $\mathbb{Z}_{13}$, oder aber einen Galois-Körper (siehe Punkt 10.6)) berechnet werden, um einzelne Punkte auf der Ebene auszumachen.

7.6.1 Addition

Alle Punkte lassen sich innerhalb des Modul-Körpers addieren, das Addieren ist allerdings etwas komplizierter. Graphisch gesehen wird zwischen zwei Punkten eine Gerade gezeichnet und der Punkt gesucht, an dem diese die Kurve ein drittes Mal schneidet. Dann wird von diesem Punkt das *konjugiert Komplexe* berechnet (= der y-Wert wird mit -1 multipliziert). Der so entstandene Punkt ist das Ergebnis der Addition. Sind die beiden addierten Punkte dieselben, oder das konjugiert Komplexe voneinander, so schneidet die Gerade die Kurve kein drittes Mal, sodass der Punkt anders ermittelt werden muss.

Rechnerisch sieht das alles etwas komplizierter aus. Zuerst müssen wir ein r berechnen; Sind die beiden Summanden p_1 und p_2 nicht gleich und nicht das konjugiert Komplexe voneinander, so lautet die Formel:

$$r = \frac{y_2 - y_1}{x_2 - x_1}$$

Sind die Summanden hingegen gleich oder das konjugiert Komplexe, so gilt

$$r = \frac{3x_1^2 + a}{2y_1}$$

wobei a das a aus der Kurvengleichung (7.1) ist.

Schließlich werden die Koordinaten des Ergebnispunktes p_3 berechnet:

$$x_3 = r^2 - x_1 - x_2$$

und

$$y_3 = r(x_1 - x_3) - y_1$$

Bei allen Berechnungen ist zu beachten, dass die Operationen ebenfalls innerhalb des Körpers durchgeführt werden müssen (z.B. teilen wir nicht, sondern multiplizieren mit dem multiplikativen Inversen).

Zusätzlich zu den normalen Punkten benötigt es noch einen abstrakten „Punkt im Unendlichen“ als neutrales Element bezüglich der Addition.

7.6.2 Erzeugender Kurvenpunkt

Da es sich um einen Modulkörper handelt, und dieser zwangsweise zyklisch ist, lässt sich die komplette Kurve durch einen Punkt P erzeugen, der so lange auf sich selbst addiert wird, bis wir wieder beim ursprünglichen Punkt angelangt sind. Auf dem Weg erzeugen wir jeden möglichen Punkt in diesem Körper. Man spricht auch von einer *Zyklischen Gruppe mit erzeugendem Element P*

7.6.3 Multiplikation

Die Multiplikation von Punkten mit einem skalaren Faktor s ist definiert durch das s-fache Addieren des Punktes P auf sich selbst:

$$s \cdot P = \underbrace{P + \cdots + P}_{s \text{ Mal}}$$

7.6.4 Elliptische Kurven und Kryptographie

Da das Multiplizieren so einfach definiert ist, das Faktorisieren jedoch nicht in polynomieller Laufzeit lösbar ist, eignen sich Funktionen auf elliptischen Kurven auch gut für kryptographische Verfahren.

Bei allen Verschlüsselungsalgorithmen, die durch das Problem des diskreten Logarithmus geschützt sind (also $a^b \mod p$) lässt sich die Potenzierung $\mod p$ durch eine Multiplikation von Punkten auf einer elliptischen Kurve ersetzen.

Durch die komplexe Berechnung und effiziente Speicherung lässt sich die Schlüssellänge bei elliptischen Kurven auf selbem Sicherheitsniveau stark verringern. Aus diesem Grund werden elliptische Versionen von Verschlüsselungsalgorithmen oft mit Chipkarten verwendet, auf denen der Speicherplatz immer knapp ist.

8 Authentifikation und digitale Signatur

8.1 Einwegfunktionen und Einweg-Hashfunktionen

Definition 8.1: Einwegfunktion

Eine Einwegfunktion ist eine Funktion f, die sich leicht berechnen lässt, deren Umkehrfunktion f^{-1} sich jedoch nicht oder nur sehr schwer berechnen lässt. f sollte öffentlich bekannt sein.

Definition 8.2: Einweg-Hashfunktion

Eine Einweg-Hashfunktion ist eine Einwegfunktion, die beliebig lange Klartexte auf einen Hashwert fester Länge abbildet.

Da eine Einweg-Hashfunktion nach obiger Definition eine unendlich große Menge auf eine endliche Menge abbildet, muss es zu Kollisionen kommen und somit ist die Funktion nicht injektiv. Bei einer guten Einweg-Hashfunktion muss es also sehr schwierig sein, so eine Kollision hervorzurufen.

Einige wichtige Hash-Funtionen sind (Message Digest) MD 4 und MD5 mit je 128 Bit langen Hashwerten, sowie der RIPE-MD und der Secure Hash Algorithmus SHA-1 mit 160-Bit Hashes, sowie dessen Nachfolger SHA-256 mit entsprechend 256 Bits.

8.1.1 Passwortverschlüsselung

Hashfunktionen werden unter anderem zum sicheren Speichern von Passwörtern verwendet. Passwörter sollten nie im Klartext gespeichert werden, daher sollte eine Verschlüsselung verwendet werden. Wird ein Passwort z.B. von einem Server verschlüsselt und gespeichert, so kann der entsprechende Serveradministrator den verwendeten Schlüssel allerdings missbrauchen, um die Passwörter wieder zu entschlüsseln. Dies soll ebenfalls verhindert werden. Aus diesem Grund werden oft nur die Hash-Werte von Passwörtern gespeichert, sodass kein Rückschließen auf das Originalpasswort möglich ist. Beim Einloggen wird das eingegebene Passwort mit demselben Algorithmus gehasht und das Ergebnis mit dem Datenbankeintrag verglichen: Sind sie gleich, so wurde das korrekte Passwort eingegeben.

Sollten die gespeicherten Passwörter nun gestohlen werden, so hat man das Passwort zwar nicht im Klartext, allerdings können mit der Hashfunktion verschiedene Möglichkeiten ausprobiert und verglichen werden; Ist es ein häufig verwendetes Passwort, so wird ein Angreifer schnell denselben Hashwert produzieren können und hat somit das Passwort des Nutzers. Aus diesem Grund werden Pasworthashes oft „gesalzen“. Hierbei wird eine vorbestimmte Bitfolge an das Passwort angehängt ud dann erst gehasht. Allein bei einem *Salt* von 8 Bit gibt es $2^8 = 256$ mögliche Bitfolgen. Möchte man also in einem Angriff ein Passwort vergleichen, so muss man alle möglichen Kombinationen aus Passwort und Hashes ausprobieren, also 256 anstatt 1.

8.1.2 Der Geburtstagsangriff

Ein weiteres Anwendungsgebiet von Hashfunktionen sind digitale Signaturen. Hier wird der Hashwert einer Nachricht berechnet und dieser mit dem private key eines asymmetrischen Verschlüsselungsverfahrens signiert. Der Empfänger erhält Nachricht und Signatur und kann den Hashwert der Nachricht nachrechnen und den Hashwert der Signatur entschlüsseln, indem er sie mit dem öffentlichen Schlüssel potenziert. Stimmen die Werte überein, so ist alles in Ordnung. Wenn nicht, so wurde die Nachricht modifiziert.

Ein naiver Angriff wäre es, so lange eine andere Nachricht zu konstruieren und abzuändern, bis derselbe Hashwert erzeugt wird. Diese Nachrichten können nun vertauscht werden, ohne dass die Signatur einen Hinweis darauf gibt, dass der Inhalt verändert wurde. Dies gleicht jedoch einem BruteForce-Angriff, da bei

guten Hashfunktionen kein Gesetz der Nähe gilt und das Ergebnis scheinbar zufällig ist – somit kann man nichts anderes tun als auszuprobieren.

Eine bessere Methode ist es, zufällige Texte zu erstellen und darauf zu hoffen, dass zwei davon denselben Hashwert besitzen. Mit noch mehr Glück enthält einer der beiden Texte eine Anweisung, die für den Angreifer vorteilhaft ist. Nun übergibt er den harmlosen Text dem Opfer zur Signatur, und ersetzt ihn dann durch den „böswilligen“ Text, bevor die Nachricht übermittelt wird.

Warum genau dieser Ansatz effizienter ist, kann gut mit dem sogenannten Geburtstagsparadoxon erklärt werden. Möchte man z.B. wissen, wie viele Leute man zu einer Feier einladen muss, damit mit einer Wahrscheinlichkeit von mindestens 50% mindestens eine Person an diesem Tag x Geburtstag hat so muss man für jede Person g_i von 1 bis n die Wahrscheinlichkeiten aufaddieren, dass entweder g_1 an diesem Tag Geburtstag hat, oder g_2, bis hin zu g_n. Dies lässt sich auch über das Gegenereignis ausdrücken, also 1− die Wahrscheinlichkeit, dass weder g_1, noch g_2 noch ..., noch g_n heute Geburtstag haben. Somit ergibt sich also

$$P(g_1 = x \vee g_2 = x \vee \cdots \vee g_n = x) = 1 - P(g_1 \neq x \wedge g_2 \neq x \wedge \cdots \wedge g_n \neq x)$$

Und diese Wahrscheinlichkeit soll größer sein als 50%; Also:

$$\begin{aligned} 1 - \left(\frac{364}{365}\right)^n &> 0,5 && | -1 \\ -\left(\frac{364}{365}\right)^n &> -0,5 && | \cdot (-1) \\ \left(\frac{364}{365}\right)^n &< 0,5 \end{aligned}$$

Diese Gleichung lässt sich nach n auflösen:

$$\begin{aligned} \left(\frac{364}{365}\right)^n &< 0,5 && | \ln \\ n \cdot \ln\left(\frac{364}{365}\right) &< \ln(0,5) && | : \ln\left(\frac{364}{365}\right) \\ n &> \frac{\ln(0,5)}{\ln\left(\frac{364}{365}\right)} \end{aligned}$$

Somit ergibt sich ca. $n \geq 253$ Personen, bis eine wahrscheinlich an diesem Tag Geburtstag hat.

Möchte man stattdessen wissen, wie viele Menschen k anwesend sein müssen, damit zufällig zwei am selben Tag Geburtstag haben, so müsste man berechnen $P(g_1 = g : 2 \vee g_1 = g_3 \vee \cdots \vee g_{k-1} = g_k)$, oder eben die Gegenwahrscheinlichkeit, dass niemand am selben Tag Geburtstag hat: $1 - P(g_1 \neq g_2 \wedge g_1 \neq g_3 \wedge \cdots \wedge g_{k-1} \neq g_k)$. Die Anzahl an Vergleichen die wir durchführen müssen, entspricht der Anzahl an Paaren innerhalb einer Menge; Die Formel um das zu Berechnen kennen wir bereits aus dem Friedmann-Test aus Punkt 3.2.1.2: $\frac{k(k-1)}{2}$. Somit ergibt sich die Formel

$$1 - \left(\frac{364}{365}\right)^{\frac{k(k-1)}{2}} > 0,5$$

Diese erinnert sehr stark an unsere vorherige Gleichung, welche wir bereits gelöst haben. Aus diesem Grund setzten wir nun unsere Exponenten gleich, um uns das erneute Auflösen der Formel zu sparen:

$$\frac{k(k-1)}{2} = n \Leftrightarrow k = \frac{1}{2}\sqrt{\frac{1}{4} + 2n}$$

Für unser Beispiel ergibt sich 22, 98, und somit gilt, dass $k \geq 23$ Leute eingeladen werden müssen, damit mindestens zwei Menschen am selben Tag Geburtstag haben. Für große n werden die konstanten Faktoren unwichtig, so dass näherungsweise gilt $k = \sqrt{2n}$.

Überträgt man diese Berechnung nun wieder auf unser Beispiel mit den Hashwerten, die aus m Bit bestehen, die jeweils 2 Werte annehmen können, so ergibt sich für einen vorgegebenen Hashwert eine Testanzahl von 2^{m-1}. Bei einem Geburtstagsangriff ergeben sich jedoch nur $k = \sqrt{2 \cdot 2^{m-1}} = \sqrt{2^m} = 2^{\frac{m}{2}}$. Um gegen Geburtstagsangriffe sicher zu sein, muss der Hashwert also doppelt so lang sein wie für das Berechnen eines vorgegebenen Wertes benötigt wird.

8.2 Zero Knowledge Protokolle

Eine Schwachstelle bildet weiterhin der Weg des Passworts zwischen Eingabe und Verschlüsselung, sobald also etwas im Klartext übertragen wird. Um dies zu verhindern gibt es sogenannte *Zero Knowledge Protokolle*, bei denen keinerlei sensible Information abhörbar übertragen werden soll.

8.2.1 Challenge and Response

Die Idee des *Challenge and Response* beginnt zuerst mit einem *Prädialog*, der für die eigentliche Vorgehensweise benötigt wird. Person A gibt über den unsicheren Kanal eine Benutzerkennung ein. Partei B, meist ein Rechenzentrum oder eine Organisation mit mehreren Benutzern, such für sich anhand der eingegebenen Benutzerkennung den passenden Schlüssel k von A heraus.

Nun beginnt die eigentliche Challenge: B generiert eine Zufallszahl r und übermittelt sie an A. A berechnet dann einen Wert R' mit einer Funktion $f(k', r)$, wobei k' der Schlüssel ist, der sich in Besitz von A befindet. Die Response $R' = f(k', r)$ wird dann wieder an B übermittelt, wo überprüft wird, ob $R' = f(k, r)$. Ist das korrekt, ist A authentifiziert, ohne dass das identifizierende Element preisgegeben wurde.

Hier handelt es sich allerdings nicht um ein reines Zero-Knowledge-Protokoll, da B zu irgend einem früheren Zeitpunkt den geheimen Schlüssel von A erhalten muss.

8.2.2 Das Fiat-Shamir-Protokoll

Ein Zero Knowledge Protokoll ist das Fiat-Shamir-Protokoll. Hier hat die Person P, die beweisen (prove) will, dass sie ein Geheimnis kennt, einen geheimen Schlüssel s und einen öffentlichen Schlüssel $s^2 \mod n$ wobei $n = p \cdot q$. Nun generiert P eine Zufallszahl r und bildet daraus zwei Nachrichtenteile (alle berechnet $\mod n$): $x = r^2$ und $y = r \cdot s$. Diese beiden Nachrichten werden an V geschickt, der verifizieren muss, dass P das Geheimnis kennt. V berechnet, ob $y^2 = (r \cdot s)^2 = r^2 \cdot s^2$ dasselbe ist wie $v \cdot x = s^2 \cdot r^2$. Stimmen die beiden Werte überein, so muss P die Kenntnis über s gehabt haben.

In dieser Form ist das Protokoll allerdings angreifbar. Da wir wissen, dass $y^2 = x \cdot v$, und v als öffentlicher Schlüssel bekannt ist, können wir y auf einen beliebigen Wert festlegen und die Gleichung dann nach x auflösen. Somit müssen wir nur eine Übertragung abhören und können uns dann ein Zahlenpaar konstruieren, das diesen Test besteht.

Um diese Schwachstelle zu beheben, kann das Programm abgewandelt werden: Zuerst schickt P das bekannte $x = r^2$. Nun fordert V über ein zurückgesendetes Bit (je nachdem ob es 0 oder 1 ist) zufällig entweder $y = r \cdot s \mod n$ an, was bisher die Schwachstelle des Algorithmus ist, oder es fordert $y = r \mod n$ an. Haben wir nun unser x konstruiert und nicht selbst erzeugt, so können wir diese Anforderung nicht erfüllen. So können wir uns nur mit einer Wahrscheinlichkeit von 50% verifizieren: entweder wir konstruieren x und müssen $r \cdot s \mod n$ liefern, oder wir berechnen $x = r^2$ und hoffen, dass wir nur $r \mod n$ liefern müssen. Dieser Test kann nun k-Mal wiederholt werden. Die Wahrscheinlichkeit, dass wir uns jedes Mal per Zufall durchmogeln beträgt hierbei $\frac{1}{2^k}$. Je öfter V also nachfragt, desto sicherer kennt P auch wirklich s.

Auf diese Weise ist das Protokoll bei genügend Wiederholungen sicher. Der Knackpunkt hierbei ist, dass wir $r = \sqrt{x}$ in einem Modularen Zahlenraum nicht bzw. nur mit sehr viel Aufwand berechnen können und somit auf die oben beschriebenen Methoden zurückgreifen müssen, die im modifizierten Algorithmus aber abgefangen werden.

8.3 Digitale Signaturen

Eine gute Signatur sollte folgende Kriterien erfüllen:

- Sie ist authentisch
- Sie ist fälschungssicher
- Sie ist nicht wiederverwendbar
- Sie ist unveränderbar
- Sie ist bindend

Eine Signatur mit Tinte auf Papier erfüllt kaum eine dieser Anforderungen, eine digitale Signatur jedoch alle; Warum also hat sie sich noch nicht durchgesetzt?

Hier gibt es das Problem der Unmittelbarkeit. Während ich einer Person beim Unterschreiben zuschauen kann und dann die Unterschrift mit vorherigen Signaturen selbst vergleichen kann, bin ich bei digitalen Signaturen darauf angewiesen, dass eine Maschine für mich signiert und eine andere Maschine diese Signatur verifiziert. Hier sind wir also auf Mittelwege angewiesen, die manipuliert werden könnten.

Digital signiert werden kann eine Nachricht recht einfach mit einem Public Key Algorithmus. Bei einer kurzen Nachricht, kann diese mit dem privaten Schlüssel verschlüsselt und mit dem passenden öffentlichen Schlüssel entschlüsselt werden. Ist die Nachricht länger, so kommt eine öffentlich bekannte Einweg-Hashfunktion ins Spiel. Diese Funktion muss sicher sein; Also muss es schwer bis unmöglich sein der Originalwert zu berechnen und es muss sehr schwer sein eine Kollision herbeizuführen. Die Nachricht wird gehasht und der Hash wird verschlüsselt. Schließlich wird der Hash wieder entschlüsselt und mit dem neu berechneten Hash der erhaltenen Nachricht verglichen.

8.3.1 Digital Signature Algorithm

Der Digital Signature Algorithm ist ein von der NSA entwickelter Algorithmus, der speziell zum Signieren von Nachrichten verwendet wird; Versucht man damit etwas zu verschlüsseln, so ist er nicht geeignet. Das löste eine große Diskussion auf, wie genau dieser Algorithmus denn funktioniert, wenn man damit sicher signieren, aber nicht verschlüsseln könne. Schließlich wurde der DSA zum DSS weiterentwickelt. Dieser bietet auch Digital Signature with Message Recovery an, bei dem die Signatur einen Teil der Nachricht überdeckt und der Empfänger somit gezwungen wird die Signatur zu überprüfen, bevor er die Nachricht lesen kann.

8.3.2 Blinde Signaturen

Eine Blinde Signatur ist eine Signatur, bei der die Signierende Person den Inhalt der zu signierenden Nachricht nicht kennt. Doch wie funktioniert das elektronisch?

Zuerst wählt Person A eine Zufallszahl k, merkt sich diese und potenziert sie mit dem öffentlichen Schlüssel e von Person b. Dann multipliziert sie das Ergebnis mit der eigentlichen Nachricht m und übermittelt $t = M \cdot k^e \mod n$ an B. B signiert t und sendet $u = t^d \mod n$ zurück.

A kann nun die signierte Originalnachricht M^d wie folgt berechnen:

$$\begin{aligned} M^d &= \frac{u}{k} \mod n \\ &= \frac{t^d}{k} \mod n \\ &= \frac{(M \cdot k^e)^d}{k} \mod n \\ &= \frac{M^d \cdot k^{ed}}{k} \mod n \\ &= \frac{M^d \cdot k}{k} \mod n \\ &= M^d \mod n \end{aligned}$$

Wie hier auffällt, wird durch k geteilt, also eigentlich mit dem multiplikativen Inversen von k multipliziert. Solch ein Inverses existiert allerdings nur, wenn k und n teilerfremd sind. n besteht allerdings aus $n = p \cdot q$. Ist also ein Mal nicht möglich die Inverse zu berechnen, so ist k entweder p, q oder ein vielfaches davon. Mit dieser Information kann ein Angreifer dann auf alle anderen Werte schließen und so schließlich den geheimen Schlüssel d berechnen.

8.4 Digitale Signatur in der Praxis

Obwohl die digitale Signatur sicher und einfach anzuwenden ist, hat sich sich aktuell noch nicht durchgesetzt, woran liegt das?

8.4.1 Speichern des geheimen Schlüssels

Der geheime Schlüssel zum Signieren muss natürlich irgendwo gespeichert werden. Dies passiert meist auf irgend einem Speichermedium, auf das theoretisch andere Menschen zugreifen könnten. Also muss der Schlüssel selbst auch nochmal verschlüsselt werden. Hierfür sollte natürlich ein Verfahren verwendet werden, das so sicher ist wie das Verfahren des Schlüssels, den man sichern möchte – den eine Kette is so stark wie ihr schwächstes Glied; Warum den 1204 Bit RSA knacken, wenn der Schlüssel mit einem 64 Bit DES verschlüsselt wurde?

Ein starkes symmetrisches Verfahren (z.B. AES) mit 128 Bit ist in ungefähr so stark wie RSA mit 1024 Bit. Um einen 128 Bit langen Schlüssel zu erhalten, müsste das Passwort zum Verschlüsseln ca. 21 Zeichen lang sein. Damit das Passwort allerdings sicher ist, und nicht leicht zu erraten ist, müsste es echt zufällig generiert werden. Und sich eine 21 Zeichen lange zufällige Zeichenfolge zu merken ohne sie irgendwo aufzuschreiben, schafft kaum jemand. Um dies zu lösen gibt es die Merkhilfe der *Passphrase*: Man merkt sich einen Satz und nimmt von jedem Wort (und auch enthaltenen Zahlen) das erste Zeichen: Schon hat man ein scheinbar zufälliges Passwort, das man sich auch merken kann. Aufgrund von Eigenheiten der deutschen Sprache sollte der Merksatz jedoch mindestens 80 Zeichen lang sein.

Zudem kann man sich bei einem so langen Passwort immer wieder vertippen. Hier helfen jedoch moderne Speichermedien, wie z.B. eine Chipkarte, die das Passwort sicher speichert, die Nachricht entgegen nimmt, sie auf dem Chip signiert und die signierte Nachricht wieder ausgibt – aber nur, wenn der Chip vorher durch weitere Sicherung (PIN, Fingerabdruck etc.) entsperrt wird.

8.4.2 Vertrauen in die Software

Zudem muss man sich, wie bereits in Punkt 8.3 beschrieben, auf die Geräte verlassen, die die Signatur ausführen oder verifizieren. erden sie manipuliert bemerken wir das im Alltag meist nicht.

Hier kann aber die Signatur des Herstellers, oder noch besser, die Signatur einer unabhängigen Prüfstelle vor jedem Programmstart überprüft werden.

8.5 Das Signaturgesetz

Das Signaturgesetz setzt digitale Signaturen handschriftlichen Signaturen (inkl. Rechtsgültigkeit) gleich und unterscheidet in drei verschiedene Klassen digitaler Signaturen:

Klasse 1: Jegliche Daten, die irgendwie mit dem Inhalt zusammenhängen und diesen Authentifizieren sollen (z.B. ein „viele Grüße, Malte Jakob“ in Textform, oder eine eingescannte Unterschrift)

Klasse 2: Sind Signaturen nach z.B. dem PGP-Standard mit ausreichend geschütztem Schlüssel.

Klasse 3: Sind Signaturen mit einem Schlüssel der auf einer Chipkarte gespeichert, und der von einem zertifizierten Trustcenter zertifiziert und verwaltet werden muss.

8.6 Biometrische Verfahren

Bei vielen Authentifikationsmethoden (z.B. Besitz oder Wissen) kann der Schlüssel gestohlen und missbraucht werden. Biometrische Daten (wie z.B. Fingerabdruck oder Netzhaut) hingegen sind nur mit großem Aufwand vom Besitzer trennbar.

Hierbei gibt es verschiedene Merkmale, die überprüft werden können:

Gesicht: Die Form des Gesichtes

Iris: Das „farbige“ im Auge

Netzhaut: Der innere Aufbau des Auges (ca. 1:1.000.000.000, zwei selbe Netzhäute zu finden)

Handgeometrie: Aufbau der Hand (Fingerlänge, Handfurchen, etc.)

Fingerabdruck: Die „Rillen“ auf Fingerkuppen (ca. 1:1.000.000, zwei selbe Abdrücke zu finden). Hier wird nicht der komplette Abdruck gespeichert, sondern nur markante Punkte , wie z.B. das Ende einer Rille, oder das Zusammenlaufen von zwei Rillen.

Manche solche Merkmale lassen sich allerdings leicht kopieren. So können Fingerabdrücke abgenommen werden, und durch Fotos lassen sich Iren und Abdrücke rekonstruieren. Aus diesem Grund gibt es auch verhaltensbasierte Verfahren, wie z.B.

Schreibrithmus: Wie schreibt eine Person (z.B. Pausen zwischen Buchstaben oder Wörtern)?

Stimme: Die Stimme identifiziert eine Person auch recht eindeutig

Dynamisch Unterschrift: Wie unterschreibt eine Person (welche Pausen, welche Reihenfolge etc.)?

Diese Methoden zu fälschen ist zwar auch möglich, allerdings etwas aufwändig.

In allen Methoden werden die Daten zuerst erfasst und analysiert, um diese dann aufzubereiten und mit den Referenzdaten zu vergleichen. Je nach Ähnlichkeit und Schwellwert ist die Person daraufhin autorisiert, oder eben nicht.

Die biometrischen Werte sind jedoch nicht immer konstant (z.B. variiert die Körpergröße je nach Tag), sodass gewisse Abweichungen toleriert werden müssen. Die Wahl dieses Schwellwertes ist hierbei sehr wichtig. Ist er zu klein, so wird eine berechtigte Person zu unrecht abgewiesen (False Rejection Rate (FRR)), anstatt wie erwartet berechtigter weise zugelassen zu werden (True Acceptance Rate (TAR)). Wird die Schwelle jedoch zu groß gewählt, so werden Personen, die normalerweise berechtigterweise abgewiesen werden (True Rejection Rate (TRR)), stattdessen fälschlicherweise Zugelassen (False acceptance Rate (FAR)).

Der Punkt, an dem die FRR und die FAR gleich sind, wird auch Gleichfehlerrate genannt. Das Verhältnis von FAR und FRR lässt sich auch als $f(\Theta) = \frac{1}{k \cdot \Theta}$ mit $\Theta \in (0, \infty)$ darstellen. Die Gleichfehlerrate entspricht hierbei der ersten Winkelhalbierenden Θ. Je genauer bzw. strenger das Verfahren, desto größer der Faktor k und desto enger liegt die Funktion an den Achsen.

Damit ein Merkmal sich für biometrische Authentifikation eignet, müssen folgende Bedingungen gegeben sein:

- Es muss von der Methode, Dauer und Kosten gut messbar sein
- Es muss eindeutig zuordenbar sein
- Es muss schwer fälschbar sein
- Es muss effizient speicherbar sein
- Es muss über die Zeit konsistent sein
- Es muss vom Benutzer akzeptiert werden

9 Blockchain

Wer digital Geld senden möchte, muss dies über ein Bankkonto tun. Die Blockchain ist eine Lösung, die dezentral ohne Banken funktioniert. Um zu verstehen wie sie funktioniert, eine Analogie:

Verschiedene Menschen sind auf einer Insel gestrandet; Jeder versucht auf sich selbst gestellt zu überleben – sollte eine Person nicht von selbst zurechtkommen, kann sie Dinge gegen andere Dinge eintauschen. Aber was, wenn Person B nichts von dem will, was Person A anzubieten hat? Deswegen einigen sich die Inselbewohner auf eine alternative Währung: Muscheln. Jeder bekommt zu Beginn eine feste Anzahl an Muscheln und kann diese Handeln wie er/sie möchte. Nach einiger Zeit, gehen die Muscheln allerdings kaputt; Um das System allerdings beizubehalten, schreibt sich jeder auf, wer gerade wie viel Geld hat. Wann immer eine „Muschel“ Besitzer wechselt, wird dies auf der gesamten Insel angekündigt, damit es alle mitbekommen und aufschreiben können. Natürlich kann nicht jede Person einfach behaupten „A hat B 5 Muscheln gegeben“, das muss immer von der zahlenden Person bestätigt werden. Hat mal eine Person oder mehrere Personen doch nicht mitbekommen, dass Muscheln die Besitzer gewechselt haben, so gilt der Stand, den die Mehrheit der Inselbewohner für richtig hält.

Und wie funktioniert das alles in der Informatik? Mit der sogenannten *Blockchain*. Alle Beteiligten haben diese „Kette“. Diese besteht aus Datenblöcken, die hauptsächlich aus zwei Teilen bestehen: Der Payload, die die tatsächlichen Daten enthält (z.B. A gibt B 5 Muscheln), und dem Header, der diese Daten validiert und die Kette bildet. Jede gültige Transaktion ist in dieser Kette chronologisch gespeichert.

Wie bereits erwähnt, enthält die Payload die Daten für die Transaktion. Wer gibt wem wie viel Geld? Das muss natürlich auch alles abgesichert werden. Zum einen: Hat diese Person überhaupt genügend Muscheln, um zu bezahlen? Hierfür wird die komplette Kette von Anfang an bis jetzt durchsucht und jede eingehende (= Person bekommt Muscheln) und ausgehende (=Person bezahlt mit Muscheln) Transaktion aufsummiert, um das aktuelle Guthaben zu ermitteln. Hat die Person mehr oder gleich viele Muscheln wie überwiesen werden sollen, wird zum nächsten Schritt übergegangen. Das „A gibt B“ wird über Public keys, bzw. deren Hashes, geregelt. In Wirklichkeit steht dort also „Hash(Public-Key von A) gibt Hash(Public-Key von B)“. Die Hashfunktion wird verwendet, um Platz zu sparen und auch bei einer neuen Schlüssellänge dieselbe Adresslänge beizubehalten. Nun wissen wir also, wie wir überprüfen, ob die Person zahlen kann, und wie wir feststellen, wer an der Transaktion beteiligt ist. Jetzt müssen wir noch beweisen, dass Person A diese Transaktion auch tatsächlich beauftragt hat, und nicht jemand anderes das einfach behauptet. Dies geschieht, indem Person A beide Adresswerte einfach mit dem privaten Schlüssel signiert.

Nun haben wir also einen Datensatz validiert. In einem Payload-Block stehen immer mehrere Datensätze. Doch wer hindert uns daran, in einem bereits validierten Payload-Block einfach einen Datensatz auszutauschen, sodass die Leute glauben, wir hätten ein Mal mehr Geld bekommen, alls tatsächlich wahr ist? Hier kommt der Header-Teil des Blocks ins Spiel – für die Payload wird der Merkle-Tree-Root-Hash berechnet. Um diesen zu berechnen wird für jeden Datensatz d_i der Hash h_i berechnet. Dann werden für je zwei Hashwerte ein gemeinsamer Hash berechnet ($h_{1;2} = h(h_1, h_2), h_{3;4} = h(h_3, h_4) \dots$), dies wird so lange weiter nach oben betrieben, bis es nur noch einen Hashwert gibt (man kann es sich auch wie ein Wettkampfplan in einem Sportturnier vorstellen). So kann ich innerhalb eines Blockes die Daten nicht mehr austauschen, aber ich kann immer noch komplette Blöcke ersetzen. Um dies zu verhindern, hat jeder Header auch den Hash des Headers des vorherigen Blocks abgespeichert. Dieser Hash besteht aus folgenden Elementen, die im Header des vorherigen Blocks waren:

- Hash des vorherigen Block-Headers (= der der Block 2 vor dem jetzigen Block)
- Hash des Payloads
- die nonce-Zahl (mehr dazu im nächsten Absatz)

Dadurch, dass jeder Block-Header den Hash des vorherigen Block-Headers enthält, ergibt sich eine Verkettung. Ersetzt man nun also einen Block, so ist der Hashwert aller nachfolgenden Blöcke inkorrekt und der Betrugsversuch wird erkannt. Nun muss lediglich ein erster Pseudo-Block generiert werden, der den Anfang

der Kette darstellt. Er hat nur eine Pseudo-Payload und auch keine nonce-Zahl. Dieser Block wird auch Genesis-Block genannt.

Nun zu der Nonce-Zahl. Wie in dem anfänglichen Beispiel erwähnt, gilt im Falle einer Abweichung in der Buchführung zwischen „Inselbewohnern“ die Buchführung der Mehreheit. Im Internet kann man sich hierbei jedoch nicht auf irgendwelche Online-Identitäten verlassen, da man diese zuhauf mehrfach anlegen kann. Deshalb wird mit Rechenleistung „abgestimmt“: Es muss ein mathematisches Rätsel gelöst werden, dessen Lösung im Schnitt erst nach 10 Minuten berechnet werden kann. So kann eine Person erst dann mit „mehreren Stimmen wählen“, wenn sie auch mehrere bzw. leistungsfähigere Rechner anschafft. Das Rätsel besteht in der Bildung des Header-Hashes. Dort wird von einem Algorithmus ein Wert vorgegeben, den der Hash nicht überschreiten darf (z.B. die ersten 20 Zahlen müssen 0 sein). Da eine gute Hashfunktion zwar deterministisch (=selber Input selber Output) ist, das Ergebnis aber scheinbar zufällig ist, muss eine beliebige Zahl zu dem eigentlich zu hashenden Wert (Payload-Hash und Hash des vorgänger-Headers) hinzugefügt werden und hoffen, dass der entstehende Hash die Anforderungen erfüllt. Die Zahl, die einen richtigen Hash generiert ist die nonce (=Number used Once). Die Schwelle ist so gesetzt, dass es im Schnitt genügend Versuche benötigt, um eine passende Zahl zu finden, dass ca. 10 Minuten vergehen; Sie wird automatisch von einem Algorithmus angepasst, sollte es zu große Abweichungen geben. Natürlich kann es vorkommen, dass mehrere Menschen fast zeitgleich eine Lösung finden und diese in das Internet verbreiten. Hier gilt dann wie bereits die Kette, die die Mehrheit hat. Muss sich ein Nutzer entscheiden, welche der beiden Ketten nun verwendet werden soll, so wird die längere Kette gewählt. Ab einem Unterschied von 6 Blöcken (also nach ca. 1 Stunde) kann recht sicher davon ausgegangen werden, dass sie die dominierende Kette nicht mehr ändert.

Doch wer berechnet freiwillig diese ganzen Rätsel? Hierfür gibt es zwei Anreize: Jeder, der eine Transaktion ausführen möchte, muss eine frei wählbare Transaktionsgebühr bezahlen. Wer auch immer die Transaktion am Ende validiert bekommt die Gebühr (entsprechend werden Transaktionen mit höheren Gebühren auch schneller validiert). Den größeren Anreiz bietet allerdings das System selbst: Wer auch immer das Rätsel knackt bekommt vom System einen festgelegten Geldbetrag gutgeschrieben. Diese Transaktion wird noch in denselben Payload-Block eingetragen, den er soeben validiert. Insgesamt funktioniert eine Blockchain-Transaktion also so:

1. A möchte B einen Betrag senden und erstellt einen Datensatz mit folgenden Informationen
   - Signierter Hash des Public-Keys des Absenders (A selbst)
   - Signierter Hash des Public-Keys des Empfängers (In diesem Beispiel B)
   - Zu überweisender Betrag
   - Transaktionsgebühr
2. Der Datensatz kommt in einen Pool für unvalidierte Transaktionen
3. Eine weitere Person sucht sich so viele Transaktionen heraus, bis der Payload-Block fast voll ist
4. Die Person trägt die Transaktion vom System an sich in die Payload ein
5. Für jeden Datensatz wird nun folgendes getan
   - die komplette Kette durchsucht und das Guthaben des Senders berechnet und mit der Transaktionssumme verglichen
   - Die Public-Keys und Signaturen der Transaktion überprüfen
6. Gibt es nichts zu beanstanden, so wird der Merkle-Tree-Root-Hash berechnet
7. Danach wird der Hash des vorherigen Block-Headers berechnet
8. Dann wird mit diesen beiden Hashes die passende nonce ermittelt, um das Rätsel zu lösen.
9. der Block wird veröffentlicht

Nun ist die Validierung der in der Payload enthaltenen Blöcke abgeschlossen (sofern es zu keinem „Wettrennen“ von Lösungen verschiedener Personen kommt).

Auf diese Weise ist eine dezentrale Zahlungsmethode ohne Banken oder sonstige Infrastruktur (z.B. Trustcenter) gegeben, womit es keinen Single Point of Failure gibt. Durch diese Organisation benötigen wir kein

Vertrauen in die vorhandenen Institutionen, die Transaktionen sind anonymer und das Protokoll ist quelloffen und somit transparent. Zudem gibt es keine dominierende Partei innerhalb der Blockchain, die genug marktmacht hätte, um irgendetwas zu manipulieren. Allerdings hat die Blockchain auch Nachteile:

- Durch das kompetitive Berechnen der Rätsel-Lösungen mit dem „The Winner Takes it All"-Prinzip wird sehr viel Rechenleistung verschwendet
- Das Nachverfolgen von Zahlungen ist komplizierter
- Durch dezentrale Struktur kann es zu kurzzeitigen Unterschieden in der Blockchain kommen und bereits bestätigte Transaktionen wieder rückgängig gemacht werden.
- Durch die dezentrale Struktur gibt es keine Verantwortlichen im Falle eines Fehlers
- Durch die Belohnung für das Rätsellösen wird zuvor nicht vorhandenes Geld in das System geschleust und somit eine Inflation herbeigeführt. Einmal eingeführtes Geld kann allerdings nicht mehr eingezogen werden, wo mit es zwangsläufig zu einer Deflation kommt
- Bei abnehmenden Belohnungen geht auch der Anreiz für das Berechnen der Lösungen verloren, sodass irgendwann keine Transaktionen mehr bestätigt werden
- Verliert man seinen digitalen Schlüssel, verliert man auch sein komplettes Geld. Bei Banken hingegen ist es möglich sich erneut zu authentifizieren, um den Zugang zu erlangen.

10 Mathematische Hintergründe

10.1 Modulare Arithmetik

Ein nützliches Hilfsmittel ist die Modulo-Funktion. Diese Kann wie folgt definiert werden:

Definition 10.1: Modulo-Funktion

seien $a, b \in \mathbb{N}$ und $b > 0$. Dann gibt es eindeutige natürliche Zahlen q und r mit $a = q \cdot b + r$ und $r < b$.

Diese Definition wird nachfolgend bewiesen:

b muss größer sein als null, und kleiner gleich a. b ist also mindestens 1, oder größer. Multipliziert man also a und b, so muss das Ergebnis entweder a sein (im Fall von $b = 1$), oder größer:

$$0 < b \leq a \land a \cdot b \geq a$$

Zugleich gibt es q. q ist die größte Zahl, mit der b multipliziert werden kann, ohne dass das Produkt größer wird als a. Dementsprechend, erhöht man q um 1, so wäre das Produkt also größer.

$$q \cdot b \leq a \land (q + 1) \cdot b = q \cdot b + b > a$$

Zuletzt gibt es noch ein r, das den Rest bildet, also die Zahl, die noch benötigt wird, um die Differenz zwischen qb und a zu überbrücken. Durch die Anforderung von q und b, dass $qb + b > a$ ist, bedingt sich also, dass r kleiner sein muss als b, da der Wert von a ansonsten überstiegen wäre.

$$r = a - qb < b$$

Dass es mehrere Ergebnisse für diese Rest-Berechnung geben könnte ist ausgeschlossen, was durch nachfolgenden Beweis gezeigt wird:

Angenommen es gibt zwei Paare r_1, q_1 und r_2, q_2. Da sich die Paare nicht identisch sind, muss natürlich ein r größer sein als das andere. Um diesen Unterschied wettzumachen, muss das q des kleineren r natürlich größer sein als das q des größeren r. Somit ergeben sich die Bedingungen

$$r_1 > r_2 \land q_1 < q_2$$

Zudem nehmen wir natürlich an, dass beide Rest-Berechnungen korrekt sind und sie somit jeweils die untenstehenden Gleichungen erfüllen

$$a = q_1 b + r_1 \land a = q_2 b + r_2$$

Da beide demselben Wert entsprechen, lassen sie sich gleichsetzen

$$q_1 b + r_1 = q_2 b + r_2$$

Formt man diese Gleichung nun um und packt die beiden qs (und das b) und die rs zusammen, erhält man folgende Gleichung

$$(q_2 - q_1)b = r_1 - r_2$$

Da die beiden qs unterschiedlich Werte besitzen, ist das Ergebnis auf jeden Fall mindestens 1 oder größer. Dementsprechend muss das Ergebnis der Subtraktion der rs ebenfalls mindestens b betragen oder mehr. Wenn bereits die Differenz $\geq b$, dann ist ein einzelnes r auf jeden Fall größer als b. Bei obiger Definition wurde jedoch festgelegt, dass $r < b$, was im Widerspruch zu diesem Ergebnis steht. Somit ist bewiesen, dass es nur ein richtiges Ergebnis einer Modulo-Berechnung geben kann.

Definition 10.2: Restgleichheit

Seien $a, b \in \mathbb{Z}$ und sei $a = br + r$, dann schreibt man

$$r = a \mod b$$

Zwei Zahlen $a, b \in \mathbb{Z}$ heißen *restgleich*, wenn $a \mod n = b \mod n$. Man schreibt $a \equiv b \mod n$ und spricht *a ist kongruent zu b modulo n*
Wenn zwei Zahlen bei Division durch n den gleichen Rest haben, ist ihre Differenz ein Vielfaches des Moduls von n. Für $a, b \in \mathbb{Z}$ gilt also

$$a \equiv b \mod n \Leftrightarrow (a - b) \text{ ist teilbar durch } n$$

Der Beweis von links nach rechts lautet wie folgt:
Wenn $a \equiv b \mod n$, dann haben a und b bei der Division durch n den gleichen Rest r, sodass gilt

$$a = q_1 n + r \wedge b = q_2 n + r$$

Formt man diese beiden Gleichungen un jeweils nach r um und setzt diese dann gleich, so erhält man

$$a - b = (q_1 - q_2)n$$

Dadurch ist bewiesen, dass die Differenz der beiden Zahlen ohne Rest durch n teilbar ist, da sie ein Vielfaches von n ist.

Der Beweis von rechts nach links ist etwas komplizierter:

Gegeben ist, dass $a - b$ durch n teilbar ist. Es muss also eine weitere Zahl $q \in \mathbb{N}$ geben, die mit n multipliziert wird, sodass das Ergebnis der Multiplikation gleich der Differenz ist:

$$a - b = qn$$

gemäß der Definition des Modulo können a und b auch jeweils in ihrer „Modulo-Form“ geschrieben werden:

$$a - b = q_a n + r_a \wedge b = q_b - r_b \Rightarrow a - b = (q_a n + r_a) - (q_b n - r_b) = qn$$

Dieser Term kann nun so umgeformt werden, dass die $q_{a/b}$s und die rs zusammenstehen:

$$(q_a - q_b)n + (r_a - r_b) = qn$$

Nun können noch alle Terme mit n zusammengefasst werden:

$$(r_a - r_b) = (q - q_a + q_b)n$$

Nach dieser Formel muss $r_a - r_b$ durch n teilbar sein, da für alle r jedoch gegeben ist, dass $r < b$ (b ist in unserem Fall das n), muss $r_a - r_b = 0$ sein, und somit sind a und b restgleich.

10.2 Primzahlen

Definition 10.3: Primzahl

Eine natürliche Zahl $n > 1$ heißt *Primzahl*, wenn sie nur durch 1 und sich selbst ohne Rest teilbar ist.

Es gibt unendlich viele Primzahlen, die jedoch in ihrer Häufigkeit abnehmen. Bewiesen werden kann das, durch einen Widerspruchsbeweis: Man nehme eine hypothetische Menge, die alle Primzahlen enthält. Nun multipliziert man alle Zahlen dieser Menge miteinander und addiert eine 1 hinzu – man erhält eine neue Primzahl, die nicht in der Menge enthalten ist, die doch eigentlich alle Zahlen enthalten sollte. Somit ist bewiesen, dass es keine endliche Menge an Primzahlen geben kann.

10.3 Gruppen

Eine Gruppe ist eine Kombination aus Zahlenmenge und beliebigen Operation(en) (nachfolgend dargestellt als $\circ$), die bestimmte Bedingungen erfüllen müssen.

Angenommen man möchte überprüfen, ob die Menge $M = \{0, 1, 2, 3, 4\}$ und die Operation $\circ$ eine Gruppe bilden, so muss zuerst auf die *0. Bedingung* überprüft werden: Die Operation führt nicht aus der Menge hinaus. Wenn wir nun unsere Operation $\circ$ definieren als $a + b$, so ist diese Bedingung nicht erfüllt, da $4 + 1 = 5 \land 5 \notin M$. Somit muss $\circ$ anders definiert werden: $(a + b) \mod 5$. Nun führt sie nicht mehr aus der Menge heraus, da $(4 + 1) \mod 5 = 5 \mod 5 = 0$.

Die nächste Bedingung ist, dass das *Assoziativgesetz* gelten muss, also dass $(1 \circ 2) \circ 3 = 1 \circ (2 \circ 3)$. Im oben definierten Beispiel ist dies der Fall.

Zudem braucht die Gruppe ein *neutrales Element* n, das sowohl von Rechts als auch von Links mit einem anderen Element verknüpft werden kann, ohne dass sich der Wert des Ergebnisses ändert: $\forall a \in M : a \circ n = a = n \circ a$ In unserem Fall ist das Neutrale Element die 0.

Eine weitere Bedingung ist, dass für jedes Element der Gruppe ein *inverses Element* besteht, das verknüpft mit dem entsprechenden Element das neutrale Element ergibt: $\forall a \in M : a \circ b = n$. In unserem Fall ist das neutrale Element $n = 0$ und für das Beispiel $2 \circ ? = 0$ wäre 3 das inverse Element, da $(2 + 3) \mod 5 = 0$.

Definition 10.4: Gruppe

Eine Gruppe besteht aus einer Menge M und einer Operation $\circ$ und muss folgende Bedingungen erfüllen:

0. Die Operation darf nicht aus der Menge herausführen
1. Die Operation muss assoziativ sein
2. Die Operation muss ein neutrales Element besitzen
3. Für jedes Element aus M muss es ein inverses Element, ebenfalls aus M, geben.

Zudem gibt es noch weitere Strukturen, die aber nicht mehr viel Abweichen:

Definition 10.5: Ring

Ein Ring ist eine Struktur aus einer Menge M und *zwei* Operationen $\circ$ und $\dagger$, für die folgende Bedingungen gelten:

- Für Beide Operationen müssen Die Bedingungen 0, 1, 2 und das Distributivgesetz $(a \circ (b \dagger c) = a \circ b \dagger a \circ c)$ gelten.
- Für eine der Operationen (z.B. $\circ$) muss zusätzlich das Kommutativgesetz gelten – es ist also eine kommutative Gruppe – und es muss ein inverses Element geben.
- Für die andere Operation ($\dagger$) muss hingegen zusätzlich nur ein neutrales Element existieren und sie muss assoziativ sein.

Dies ist ebenfalls in Tabelle 10.1 dargestellt.

Definition 10.6: Körper

Die Definition ist analog zu der eines Rings, allerdings muss für die Zweite Operation $\dagger$ ebenfalls das Kommutativgesetz gelten und ein inverses Element existieren.
Zudem ist jeder Körper *nullteilerfrei*, was bedeutet, dass für alle $x, y \in M \backslash \{0\}$ gilt $x \dagger y \neq 0$

Bei multiplikativen Restklassen ist dies nur erfüllt, wenn der Modulo der Restklasse eine Primzahl ist, da bei der Multiplikation eines Teilers des Modulos sonst der Modulo selbst und somit 0 herauskommt, sodass diese Restklasse per Definition kein Körper sein kann.

Gesetz	$\circ$	$\dagger$
0	ja	ja
1	ja	ja
2	ja	ja
3	ja	nein
Distributiv	ja	ja
Kommutativ	ja	nein

Tabelle 10.1: Bedingungen für einen Ring

Für einen Ring $(\mathbb{Z}_n, \circ, \dagger)$ gilt also, wenn n Primzahl $\Leftrightarrow (\mathbb{Z}_n, \circ, \dagger)$ ist ein Körper. Der Beweis hierfür ist nachfolgend erklärt; zuerst von rechts nach Links:

Damit für zwei beliebige Elemente x, y aus einem Körper $\mathbb{Z}_n$ gilt $xy \mod n \neq 0$, muss auch gelten, dass für alle beliebigen Kombinationen $x, y \in \mathbb{Z}_n$ gilt $xy \neq n$, was der Definition einer Primzahl entspricht.

Nun in die andere Richtung; Eine Restklasse $\mathbb{Z}_n$ enthält alle Zahlen von 0 bis $n-1$. Da eine Primzahl keine Teiler hat, kann sich bei verschiedenen Zahlen auch der Rest nicht wiederholen. Um dies zu beweisen, nimmt man eine beliebige Zahl $a \in \mathbb{Z}_n \wedge a \neq 0$ und multipliziert sie mit zwei anderen Zahlen $b, c \in \mathbb{Z}_n \wedge 0 \leq b < c < n$. Man vergleicht somit in einer Tabellenansicht also den Inhalt von zwei verschiedenen Spalten in einer Zeile. Geht man davon aus, dass die Werte gleich sind, so müsste gelten: $a \cdot b = a \cdot c$. Dies kann entsprechend umgeformt und mit dem Modulo berücksichtigt werden: $a \cdot (c - b) \equiv 0 \mod n$. Damit bei der Restebildung 0 als Ergebnis entsteht, muss das Ergebnis entweder 0, oder ein Vielfaches von n sein: $a \cdot (c - b) = x \cdot n$. Da aber $0 \leq b < c < n$ sind, kann die Differenz der Beiden kein Vielfaches von n sein. Somit ist bewiesen, dass alle Zahlen einer Zeile verschieden sind.

Da alle Zahlen einer Zeile verschieden sind, müssen bei $n - 1$ Spalten auch alle Zahlen von 0 bis $n - 1$ vorkommen. Da dies in jeder beliebigen Zeile einer beliebigen Zahl a geschehen kann, muss somit also gelten:

$$\{a \cdot 0, a \cdot 1, a \cdot 2, \ldots, a \cdot (n-1)\} = \{0, 1, 2, \ldots, n-1\}$$

Da 1 ebenfalls Teil dieser Menge ist, und bezüglich der Multiplikation das neutrale Element darstellt, muss also gelten, dass es für jedes a auch ein inverses Element gibt, da bei der Multiplikation der Zahlen auch irgendwann eine 1 auftaucht.

10.4 Euklidischer Algorithmus

Definition 10.7: Fermat und Inversen

Sei n eine Primzahl, dann gilt in $\mathbb{Z}_n$ für alle $a \neq 0$

$$a^{n-1} = 1$$

Als Nebeneffekt dieses Satzes resultiert ebenfalls

$$a^{-1} = a^{n-2}$$

Wie oben bereits erwähnt, gilt $\{a \cdot 0, a \cdot 1, a \cdot 2, \ldots, a \cdot (n-1)\} = \{0, 1, 2, \ldots, n-1\}$. Mit dieser Voraussetzung können wir also auch sagen $1 \cdot \cdots \cdot n - 1 = a \cdot 1 \cdot \cdots \cdot a \cdot (n-1)$ (Bitte beachten: Aus Faulheit habe ich nur $a \cdot b$ anstatt $a \cdot b \mod n$ geschrieben; Das ist beim Rechnen in Körpern aber durchaus üblich). Da $\mathbb{Z}_n$ ein Körper ist, können wir die Zahlen 2,3,...,n-1 kürzen (also mit der Inverse eines Faktors multiplizieren) und erhalten $a^{n-1} = 1$. Nun kann man noch etwas weiterrechnen und kommt auf den Zusammenhang $a^{n-1} = aa^{n-2} = 1$; Diesen Term kann man wieder kürzen (also mit der Inverse a^{-1} multiplizieren) und erhält $a^{-1} = a^{n-2}$.

Dies funktioniert jedoch nur bei Primzahlen; Doch was, wenn man bei einer Nicht-Primzahl das inverse Element braucht? Hierfür gibt es den euklidischen Algorithmus.

Mit dem euklidischen Algorithmus lässt sich ohne die aufwändige Primfaktorzerlegung ganz schnell der ggT zweier Zahlen a, b berechnen; a lässt sich, wie bereits in Punkt 10.1 beschrieben, auch darstellen als $a = q \cdot b + r$. Nun berechnet der Algorithmus erneut den Rest. Diesmal ist das neue a allerdings das alte b

und das neue b das alte r. Dies wird so lange wiederholt, bis $r = 0$; Das b ist der ggT von den ursprünglichen a, b. Ist das b in diesem Schritt 1, so sind die Zahlen *relativ prim*, also haben keinen gemeinsamen Teiler > 1, und es kann ein multiplikatives Inverses berechnet werden.

Definition 10.8: Euklidischer Algorithmus

Sei $ggT(a, b)$ der größte gemeinsame Teiler der Zahlen $a, b \in \mathbb{N}\backslash\{0\}$. Dieser ggT wird durch den euklidischen Algorithmus wie folgt berechnet:

$$ggT(a,b) = \begin{cases} ggT(b, a \mod b = r) & \text{falls } a \mod b \neq 0 \\ b & \text{falls } a \mod b = 0 \end{cases}$$

Beweisen, dass der ggT so berechnet werden kann, kann man dies wie folgt: Ist $a \mod b = 0$, so ist a restlos durch b teilbar und b ist der größte gemeinsame Teiler der beiden Zahlen. Ist $a \mod b \neq 0$, dann lässt sich a nach wie vor darstellen als $a = q \cdot b + r$. Nun gibt es eine Zahl d, die der ggT von a, b ist und somit a und b (und damit auch qb), sowie $r = a - qb$ teilt. Daher ist d auch ein Teiler von b und r; Und jeder Teiler von b und r ist auch ein Teiler von a. Damit ist bewiesen, dass $ggT(a, b) = d = ggT(b, r) = ggT(b, a \mod b)$.

Vom ggT kann natürlich auch wieder nach oben gerechnet werden, indem das jetzige b durch das vorhergehende $a - qb$ ersetzt wird. Ein Beispiel für $\mathbb{Z}_8$:

$$ggT(8, 3):$$

$$\begin{aligned} 8 &= 2 \cdot 3 + 2 \\ 3 &= 1 \cdot 2 + 1 \\ 1 &= 1 \cdot 1 + 0 \end{aligned}$$

Nun wird rückwärts wieder hochgerechnet und jeweils das Ergebnis der vorherigen Rechnung eingesetzt:

$$\begin{aligned} 1 &= 1 \cdot 1 \\ 1 &= 1 \cdot (3 - 1 \cdot 2) = 3 - 1 \cdot 2 \\ 1 &= 3 - 1 \cdot (8 - 2 \cdot 3) = 3 - 8 + 2 \cdot 3 = 3 \cdot 3 - 8 = 3 \cdot 3 - 1 \cdot 8 \end{aligned}$$

Rechnet man diesen Term nun $\mod 8$, so ergibt der Teil $-1 \cdot 8 = 0$, womit nur noch $1 = 3 \cdot 3 \mod 8$ übrig bleibt, und dies ist die multiplikative Inverse. Somit lässt sich ableiten:

Definition 10.9: Euklidischer Algorithmus und Multiplikative Inverse

Sei $d = ggT(a, b) \in \mathbb{Z}_n$. Dann gibt es ganze Zahlen x, y mit

$$d = ggT(a, b) = ax + by$$

Sei $ggT(a, n) = 1$, dann gibt es ein x, y mit $1 = ax + ny$ und

$$1 = 1 \mod n = ax \mod n + ny \mod n = ax \mod n$$

und somit gilt $x = a^{-1}$

10.5 Die Eulersche Phi-Funktion

Die Eulersche Phi-Funktion $\varphi(n)$ gibt die Anzahl der natürlichen Zahlen $< n$ zurück, die zu n teilerfremd sind:

$$\varphi(n) = |\{0 \leq k < n | ggT(k, n) = 1\}|$$

Die Phi-Funktion ist eine multiplikative Funktion, sodass für zwei Teilerfremde Zahlen gilt

$$\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$$

Da Primzahlen mit alle Zahlen teilerfremd sind, gilt für eine Primzahl p: $\varphi(p) = p - 1$

Eine Potenz p^k, wobei p eine Primzahl ist, hat nur p als Primfaktor. Daher hat p^k nur mit Vielfachen von p und 1 einen gemeinsamen Teiler. Bei 1 bis p^k sind das die Zahlen

$$1 \cdot p, 2 \cdot p, \ldots, p^{k-1} \cdot p = p^k$$

Das sind p^{k-1} Zahlen, die nicht zu p^k teilerfremd sind. In diesem Fall gilt:

$$\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1) = p^k \left(1 - \frac{1}{p}\right)$$

Da sich jede Zahl n in verschiedene Primfaktoren zerlegen $n = \prod_{p|n} p^{k_p}$ lässt, kann die Zahl in die jeweiligen Potenzen aufgespalten werden, das Phi dieser Potenzen mittels der oben genannten Formel berechnet werden und diese aufgrund der Multiplikativen Kommutativität der Funkion wieder miteinander multipliziert werden. Somit lässt sich sagen

$$\varphi(n) = \prod_{p|n} p^{k_p - 1}(p-1) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

Wenn der Größte gemeinsame Teiler zweier Zahlen a und n $ggT(a,n) = 1$ ist, dann gilt $a^{\varphi(n)} \mod n = 1$. Ferner gilt, wenn $ggT(a,n) = 1$, dann gilt auch $a^b \mod n = a^{b \mod \varphi(n)} \mod n$; Dies lässt sich wie folgt beweisen:

Wie wir von der Modulo-Rechnung bereits wissen, lässt sich b auch schreiben als $b = q\varphi(n) + r$. Somit lässt sich schreiben

$$a^b \mod n = a^{q\varphi(n)+r} \mod n$$

Gemäß den Rechenregeln von Potenzen können wir das $+r$ als eine Multiplikation derselben Basis schreiben, sodass sich ergibt

$$a^{q\varphi(n)+r} \mod n = ((a^{q\varphi(n)} \mod n) \cdot (a^r \mod n)) \mod n$$

Das $a^{q\varphi(n)}$ können wir in q Multiplikationen von $a^{\varphi(n)}$ aufspalten, sodass sich ergibt

$$((a^{q\varphi(n)} \mod n) \cdot (a^r \mod n)) \mod n = (\underbrace{(a^{\varphi(n)} \mod n) \cdot \ldots \cdot (a^{\varphi(n)} \mod n)}_{\text{q Mal}} \cdot (a^r \mod n)) \mod n$$

Wie wir wissen, gilt für $ggT(a,n) = 1$: $a^{\varphi(n)} \mod n = 1$. Somit lässt sich unsere Gleichung schreiben als

$$(\underbrace{1 \cdot \ldots \cdot 1}_{\text{q Mal}} \cdot (a^r \mod n)) \mod n = a^r \mod n = a^{b \mod \varphi(n)} \mod n$$

10.6 Der Galois-Körper

Bisher haben wir gelernt, dass ein Körper genau dann als solcher gilt, wenn es zwei Operationen gibt, die alle in Punkt 10.3 genannten Bedingungen erfüllen. Dies ist Allerdings nur der Fall, wenn der Zahlenraum der einer Primzahl ist. In der Verschlüsselung, wäre es jedoch sehr wichtig auf Byte-Ebene, also 2^8 zu arbeiten – das ist aber keine Primzahl und somit können nicht alle Bedingungen für einen Körper erfüllt werden.

Hier stellt der Mathematiker Évariste Galois die verschiedenen Zahlen jedoch als Polynom dar; Aus 57 wird binär 01010111. Dieses wird in ein Polynom umgewandelt, indem die Ziffern als Koeffizienten eines Polynoms interpretiert werden. Der maximale *Grad* des Polynoms (also das höchste $x^{\text{irgendwas}}$) entspricht hierbei der Anzahl der Bits - 1:

0		1		0		1		0		1		1		1
0	$\cdot x^7$	$+1$	$\cdot x^6$	$+0$	$\cdot x^5$	$+1$	$\cdot x^4$	$+0$	$\cdot x^3$	$+1$	$\cdot x^2$	$+1$	$\cdot x$	$+1$

Somit ergibt sich die Zahl 57 als Polynom dargestellt: $x^6 + x^4 + x^2 + x + 1$

10.6.1 Addition

Polynome werden addiert, indem wir ihre Koeffizienten addieren. Da wir eigentlich mit Binärzahlen und somit im Zahlenraum $\mathbb{Z}_2$ rechnen, ergibt $1+1=0$. Somit XOReen wir eigentlich die beiden zugrundeliegenden Binärzahlen und bilden daraus wieder das entsprechende Polynom.

z.B. ist 57 (01010111) + 131 (10000011):

$$(x^6+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^6+x^4+x^2$$

Was 212 (11010100) entspricht. Dieses Ergebnis lässt sich auch mit dem binären XOR berechnen:

$$\begin{array}{cccccccccc} & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ \oplus & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}$$

Somit haben wir bereits die erste Verknüpfung in unserem angestrebten Körper, denn das XOR ist eine Gruppe.

10.6.2 Multiplikation

Die Multiplikation zweier Polynome geschieht unter den normalen Rechenregeln, das jedes Element mit jedem anderen Element multipliziert wird. So ergibt sich bei $3 \cdot 5$

$$(x^2+1)\cdot(x+1)=x^3+x^2+x+1$$

was 17 entspricht. Angenommen wir befinden uns uns allerdings gerade im $\mathbb{Z}_{16}$ (also einem halben Byte), so verlässt unsere Lösung den vorgegebenen Zahlenraum. Bei den gewöhnlichen Zahlen konnten wir hier einfach mod 16 rechnen, aber wie funktioniert das bei Polynomen?

Zuerst brauchen wir ein *irreduzibles Polynom*. Das ist ein Polynom, das sich nicht durch die Multiplikation zweier anderer Polynome ausdrücken lässt, also das Äquivalent einer Primzahl, nur als Polynom. Unsere Modulo-Berechnung geschieht dann mittels *Polynomdivision* und der Rest dieser Division ist dann unser Ergebnis innerhalb des vorgegebenen Zahlenraums. Zusammen mit der XOR-Addition erhalten wir nun einen Körper.

10.6.2.1 Polynomdivision

Bei der Polynomdivision wird ein Polynom durch ein anderes geteilt:

$$(3x^4+x^2+2x+1):(x^2-1)$$

Hier wird nun Grad um Grad des zu teilenden Polynoms entfernt, indem wir das Teiler-Polynom mit einem entsprechenden Wert multiplizieren und dann abziehen. In diesem Beispiel müssten wir unser Teilerpolynom mit $3x^2$ multiplizieren und erhalten dann $3x^4-3x^2$. Dieses Ergebnis ziehen wir dann vom andern Polynom ab und erhalten $4x^2+2x+1$. Nun multiplizieren wir unser Teilerpolynom mit 4 und ziehen es ab; Somit erhalten wir als neues „Zählerpolynom“ $2x+5$. Weiter können wir nicht teilen, da nun der Grad des Zählerpolynoms den Grad des Teilerpolynoms unterschritten hat. Somit ist das Ergebnis der Polynomdivision $3x^+4$ Rest $2x+5$.

Innerhalb unserer vorgegebenen Zahlenraums von $\mathbb{Z}_2$ gelten natürlich zusätzlich entsprechend die mod - Rechenregeln.

10.6.3 Höhere Koeffizienten

Bisher hatten wir nur Koeffizienten von 0 oder 1, was Sinn ergibt, da wir diese mittels Bits dargestellt haben. Nun gibt es aber auch die Möglichkeit, ein Polynom zu bauen, das höhere Koeffizienten hat z.B. $17x^3+5x^2+35$. Hier wurden nun ganz gewöhnliche Zahlen verwendet, man könnte sie aber auch hexadezimal darstellen ($0x11x^3+0x05x^2+0x23$); Letzten Endes, lassen sich diese Zahlen allerdings wieder als Polynome darstellen, wie wir es bereits zu Beginn getan haben. Somit sind nun die Koeffizienten von Polynomen, selbst Polynome. Da kann man beim Berechnen leicht den Überblick verlieren; Eine übersichtliche Variante ist

es, die Multiplikation mittels Matrizen durchzuführen, so ergibt sich z.B. für die Multiplikation von zwei Polynomen dritten Grades $a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0$ und $b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$ folgender Term:

$$\begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

wobei d_n die Koeffizienten für das Ergebnispolynom sind. Bei der Matrix von a werden alle Koeffizienten von einer Zeile in die nächste einfach eins nach rechts verschoben.

Die Multiplikationen und Additionen, die im Rahmen dieser Operation ausgeführt werden, müssen natürlich auch alle innerhalb des vorgegebenen Galois-Körpers stattfinden.

Definitionsverzeichnis

Tabellenverzeichnis