Fakultät für Informatik

Bachelorarbeit
zur Erlangung des akademischen Grades eines
Bachelor of Science

Implementieren eines Plugins zur Bitshift-Chiffre für die Webanwendung CrypTool-Online (CTO)

1. Prüfer: Dr. S. Kleine
2. Prüfer: Prof. Dr. B. Esslinger

vorgelegt von:	Michael Schmid
Jahrgang:	Informatik 2021
Matrikelnummer:	1213549
Tag der Abgabe:	29. Februar 2024 (Update 1 vom 10.04.24)

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit befasst sich mit der Verbesserung der bestehenden Bit-Shift-App in CrypTool-Online (CTO) und der Übertragung auf eine neue Weboberfläche. CrypTool-Online ist eine Website, welche die alte und moderne Kryptografie für jeden verständlich und anwendbar machen möchte. [11]

Die Arbeit gliedert sich in vier Hauptkapitel. Im ersten Hauptteil (Kapitel 2) werden die für später notwendigen Begriffe definiert. Kapitel 3 erklärt das verbesserte Verschlüsselungsverfahren Bit-Shift und im darauffolgenden Kapitel 4 wird die Sicherheit dieses Bit-Shift-Verschlüsselungsverfahrens gegen Brute-Force-Angriffe und Angriffe mit Häufigkeitsanalyse untersucht. Kapitel 5 befasst sich mit der Implementierung beider Bit-Shift-Appversionen in CrypTool-Online. Dargestellt wird hier zuerst die schon vorhandene Bit-Shift-App und im Anschluss wird die Implementierung der neuen, verbesserten App vorgestellt.

Die Bit-Shift-Verschlüsselung wurde ursprünglich von Arthur Guiot auf der Website „CrypTools“ [14] vorgeschlagen. Die bisherige CrypTool-Online-Website [12] zeigt die alte Implementierung der Bit-Shift-App. Die in dieser Arbeit gestaltete neue Implementierung wird voraussichtlich im zweiten Quartal 2024 live gehen.

2 Grundlagen

Zu Beginn wird die Bit-Shift-Operation und das Base64-Verfahren erklärt, da diese den Hauptbestandteil der Bit-Shift-Verschlüsslung ausmachen.

2.1 Bit-Shift-Operation

Beim Bit-Shift-Verfahren werden Binärzahlen bitweise nach rechts oder links verschoben. Die Stellen, welche nach dem Verschieben frei bleiben, werden mit Nullen aufgefüllt. Kommt es bei einem Links-Shift zum Überlauf des Zahlenbereichs, werden die höchstwertigen Bits ersatzlos gestrichen. Bei einem Shift nach rechts werden die niederwertigsten Bits, welche unterhalb des Zahlenbereichs liegen, gestrichen. In der Informatik wird für einen Bit-Shift nach rechts das Zeichen $>>$ und für einen Bit-Shift nach links das Zeichen $<<$ verwendet.

Beispiel 2.1 Bit-Shift nach links
Die Zahl `00100010` soll drei Stellen nach links verschoben werden. Dafür verschiebt man die vorhandenen Stellen alle um drei nach links und fügt am Ende drei Nullen an.

$$\begin{array}{r} 0100\,0010 \quad << 3 \\ 010\,0001\,0000 \end{array}$$

Wird diese Binärzahl, bei einer maximalen Zahlendarstellung von 12 Bits, um weitere drei Stellen verschoben, so kommt es hier zu einem Überlauf des Zahlenbereichs. Dabei werden die höchstwertigen Bits ersatzlos gestrichen.

$$\begin{array}{r} 010\,0001\,0000 \quad << 3 \\ 0000\,1000\,0000 \end{array}$$

Bei diesem Beispiel werden die führenden Nullen mit aufgeschrieben, um die 8-Bit-Struktur zu erhalten, die notwendig ist, um eine Binärzahl in ein ASCII-Zeichen umzuwandeln.

Ein Bit-Shift kann auch durch Multiplikation realisiert werden. Dabei wird die zu verschiebende Binärzahl mit zwei hoch die Anzahl der zu verschiebenden Stellen multipliziert.

$$\begin{array}{ccccc} 0000\,0010 & * & 2^3 & = & 000\,0001\,0000 \\ 2 & * & 8 & = & 16 \end{array}$$

Beispiel 2.2 Bit-Shift nach rechts
Die Zahl `00100010` soll zwei Stellen nach rechts verschoben werden. Dafür verschiebt man die vorhandenen Stellen alle um zwei nach rechts und streicht die niederwertigsten Bits.

$$\begin{array}{c} 00100010 \quad >> 2 \\ 001000 \end{array}$$

2.2 Base64-Verfahren

Beim Base64-Verfahren wird eine Zeichenfolge in eine andere codiert, die lediglich 64 Zeichen als Grundlage hat. Dabei darf die gewählte Zeichenfolge nur Zeichen enthalten, welche sich binär codieren lassen. Die Umwandlung erfolgt hierbei mit der 8-Bit-ASCII-Tabelle, somit sind auch nur die 256 Zeichen als Eingabe für die Base64 Codierung erlaubt.
In dieser Arbeit wird die nach dem ISO 8859-1 Standard geltende 8-Bit-ASCII-Tabelle auf Windows-1252 mit 256 Zeichen verwendet. [1]
Dabei enthält die Umwandlungstabelle, wie der Name schon sagt, nur 64 Zeichen. Das sind alle 26 Buchstaben des Alphabets in Groß- und Kleinschreibung, die Zahlen von 0 bis 9 und zwei Sonderzeichen: das „+“ und der „/“. [19, S. 720]

Base64 wird vor allem für die Multipurpose Internet Mail Extensions, kurz MIME, verwendet, welche es ermöglicht, auch Sonderzeichen, zum Beispiel Umlaute, zu verwenden. Die Nachrichten werden dabei in Base64-Zeichen codiert und können anschließend in den unterschiedlichen Sprachversionen, wie ISO646, US-ASCII und allen Versionen des Extended Binary Coded Decimal Interchange Code, kurz EBCDIC, gleich dargestellt werden. [6]

Um eine Textnachricht in Base64 umwandeln zu können, muss diese zuerst in Binärschreibweise dargestellt werden. Dies erfolgt mit Hilfe der ASCII-Tabelle, die jedem Zeichen aus der Nachricht eine 8-Bit-Zahl zuweist. Anschließend ist zu prüfen, dass die Anzahl der Bits durch 24 teilbar ist. Falls nicht, muss die Zahlenfolge am Ende mit entsprechenden Nullen aufgefüllt werden, damit die Stellenanzahl ein Vielfaches von 24 ist.
Das liegt daran, dass die Binärzahl sowohl durch 6 als auch durch 8 teilbar sein muss. Beim Codieren wird die Binärzahl für die Base64-Darstellung in 6er-Blöcke aufgeteilt.

Binärzahl	Zeichen	Binärzahl	Zeichen	Binärzahl	Zeichen	Binärzahl	Zeichen
000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011111	f	101111	v	111111	/

Tabelle 2.1: Base64-Tabelle mit allen 64 Zeichen

Umgekehrt muss beim Decodieren die Anzahl der Binärstellen für die 8-Bit-ASCII-Darstellung in 8er-Blöcke aufgeteilt werden. Das kleinste gemeinsame Vielfache von 6 und 8 ist 24, daher muss die Stelligkeit der Binärzahl ein Vielfaches von 24 sein. [2] Wenn dabei ein oder mehrere Nullerblöcke (jeweils bestehend aus 6 Nullen) angehängt werden müssen, so sind diese bei der späteren Codierung nicht zu berücksichtigen, müssen aber trotzdem gekennzeichnet werden. Für jeden hinzugefügten Nullerblock wird bei der Codierung am Ende der Zeichenfolge ein „=“ hinzugefügt.

Die gesamte Binärzahlenfolge wird anschließend in Sechsergruppen eingeteilt und mittels der Base64-Umwandlungstabelle in das passende Zeichen umgewandelt. So erhält man die fertig codierte Nachricht.

Beispiel 2.3 Möchte man die Zeichenfolge „Üben“ codieren, so muss diese zuerst mittels ASCII in Binärzahlen dargestellt werden.

ASCII-Zeichen	Ü	b	e	n
Dezimalzahl	220	98	101	110
Binärzahl	11011100	01100010	01100101	01101110

Anschließend werden diese in Sechsergruppen dargestellt und geprüft, ob die Bit-Anzahl durch 24 teilbar ist. Im Beispiel ergibt sich die Zahl 32, die nicht restlos durch 24 geteilt werden kann. Es sind daher 16 Nullen zu ergänzen, um ein Vielfaches von 24 zu erhalten.

Die 6-Bit Binärzahlen werden nun mit der Umwandlungstabelle in die entsprechenden Zeichen codiert und die angehängten Nullergruppen zu „=“ umgewandelt.

Binärzahl	110111	000110	001001	100101	011011	10 0000	000000	000000
Base64-Zeichen	3	G	J	l	b	g	=	=

Aus dem Wort „Üben“ wird mittels Base64 somit die Zeichenfolge „`3GJlbg==`“.

Hier ist anmerkend noch zu erwähnen, dass die Eingabe von „Üben“ in die bisherige CrypTool-Online Base64-App eine andere Ausgabe liefert. Dies liegt daran, dass der Buchstabe „Ü“ ein nicht unterstütztes Zeichen in der bisherigen Implementierung ist und daher als die Zeichen „Ãœ“ darstellt wird.

3 Bit-Shift-Verschlüsselung

Die Bit-Shift-Verschlüsselung ist ein symmetrisches Verschlüsselungsverfahren. Bei diesem wird der gleiche Schlüssel sowohl zum Ver- als auch zum Entschlüsseln einer Nachricht verwendet. Daher ist es wichtig, dass der verwendete Schlüssel geheim gehalten wird. Sollte Angreifern der passende Schlüssel bekannt werden, könnten diese die versendete Nachricht einfach entschlüsseln. Damit der Schlüssel ausschließlich den legitimen Gesprächspartnern bekannt ist, muss dieser vor Beginn einer Kommunikation persönlich oder über ein anderes sicheres Verfahren ausgetauscht werden. Das gilt nur für die erste Nachricht, die mit einer symmetrischen Verschlüsselungstechnik verschlüsselt wird. Alle zukünftigen Schlüssel können bei Übersendung einer Nachricht mit übertragen werden.

Spezifischer kann das Bit-Shift-Verfahren den polyalphabetischen Substitutionsverfahren zugeordnet werden. [13, S. 38] Hierbei ist die Zuordnung von Klartext- und Geheimtext-Alphabet nicht statisch, sondern variabel und hängt meist von einem Schlüssel ab. Auch die Bit-Shift-Verschlüsselung nutzt einen Schlüssel, um die Abbildung des Klartextes auf den zugehörigen Geheimtext variabel zu halten. Der Schlüssel besteht dabei aus drei Parametern, die schon durch kleine Veränderungen eine andere verschlüsselte Nachricht ergeben.

Die Bit-Shift-Verschlüsselungstechnik besteht aus mehreren einzelnen Teilschritten, die im Folgenden genauer erklärt werden, beginnend mit dem Verschlüsseln von Nachrichten und anschließend mit der zugehörigen Entschlüsselung.

Definition 3.1 Sei $\mathcal{A}$ das Alphabet, das alle 8-Bit-ASCII-Zeichen enthält, und sei $\mathcal{B}$ das Alphabet, das alle Base64-Zeichen enthält, dann kann das symmetrische Kryptosystem Bit-Shift wie folgt eingeordnet werden:

$$\begin{array}{lll} \mathcal{P} & \subseteq & \mathcal{A}^n & \text{Klartextmenge} \\ \mathcal{C} & \subseteq & \mathcal{B}^m & \text{Geheimtextmenge} \\ \mathcal{K} & \subseteq & (\mathcal{A}^l \times \mathbb{N} \times \mathbb{N}) & \text{Schlüsselmenge} \end{array}$$

Dabei gilt: $n, m, l, \in \mathbb{N}$ und $n < m$

$$\begin{array}{ll} \text{Verschlüsselungsfunktion} & \mathbf{E} : \mathcal{P} \times \mathcal{K} \to \mathcal{C} \\ \text{Entschlüsselungsfunktion} & \mathbf{D} : \mathcal{C} \times \mathcal{K} \to \mathcal{P} \end{array}$$

Zudem gilt folgende Symmetrieeigenschaft: $\mathbf{D}(\mathbf{E}(p, k), k) = p, c \in \mathcal{C}, p \in \mathcal{P}, k \in \mathcal{K}$.

Es ist noch anzumerken, dass für die Klar- und Geheimtextmenge immer $n < m$ gilt. Denn durch die Base64-Codierung im letzten Verschlüsselungsschritt wird der Geheimtext immer um mindestens ein Drittel größer sein wie der Klartext, auch dann wenn gar kein Bit-Shift durchgeführt wurde. [2]

Schlüsselgenerierung Bevor mit einer verschlüsselten Kommunikation begonnen werden kann, müssen zuerst die drei gemeinsamen Schlüsselparameter festgelegt und zwischen den Kommunikationspartnern ausgetauscht werden. Dazu zählt das Schlüsselwort, welches später die Anzahl der Stellen des Bit-Shifts bestimmt, der maximale Shift-Wert b, welcher die maximale Verschiebung pro Schlüsselzeichen festlegt, und der Additionswert n, welcher beim Verschlüsseln addiert und beim Entschlüsseln subtrahiert wird. Formal lässt sich dies wie folgt beschreiben:

$$k = (w, b, n), w \in A^l, b \in \mathbb{N}, n \in \mathbb{N} \tag{3.1}$$

Bemerkung 3.2 Ursprünglich wurde für das Bit-Shift-Verfahren nur das Schlüsselwort w als Parameter verwendet. Um die Sicherheit des Verfahrens zu erhöhen, wurden die zwei Schlüsselparameter b und n hinzugefügt. Diese waren ursprünglich fest gewählt: $b = 8$, $n = 1$.

3.1 Verschlüsselung

Die vier Schritte der Bit-Shift-Verschlüsselung erfolgen für jedes Zeichen der zu verschlüsselnden Nachricht einzeln. Am Ende des Prozesses, nach der Base64-Umwandlung, entsteht daraus eine zusammenhängende Nachricht, die an den Empfänger übermittelt werden kann.

1. Schritt: Binärumwandlung Um eine Textnachricht mittels Bit-Shift-Verfahren zu verschlüsseln, müssen die einzelnen Textzeichen zuvor in Binärzahlen umgewandelt werden. Dies erfolgt mit Hilfe der ASCII-Tabelle, welche jedem Zeichen eine eindeutige Binärzahl mit 8 Bits zuweist.

2. Schritt: Verschiebung bestimmen Das gewählte Schlüsselwort $w = w_1, ..., w_l$ legt fest, um wie viele Stellen später der Bit-Shift durchgeführt wird. Dafür wird für jedes Schlüsselzeichen w_i mit Hilfe der ASCII-Tabelle die passende Dezimalzahl, a_i, bestimmt und diese anschließend modulo b gerechnet. Heraus kommt der Shift-Wert v_i, welcher im späteren Bit-Shift-Algorithmus, zur Verschiebung der Binärzahlen, verwendet wird. Dieser ist für alle Schlüsselzeichen w_i wie folgt definiert:

$$v_i = a_i \mod b \text{ für } i = 1, ..., l \tag{3.2}$$

3. Schritt: Bit-Shift-Algorithmus Das erste Byte wird zuerst plus n gerechnet und anschließend ein Bit-Shift nach links durchgeführt. Dieser Shift, erfolgt wie in Kapitel 2.1 beschrieben, mit der Besonderheit, dass hier keine Bits gestrichen werden. Um wie viele Stellen genau der Bit-Shift durchgeführt wird, bestimmt das Ergebnis des ersten Schlüsselzeichens aus dem 2. Schritt. Nach dem Verschieben der Zahl wird diese mit n addiert und anschließend um das Ergebnis des zweiten Schlüsselzeichens erneut nach links verschoben. Die weiteren Additionen und das Bit-Shiften werden analog für alle weiteren Zeichen des Schlüsselwortes durchgeführt. Nach der letzten Verschiebung wird die Zahl nicht mehr mit n addiert, sondern direkt in ein Array gespeichert. Dieses Array enthält nach Abschluss aller Operationen alle verschobenen Binärzahlen der zu verschlüsselnden Nachricht. Formal lässt sich dieser Schritt wie folgt beschreiben: Sei $p \in \mathcal{P}$ der zu verschlüsselnde Klartext. Für jedes p_j wird eine Schleife über v_i durchlaufen:

$$m_0 = p_j;\ m_i = (m_{i-1} + n) << v_i \text{ für } i = 1, ..., l;\ c_j = m_l \tag{3.3}$$

Immer wenn eine verschobene Binärzahl im Array abgespeichert wurde, ist das Schlüsselwort umzudrehen, bevor die nächste Binärzahl mit Hilfe der Schlüsselzeichen verschoben wird. Jedes zweite Nachrichtenzeichen wird somit mit dem gleichen Schlüssel verschoben.

4. Schritt: Base64-Umwandlung Die in Schritt 3 erwähnten Schritte werden für alle Zeichen der Nachricht durchgeführt, bis alle Zeichen als Binärzahl im Array gespeichert sind. Anschließend werden alle Array-Einträge in Dezimalzahlen umgewandelt, da diese sich schneller mit Base64 codieren lassen.
Wie in Kapitel 2.2 beschrieben, wird jedes Zeichen einzeln in die Binärdarstellung gebracht, bevor diese anschließend in Base64-Zeichen codiert werden. Für eine 1-Byte-Binärzahl muss dieser Schritt acht Mal durchgeführt werden. Um die gleiche Zahl in Dezimaldarstellung zu codieren, werden höchstens 3 Umwandlungen benötigt.
Das Dezimalarray wird mit Hilfe von Base64 wieder in eine Zeichenfolge umgewandelt. Dabei ist zu beachten, dass sowohl die eckigen Klammern am Anfang und Ende des Arrays als auch die Kommas zur Abtrennung der einzelnen Einträge mit codiert werden. Die dadurch erhaltene verschlüsselte Nachricht kann nun an den Empfänger gesendet werden.

Beispiel 3.3 Es wird ein Beispiel betrachtet, für das die Werte $b = 8$ und $n = 1$ festgelegt wurden. Um die Nachricht „Bayern“ mit Bit-Shift zu verschlüsseln, muss sie zuerst in Binärzahlen umgewandelt werden.

Nachrichtenzeichen	B	a	y	e	r	n
Dezimalzahl	66	97	121	101	114	110
Binärzahl	01000010	01100001	01111001	01100101	01110010	01101110

Alle Zeichen werden einzeln verschlüsselt und es wird mit dem ersten Buchstaben der Nachricht begonnen. Die Binärzahl von „B“ wird erst um eins erhöht und anschließend

ein Bit-Shift nach links durchgeführt. In diesem Beispiel ist das Schlüsselwort „Cryp“. Zuerst werden alle Schlüsselzeichen mittels ASCII in eine Dezimalzahl umgewandelt und Modulo acht gerechnet. Der Buchstabe „C“ hat nach ASCII die Zahlendarstellung 67. Diese Modulo acht gerechnet ergibt einen Rest von 3. Diese Operation für alle anderen Zeichen fortgeführt ergibt folgende Schlüsseltabelle:

Schlüsselzeichen	C	r	y	p
Dezimalzahl	67	114	121	112
Modulo 8	3	2	1	0

Das erste Nachrichtenzeichen wird zuerst mit eins addiert. Anschließend wird ein Links-Bit-Shift um drei Stellen durchgeführt und die entsprechenden Nullen rechts angehängt. Führt man das Prinzip der Addition von eins und das anschließende Anhängen von Nullen mit den nächsten Schlüsselzeichen fort, ergibt sich folgende Tabelle:

Binärzahl	01000010
+1	01000011
1. Bit-Shift	01000011000
+1	01000011001
2. Bit-Shift	0100001100100
+1	0100001100101
3. Bit-Shift	01000011001010
+1	01000011001011
4. Bit-Shift	01000011001011

Anschließend wird die erhaltene Binärzahl in eine Dezimalzahl umgewandelt und in einem Array abgespeichert. Zudem wird das Schlüsselwort umgedreht und diese Schritte für alle folgenden Nachrichtenzeichen gleichermaßen angewendet.

Wurden alle Zeichen der Nachricht in eine Binärzahl umgewandelt, bitweise mit Hilfe des Schlüsselwortes verschoben und in Dezimalzahldarstellung gebracht, erhält man folgendes Array: [4299,6376,7819,6632,7371,7208].

Jedes dieser 31 Zeichen (inklusive Klammern und Kommas) wird nun einzeln mit Base64 in Nachrichtenzeichen codiert. Es ergibt sich folgende zu übermittelnde Textnachricht:

„WzQyOTksNjM3Niw3ODE5LDY2MzIsNzM3MSw3MjA4XQ==“

Da die Klammern des Arrays bei jeder Verschlüsselung immer mit verschlüsselt werden, beginnen alle verschlüsselten Nachrichten immer mit einem großen „W“. Das „[“-Zeichen binär codiert, nach der 8-Bit-ASCII-Tabelle, lautet `01011011`. Die ersten sechs Zeichen hieraus ergeben nach der Base64-Tabelle, aus 2.1, den Buchstaben „W“. Nachdem die ersten sechs Bits nie verändert werden, fängt somit jede Nachricht mit diesem Buchstaben an. Die übrigen zwei Bits bestimmen den Beginn des nächsten codierten Zeichens. Dies hat zur Folge, dass an zweiter Stelle nur eines der letzten 16 Zeichen, „w“ bis „/“, aus der in 2.1 beschriebenen Base64-Tabelle stehen kann. Dadurch wird die Anzahl an Möglichkeiten für das zweite Zeichen der Nachricht eingeschränkt, was es für Brute-Force-Angriffe einfacher macht, das richtige Zeichen zu erraten. Auf die Sicherheit des Bit-Shift-Verfahrens wird in Kapitel 4 noch genauer eingegangen.

3.2 Entschlüsselung

Um eine mit Bit-Shift verschlüsselte Nachricht lesen zu können, muss diese zuvor richtig entschlüsselt werden. Seien ein Geheimtext c und ein Schlüssel $k = (w, b, n)$ gegeben.

1. Schritt: Base64-Umwandlung Die erhaltene Zeichenfolge wird mit Hilfe von Base64 wieder in ein Array mit Dezimalzahlen zurück codiert und in Binärdarstellung gebracht, um später den Bit-Shift durchführen zu können. Anschließend kann jeder Eintrag des Arrays einzeln entschlüsselt werden. Es gilt zu beachten, dass die Einträge in der richtigen Reihenfolge von links nach rechts entschlüsselt werden müssen.

2. Schritt: Verschiebung bestimmen Wie bei der Verschlüsselung legt das gewählte Schlüsselwort fest, um wie viele Stellen die Bit-Shift-Operation durchgeführt wird. Um diesen Wert zu berechnen, wird, wie bei der Verschlüsselung im Schritt 2 schon beschrieben, für jedes Schlüsselzeichen mit Hilfe der ASCII-Tabelle die Dezimalzahl bestimmt und diese anschließend Modulo b gerechnet.

3. Schritt: Bit-Shift-Algorithmus Um die Bit-Shift-Operation der Verschlüsselung rückgängig zu machen, also die Nachricht zu entschlüsseln, wird hier zuerst der Bit-Shift nach rechts durchgeführt und im Anschluss n subtrahiert. Die in Schritt 2 errechneten Shift-Werte geben an, um wie viele Stellen nach rechts verschoben wird. Nachdem die Entschlüsselung rückwärts zur Verschlüsselung funktioniert, muss das Schlüsselwort zu Beginn umgedreht werden. Das letzte Schlüsselzeichen gibt also als Erstes an, um wie viele Stellen der Bit-Shift nach rechts durchgeführt wird. Wie im Kapitel 2.1 beschrieben, werden hierbei die niederwertigsten Bits gestrichen. Nach dem Rechts-Bit-Shift wird die Zahl um n verringert und anschließend die Zahlenfolge um die entsprechenden Stellen des nächsten Shift-Werts verschoben. Diese Schritte werden so lange wiederholt, bis alle Zeichen des Schlüsselwortes durchlaufen wurden und eine Binärzahl mit acht Stellen übrig bleibt. Formal lässt sich dieser Schritt wie folgt beschreiben: Sei $c \in \mathcal{C}$ der zu

entschlüsselnde Geheimtext. Für jedes c_j wird eine Schleife über v_i durchlaufen:

$$m_0 = c_j;\ m_i = (m_{i-1} >> v_i) - n) \text{ für } i = 1, ..., l;\ p_j = m_l \tag{3.4}$$

Nach jedem Array-Eintrag wird der Schlüssel umgedreht und die nächste Binärzahl entschlüsselt.

Bemerkung 3.4 In der ursprünglichen Version des Bit-Shift-Verfahrens wurde zuerst subtrahiert und anschließend der Bit-Shift durchgeführt. Diese Reihenfolge funktioniert aber nur für die beiden Zahlen null und eins. Bei Zahlen größer eins wird diese Reihenfolge unzuverlässig und ist daher für die neue Version des Bit-Shift-Verfahrens unbrauchbar. Details werden im Folgenden erläutert.
Wird eine Binärzahl mit eins addiert, so hat dies nur Einfluss auf ihre letzte Bitstelle. Durch einen Übertrag können aber auch andere Bitstellen betroffen sein. Diese gesamte Veränderung wird aufgrund des Übertragsverhaltens bei der Subtraktion von eins wieder rückgängig gemacht.
Wird bei Binärzahlen eine eins von einer null subtrahiert, entsteht ein Übertrag, welcher zur nächsthöheren Stelle weitergegeben wird.
Wenn also beim Entschlüsselungsschritt eins subtrahiert wird, werden die im letzten Verschlüsselungsschritt angehängten Nullen zu Einsen und die eins wird dadurch von der Binärzahl vor dem Bit-Shift subtrahiert. Mit dem anschließenden Rechts-Bit-Shift werden die entstandenen Einsen gestrichen.
Hier muss dazu gesagt werden, dass dies nur für zuvor verschlüsselte Binärzahlen funktioniert und daher nicht auf zufällig gewählte Binärzahlen anwendbar ist.
Für Zahlen, welche größer eins sind, kann es vorkommen, dass es einen Unterschied macht, ob zuerst subtrahiert und anschließend verschoben wird oder umgekehrt. Denn die Addition mit einer Zahl größer als eins kann beim Verschlüsseln die letzte Stelle der Binärzahl unter Umständen nicht verändern. Wird allerdings vor dem Bit-Shift subtrahiert, kann dies ungewollt die letzte Stelle der ursprünglichen Binärzahl verändern und es kommt zu einem falschen Ergebnis. Zur einfacheren Verständlichkeit folgen anschließend zwei Beispiele. Beispiel 3.5 zeigt, dass es mit dem Wert eins, zumindest in diesem Beispiel, egal ist, ob zuerst der Shift durchgeführt oder subtrahiert wird. Beispiel 3.6 zeigt, dass dies bei Werten größer eins zu einem anderen Ergebnis führt.

Beispiel 3.5 Seien $c = \texttt{0100101000}, k = (\mathrm{R}, 8, 1)$.
Entschlüsselung zuerst mit Subtraktion der Zahl 1 und anschließend dem Bit-Shift:

$$\texttt{0100101000} - \texttt{1} = \texttt{0100100111} >> \texttt{2} = \texttt{01001001}$$

Entschlüsselung zuerst mit dem Bit-Shift und anschließend Subtraktion der Zahl 1:

$$\texttt{0100101000} >> \texttt{2} = \texttt{01001010} - \texttt{1} = \texttt{01001001}$$

Beispiel 3.6 Seien $c = \texttt{0100101100}, k = (\mathrm{R}, 8, 2)$.
Entschlüsselung zuerst mit Subtraktion der Zahl 2 und anschließend dem Bit-Shift:

$$\texttt{0100101100} - \texttt{2} = \texttt{0100101010} >> \texttt{2} = \texttt{01001010}$$

Entschlüsselung zuerst mit dem Bit-Shift und anschließend mit Subtraktion der Zahl 2:

$$\texttt{0100101100} >> \texttt{2} = \texttt{01001011} - \texttt{2} = \texttt{01001001}$$

Wie in den Beispielen gezeigt ist es bei der neuen Bit-Shift-Verschlüsselung erheblich, ob zuerst subtrahiert und danach der Shift durchgeführt wird oder umgekehrt. Deshalb ist folgende Definition notwendig:
In der neuen Bit-Shift-Version erfolgt die Entschlüsselung zuerst mit einem Bit-Shift und anschließend erfolgt die Subtraktion von n.

4. Schritt: ASCII-Umwandlung Nachdem alle Binärzahlen des Arrays entschlüsselt wurden, können diese mit Hilfe der ASCII-Tabelle wieder in Textzeichen umgewandelt werden. Anschließend fügt man diese zu einer Zeichenkette zusammen und erhält die entschlüsselte Nachricht.

Beispiel 3.7 Es wird ein weiteres Beispiel betrachtet, bei dem die Werte $b = 8$ und $n = 1$ festgelegt wurden. Um die erhaltene Nachricht „WzEwNTgsMTY4OF0=“ mit Bit-Shift zu entschlüsseln, muss diese zuerst mit Base64 decodiert werden. Aus dieser Nachricht ergibt sich dann das folgende Array: [1058,1688].

Da alle Zeichen einzeln entschlüsselt werden sollen, wird mit dem ersten Eintrag aus dem Array begonnen. Für die Binärzahl von „1058“ wird zuerst ein Bit-Shift nach rechts durchgeführt und anschließend eins subtrahiert. In diesem Beispiel ist das Schlüsselwort „KI“. Zuerst muss man alle Zeichen mittels ASCII in eine Dezimalzahl umwandeln und Modulo acht rechnen. Der Buchstabe „K“ hat nach ASCII die Zahlendarstellung 75. Diese Modulo acht gerechnet ergibt einen Rest von 3. Das für alle anderen Zeichen fortgeführt ergibt folgende Schlüsseltabelle:

Schlüsselzeichen	K	I
Dezimalzahl	75	73
Modulo 8	3	1

Beim Entschlüsseln wird gleich zu Beginn der Schlüssel umgedreht, um den Bit-Shift-Algorithmus rückwärts zu durchlaufen. Für die erste Dezimalzahl wird also als Erstes

ein Rechts-Bit-Shift um eine Stelle durchgeführt. Führt man das Prinzip der Subtraktion um eins und das anschließende Streichen der niedrigsten Stellen mit den nächsten Schlüsselzeichen fort, ergibt sich folgende Tabelle:

Binärzahl	010000100010
1. Bit-Shift	01000010001
-1	01000010000
2. Bit-Shift	01000010
-1	01000001

Nach dem ersten entschlüsselten Eintrag des Arrays wird der Schlüssel umgedreht und mit dem nächsten Eintrag, in gleicher Weise, fortgefahren. Um die gesamte entschlüsselte Nachricht zu erhalten, müssen am Ende alle Zeichen des Arrays zu einer Zeichenkette zusammengefügt werden.

Binärzahl	01000001	01101000
ASCII-Zeichen	A	h

4 Sicherheit

Für das Bit-Shift-Verfahren gibt es noch keine dokumentierte Sicherheitsanalyse. Daher wird im folgenden Kapitel die Sicherheit des Verfahrens untersucht. Zu Beginn werden die verschiedenen Angriffsszenarien auf Verschlüsselungsverfahren aufgezeigt. Anschließend wird darauf eingegangen, wie sicher die Bit-Shift-Verschlüsselung gegen Brute-Force-Angriffe und Häufigkeitsanalyse ist.

Es gibt vier Arten von Attacken. Liegt dem Angreifer nur ein verschlüsselter Text vor, so spricht man von einer Ciphertext-Only-Attacke. Diese Art ist die häufigste Angriffsform. Kennt der Angreifer auch noch den zugehörigen Klartext, so wird dieser Angriff Known-Plaintext-Angriff genannt. Von einem Chosen-Plaintext-Angriff wird gesprochen, wenn es dem Angreifer möglich ist, sich zu beliebig vielen frei gewählten Klartexten die zugehörigen verschlüsselten Nachrichten erzeugen zu lassen. Dabei ist dem Angreifenden aber nicht bekannt, welche Schlüsselparameter dafür verwendet wurden. Kann sich der Angreifer beliebige Geheimtexte, ohne das Wissen der Schlüsselparameter, entschlüsseln lassen, so wird von einem Chosen-Ciphertext-Angriff gesprochen. Es versteht sich von selbst, dass hierbei der Zieltext nicht als Geheimtext zum Entschlüsseln gewählt werden darf. [9, S. 79ff]

4.1 Brute-Force-Angriff auf Schlüsselwort

Um die Schlüsselparameter oder den entschlüsselten Text einer verschlüsselten Nachricht herausfinden zu können, gibt es mehrere Methoden. So kann mit einem Brute-Force-Angriff, durch stumpfes Ausprobieren aller Möglichkeiten, der verwendete Schlüssel erraten werden. Diese Methode ist aber nicht effizient und benötigt bei steigender Anzahl an Möglichkeiten immer mehr Rechenleistung oder mehr Zeit um den richtigen Schlüssel zu finden. Möchte man den Schlüssel dennoch durch Erraten herausfinden, so empfiehlt sich ein Wörterbuchangriff. Dabei werden nicht einfach alle Möglichkeiten nacheinander durchprobiert, sondern es gibt eine vordefinierte Liste, welche typisch verwendete oder schon bei anderen Angriffen herausgefundene Schlüssel enthält. Schlüsselparameter, die nicht computergeneriert sind, können dadurch schneller erraten werden, denn Menschen neigen dazu, kurze und wenig komplexe Parameter für die Verschlüsselung zu verwenden, um sich diese leichter merken zu können.

Das Bundesamt für Sicherheit in der Informationstechnik empfiehlt für symmetrische Verschlüsselungsverfahren eine Mindestschlüssellänge von 128 Bit. [8, S. 27] Wird für die Schlüsseleingabe eine 8-Bit-ASCII-Darstellung verwendet, so müssen dort mindestens 16 Schlüsselzeichen eingegeben werden. Damit das Schlüsselwort bei dieser Länge dennoch einfach zu merken ist, werden meist Sätze oder Wortreihungen gebildet. Durch das Verwenden von Wörtern gibt es aber, im Vergleich zu zufällig gewählten Zeichenfolgen, deutlich weniger Möglichkeiten.

Im folgenden Abschnitt soll genauer untersucht werden, mit wie viel Aufwand ein Brute-Force-Angriff auf das Schlüsselwort einer mit Bit-Shift verschlüsselten Nachricht verbunden ist. Bei der ursprünglichen Variante des Bit-Shift-Algorithmus musste lediglich das verwendete Schlüsselwort herausgefunden werden, um eine Nachricht entschlüsseln zu können. Der Schlüssel kann dabei aus einer beliebig langen Zeichenkette aller 8-Bit-ASCII-Zeichen bestehen (also nicht nur aus Worten aus dem Wörterbuch). Dabei können aber die ersten 32 Zeichen der Tabelle vernachlässigt werden, da diese nicht druckbare Steuercodes für Peripheriegeräte sind. Auch können die nicht belegten Einträge, sowie nicht eingebbare Zeichen, wie „DEL“ für das Löschen von Zeichen, vernachlässigt werden. Für ein Schlüsselzeichen bleiben so noch 218 mögliche Eingaben übrig. Für ein Schlüsselwort mit 8 Zeichen ergeben sich schon $8^{218} \approx 7 \cdot 10^{196}$ Möglichkeiten. Ein Computer mit guter Rechenleistung kann gut 170 Millionen verschiedene Schlüsselwörter pro Sekunde generieren und ausprobieren. [7] Es würde also ungefähr $4,4 \cdot 10^{188}$ Sekunden oder $4,4 \cdot 10^{181}$ Jahre dauern, bis alle möglichen Schlüssel ausprobiert wurden. Man kann also davon ausgehen, dass ein Schlüsselwort dieser Verschlüsselungstechnik mit Hilfe eines Brute-Force-Angriffs, zur jetzigen Zeit, nicht herausgefunden werden kann.

4.2 Brute-Force-Angriff auf Geheimtext

Die Bit-Shift-Verschlüsselung bietet aber auch eine andere Angriffsmöglichkeit, welche ohne die Kenntnis des Schlüsselwortes den Geheimtext entschlüsseln kann. Die verschlüsselte Nachricht besteht aus Base64-Zeichen, welche mit öffentlich zugänglichem Wissen wieder decodiert werden können. Das so erhaltene Dezimalzahlen-Array kann einfach in die zugehörigen Binärzahlen umgewandelt werden. Wenn für den Additionswert n die eins gewählt wurde, kann man leicht von den ersten 8-Bit der Binärzahlen die Zahl eins subtrahieren und erhält die ASCII-Darstellung der ursprünglich versendeten Zeichen. Diese dann noch umwandeln und schon erhält man den Klartext. Dies funktioniert allerdings nicht immer so einfach. Wurde für die Addition ein anderer Wert gewählt, so muss dieser erraten werden. Zudem werden bei der Umwandlung von Dezimalzahlen in Binärzahlen die führenden Nullen nicht mit angegeben, wodurch nicht immer die ersten 8-Bit das Zeichen repräsentieren. Der Großbuchstabe „A“ beispielsweise hat nach der ASCII-Tabelle die Dezimalzahl 65. Nach einer Base64-Codierung und anschließender Decodierung erhält man die Binärzahl `1000001`, welche nur sieben Stellen hat. Die

ersten 8-Bit ergeben somit nicht den Buchstaben „A“, sondern entweder ein einfaches niedriges Anführungszeichen „‚“ oder den lateinischen Kleinbuchstaben „f“ mit Haken „*f*“. Um dennoch mit dieser Methode die Nachricht zu entschlüsseln, muss für jeden Array-Eintrag überprüft werden, ob das entsprechende Zeichen durch 6, 7 oder 8 Bits repräsentiert wird. Im Anschluss müssen diese generierten Texte dann händisch oder mit speziellen Computerprogrammen daraufhin überprüft werden, ob der Ausgabetext einen sinnvollen Klartext bildet. Ein Nachrichtenzeichen kann entweder in eine 6-, 7- oder 8-Bit-ASCII-Zahl umgewandelt werden und für die Addition kann ein beliebiger Wert aus den natürlichen Zahlen gewählt werden. Der in CTO [11] für die Bit-Shift-App verwendete Beispieltext „*Franz jagt im komplett verwahrlosten Taxi quer durch Bayern.*“ hat 60 Zeichen. Alleine für die drei unterschiedlichen Längen der Binärzahlen gibt es $3^{60} \approx 4,24 \cdot 10^{28}$ Möglichkeiten. Diese verschiedenen Varianten von einem Rechner generieren zu lassen dauert, bei 170 Millionen Möglichkeiten pro Sekunde, schon ungefähr $2,5 \cdot 10^{20}$ Sekunden, beziehungsweise fast 8 Billionen Jahre reine Rechenzeit. Dazu kommt dann noch, dass für jede Möglichkeit alle Zahlen von eins bis 2^8 für die Subtraktion ausprobiert werden müssten.
Denn diese Angriffsart funktioniert nur, wenn der für die Verschlüsselung gewählte Additionswert n nicht die Stellenzahl der Binärzahl erhöht. Durch die Addition mit n kann es vorkommen, dass aus einer 8-Bit-Zahl eine 9-Bit-Zahl entsteht und die ersten 8-Bit nicht mehr das Klartextzeichen repräsentieren.
Zudem müssen am Ende die generierten Texte noch von einem Menschen oder einem zusätzlichen Programm überprüft werden, was zusätzlich Zeit in Anspruch nimmt. Wenn natürlich nur die Groß- und Kleinbuchstaben für eine Nachricht verwendet werden, gibt es deutlich weniger Möglichkeiten: $2^{60} \approx 1,15 \cdot 10^{18}$, da nur noch 6- oder 7-Bit-ASCII-Zahlen in Betracht kommen. Daher ist es wichtig, für dem Additonswert n eine Zahl größer eins zu wählen, damit diese nicht so leicht erraten werden kann. Außerdem sollten Nachrichten länger als 33 Zeichen sein, denn ab dieser Länge dauert ein solcher Angriff mit einem guten Rechner länger als ein Jahr. Um solche Angriffe zu verhindern, sollten mit dieser Verschlüsselungstechnik keine Nachrichten mit weniger als 33 Zeichen versendet werden. Es kann daher auch notwendig sein, am Ende der Nachricht beliebige Zeichen zu ergänzen, um die Sicherheit zu verbessern.

4.3 Häufigkeitsanalyse zur Klartext-Bestimmung

Die Häufigkeitsanalyse ist besonders bei Substitutionsverfahren eine effiziente Angriffsart, um einen Geheimtext zu entschlüsseln. Dabei wird die Häufigkeit von Geheimtextzeichen bestimmt und mit der Häufigkeit von Buchstaben aus der Klartextsprache verglichen. Dadurch können Teile der oder die gesamte Substitutionstabelle bestimmt werden. Diese beschreibt, welches Zeichen des Klartextes auf welches Geheimtextzeichen abgebildet wird. Bei Buchstaben, die ähnlich oft in einer Sprache vorkommen, wird es schwieriger, diese genau zuzuordnen. Daher kann es nützlich sein, zusätzlich noch Bi-

und Trigramme zu verwenden. Dabei werden immer zwei oder drei Geheimtextzeichen zusammen betrachtet und mit der Häufigkeit von zwei oder drei aufeinanderfolgenden Zeichen der Klartextsprache verglichen. [16, S. 12]

Um einen mit Bit-Shift verschlüsselten Geheimtext mittels Häufigkeitsanalyse zu entschlüsseln, wird die Dezimaldarstellung der geheimen Nachricht untersucht, denn die Base64-Umwandlung ist öffentlich bekannt und kann somit jederzeit rückgängig gemacht werden. Es werden also die Häufigkeiten der Dezimalzahlen mit denen der Buchstaben aus der Klartextsprache verglichen.

Um die theoretische Erklärung dieser Angriffsmethode verständlicher zu machen, wird im Anschluss ein Geheimtext beispielhaft mit Hilfe der Häufigkeitsanalyse entschlüsselt.

Da beim Verschlüsseln der Nachrichten der Schlüssel nach jedem Zeichen umgedreht wird, muss der Geheimtext in zwei Gruppen unterteilt werden. Die erste Gruppe wurde mit dem Originalschlüsselwort verschlüsselt und die andere Gruppe wurde mit dem umgedrehten Schlüsselwort verschlüsselt. Die erste Gruppe beinhaltet also alle Dezimalzahlen, die an ungerader Stelle im Array stehen, wobei das Array mit dem Index eins beginnt. In der zweiten Gruppe befinden sich alle Zahlen die an gerader Stelle im Array stehen. Auf beide Gruppen wird separat die Häufigkeitsanalyse angewandt, um die Substitutionsvorschrift zu bestimmen, welche später auf alle Dezimalzahlen angewendet werden kann. Dafür wird die prozentuale Häufigkeit jeder Dezimalzahl innerhalb der Gruppe bestimmt und im Anschluss mit den Häufigkeiten der Klartextsprache verglichen. In deutschsprachigen Nachrichten kommen das Leerzeichen und der Buchstabe „e“ am häufigsten vor. [3, S. 304] Das heißt, die zwei am häufigsten vorkommenden Dezimalzahlen einer Geheimtextgruppe repräsentieren vermutlich jeweils eines dieser beiden Zeichen. Da es nur zwei Möglichkeiten gibt diese Zeichen zuzuordnen, kann die Richtige durch Ausprobieren herausgefunden werden. Wurden die beiden Dezimalzahlen den Zeichen zugewiesen, kann nun die Substitutionsvorschrift zwischen der ursprünglichen Binärzahl und der nach dem Bit-Shift herausgefunden werden. Dafür wird die Dezimalzahl in die zugehörige Binärzahl umgewandelt und mit der Binärzahl des zugeordneten Zeichens verglichen. Die beiden Zahlen unterscheiden sich dabei durch einen Bit-Shift und einen bestimmten Subtraktionswert.

Um die Anzahl der Bit-Shift-Stellen zu bestimmen, wird die Stellenanzahl der Binärzahl des zugeordneten Zeichens von der Anzahl an Bits der Geheimtext-Binärzahl subtrahiert. Anschließend führt man den Bit-Shift um diese Stellenzahl für die Binärzahl durch und subtrahiert davon die Binärzahl des zugeordneten Zeichens. Das Ergebnis ist der Subtraktionswert. Diese beiden Werte bilden die folgende Substitutionsvorschrift:

$$\text{Binärzahl des Klartextzeichens} = (\text{Geheimtextbinärzahl} >> \text{Shift-Wert}) - \text{Subtraktionswert}$$

Dieses Prinzip funktioniert allerdings nur, wenn der für die Verschlüsselung gewählte Additionswert n nicht die Stellenzahl der Binärzahl erhöht. Denn durch die Addition kann es vorkommen, dass aus einer 7-Bit-Zahl allein durch den Übertrag, also ohne einen Shift, eine 8-Bit-Zahl entsteht. Durch diese Veränderung der Stelligkeit außerhalb der Shift-Operation kann kein allgemeingültiger Shift-Wert mehr bestimmt werden.

Hat man diese beiden Werte bestimmt, kann jede Dezimalzahl des Geheimtextes mit dieser Substitutionsvorschrift entschlüsselt und mit Hilfe der ASCII-Tabelle dem zugehörigen Zeichen zugeordnet werden. Wurden alle Zahlen entschlüsselt, muss am Ende noch überprüft werden, ob diese einen sinnvollen Klartext ergeben. Ansonsten muss die Zuordnung der häufigsten Zeichen getauscht und alle Schritte erneut durchgeführt werden.

Konnte die Substitutionsvorschrift nicht eindeutig bestimmt werden, so kann dennoch durch die prozentuale Häufigkeit eine Zuordnung der Binärzahlen des Geheimtextes mit denen des Klartextes hergestellt werden. Mit Hilfe der Häufigkeitsanalyse kann man Teile der Substitutionstabelle herausfinden und diese dann nutzten, um Wörter aus dem Klartext zu erraten und somit weitere Zuordnungen in der Tabelle zu erhalten.

Wurden beispielsweise die häufigsten Buchstaben e, n, i, s und t schon bestimmt, so können bei folgendem Satz einige der fehlenden Buchstaben aus dem Zusammenhang erschlossen werden. „I_ _ esse _et_t ein Eis.“
Das erste Wort ist hierbei höchstwahrscheinlich „Ich“, wodurch schon zwei zusätzliche Buchstaben erraten werden konnten. Durch den Zusammenhang können auch beim dritten Wort schnell die Buchstaben „j“ und „z“ herausgefunden werden. Die herausgefundenen Buchstaben können dann den Geheimtext-Binärzahlen zugeordnet und die Substitutionstabelle weiter gefüllt werden. Durch immer mehr richtige Buchstaben in dieser Tabelle wird es leichter weitere Wörter aus dem Zusammenhang zu erschließen, wodurch sich irgendwann der gesamte Klartext herausfinden lässt.

Beispiel 4.1 In diesem Beispiel wird ein Geheimtext mit Hilfe der Häufigkeitsanalyse entschlüsselt. Der Klartext wurde in Deutsch verfasst und besteht aus 1.000 Zeichen. Aus Platzgründen werden hier nur die für das Verständnis wichtigen Informationen dargestellt, das Prinzip lässt sich aber einfach auf alle weggelassenen Informationen übertragen. Mit Hilfe der Base64-Decodierung kann der Geheimtext im ersten Schritt in das zugehörige Dezimalzahlenarray umgewandelt werden. Bevor aber mit der eigentlichen Häufigkeitsanalyse begonnen werden kann, müssen die Zahlen des Arrays in zwei Gruppen aufgeteilt werden, da beim Verschlüsseln das Schlüsselwort nach jedem Zeichen umgedreht und die Nachricht daher eigentlich mit zwei Schlüsselwörtern verschlüsselt wird. In der ersten Gruppe werden alle Dezimalzahlen des Arrays gespeichert, die an ungerader Stelle stehen. In der zweiten Gruppe alle anderen Zahlen, also alle an gerader Stelle im Array.

Geheimtextarray:
40349766160, 54795584648, 59140248080, 52648101000, 59677118992, 54258713736, 17801187856, 59090551944, 63435215376, 62311777416, 62361473552, 62848648328, 54845280784, 17751491720, ...

Gruppe 1:		**Gruppe 2:**	
40349766160,	59140248080,	54795584648,	52648101000,
59677118992,	17801187856,	54258713736,	59090551944,
63435215376,	62361473552,	62311777416,	62848648328,
54845280784, ...		17751491720, ...	

Die Häufigkeitsanalyse wird zuerst mit einer der beiden Gruppen durchgeführt und das Ergebnis danach auf die andere Gruppe übertragen. In anderen Fällen, in denen die Substitutionsvorschrift nur schwer herausgefunden werden kann, ist es sinnvoll die Häufigkeitsanalyse auf beide Gruppen parallel anzuwenden. In diesem Beispiel wurde mit der ersten Gruppe begonnen. Die zweite Gruppe musste anschließend nicht mehr untersucht werden, da der Geheimtext mit der ersten Gruppe bereits entschlüsselt werden konnte. Für die Entschlüsselung mit Hilfe der Häufigkeitsanalyse wird die prozentuale Häufigkeit der Dezimalzahlen innerhalb der Gruppe bestimmt. Die sechs häufigsten Zahlen sind dabei:

Dezimalzahlen	Anzahl in der 1. Gruppe	Häufigkeit in Prozent
17801187856	77	15%
54845280784	75	15%
59677118992	52	10%
61824602640	28	6%
62898344464	27	5%
62361473552	25	5%

Diese werden nun mit den häufigsten Buchstaben der Klartextsprache verglichen. Auffällig ist in diesem Beispiel, dass zwei Dezimalzahlen mit 15% sehr oft vorkommen. Allerdings kommt in der deutschen Sprache der Buchstabe „e“ mit 17% am häufigsten vor. Mit großem Abstand, mit 9%, folgt „n“. [3, S. 304] Eine der beiden Dezimalzahlen ist also kein Buchstabe, sondern das Leerzeichen, denn dieses kommt in Textnachrichten noch häufiger vor als das „e“. Da hier die Zahlen, aufgrund des geringen Unterschiedes, nicht genau einem Zeichen zugeordnet werden können, muss die richtige Zuordnung durch Ausprobieren herausgefunden werden. In diesem Beispiel wurde der Zahl „17801187856“ das Leerzeichen zugeordnet, weil diese tatsächlich am häufigsten vorkommt. Nach dem Zuordnen kann die ASCII-Binärzahl des Leerzeichens mit der Binärzahlendarstellung der zugeordneten Dezimalzahl verglichen werden. Dabei kann meist ein allgemeingültiger Unterschied festgestellt werden, welcher auf den gesamten Geheimtext angewendet

werden kann, um die zugehörigen Klartextzeichen herauszufinden. Für die Dezimalzahl „17801187856“ ist der Unterschied, wie hier dargestellt, ein Shift um 29 und anschließend 1 subtrahiert:

$$\begin{aligned} 1000010010100001000100100100001000 &>> 29 = 100001 \\ 100001 &- 1 = 100000 \end{aligned}$$

Diese Rechnung kann nun für alle anderen Binärzahlen beider Gruppen durchgeführt werden, um den folgenden Klartext zu erhalten:

„Jemand musste Josef K. verleumdet haben, denn ohne dass er etwas Böses getan hätte, wurde er eines Morgens verhaftet...“ (vollständiger Klartext im Anhang A)

Wie hier zu sehen, wurden die häufigsten Zeichen richtig zugeordnet, da der Klartext eine sinnvolle Nachricht ergibt. Sollte dies nicht der Fall sein, so kann eine andere Zuordnung der häufigsten Zeichen helfen den richtigen Klartext zu finden.

Diese Methode zum Entschlüsseln eines Geheimtextes ist nur möglich, wenn dem Angreifer die Sprache des Klartextes schon bekannt ist. Ob es auch möglich ist, die Klartextsprache anhand bestimmter Merkmale im Geheimtext herauszufinden, wird im Folgenden beispielhaft überprüft. Dazu wurden mehrere Geheimtexte in deutscher und englischer Sprache untersucht. Dabei konnte festgestellt werden, dass die häufigste Dezimalzahl aus englischen Geheimtexten öfter vorkommt als die häufigste Dezimalzahl in deutschen Geheimtexten.

Betrachtet man die häufigsten Zeichen beider Sprachen, so lässt sich feststellen, dass in beiden Sprachen das Leerzeichen am häufigsten vorkommt. Dabei unterscheidet sich die prozentuale Häufigkeit um knapp 3%. Im Englischen kommt das Leerzeichen mit ungefähr 19% [18] häufiger vor als im Deutschen mit ungefähr 16% [4, S. 15]. Dieser Unterschied kann genutzt werden, um mit Hilfe der Häufigkeitsanalyse die Sprache des Klartextes zu bestimmen. Durch den geringen Unterschied ist es nicht immer möglich das häufigste Zeichen genau einer Sprache zuzuordnen. In diesem Fall wird die zweithäufigste Dezimalzahl als Vergleichswert genutzt. Denn die zweithäufigste Dezimalzahl entspricht dem Buchstaben „e“. Dieser kommt im Deutschen mit ungefähr 17% deutlich häufiger vor als im Englischen mit nur 12%. [3, S. 304]

Beispiel 4.2 Werden zwei Geheimtexte in Deutsch und Englisch mit jeweils 1.000 Zeichen verglichen, so ist der Unterschied der häufigsten Dezimalzahl in einer Schüsselgruppe zwischen den beiden Sprachen hier knapp 3%. Im englischen Text kommt die Zahl mit 19% und im Deutschen mit 16% am häufigsten vor. Wird auch noch die zweithäufigste Dezimalzahl betrachtet, welche dem Buchstaben „e“ entspricht, so kann man hier feststellen, dass die prozentuale Häufigkeit mit 14% beim deutschen und 10% beim englischen Geheimtext nicht ganz mit denen der allgemeinen Häufigkeitsverteilung beider

Sprachen übereinstimmt. Dies liegt daran, dass die Anzahl der Leerzeichen in den 500 Zeichen der jeweiligen Schlüsselgruppe mit enthalten ist. Wird die prozentuale Häufigkeit des zweithäufigsten Zeichens ohne das Leerzeichen bestimmt, so betragen diese 17% beim deutschen und 12% beim englischen Geheimtext. Diese stimmen mit der allgemeinen Häufigkeitsverteilung überein und es kann daher mit der Häufigkeitsverteilung der Dezimalzahlen auf die verwendete Sprache rückgeschlossen werden.

Hier wurden jetzt nur die Sprachen Deutsch und Englisch betrachtet, ob sich dieses Prinzip auch auf andere Sprachen übertragen lässt, konnte im Rahmen dieser Arbeit nicht überprüft werden.
Um also einen Geheimtext der deutschen oder englischen Sprache mit Hilfe der Häufigkeitsanalyse zu entschlüsseln, bestimmt man zuerst die prozentuale Häufigkeit aller Dezimalzahlen des Geheimtextes. Mit den zwei häufigsten Dezimalzahlen kann dann die Klartextsprache zugeordnet und anschließend die häufigsten Buchstaben dieser Sprache den häufigsten Dezimalzahlen zugeordnet werden.

4.4 Neues Verfahren zur Schlüsselbestimmung

Bei dem Bit-Shift-Verschlüsselungsverfahren ist es nicht notwendig genau den verwendeten Schlüssel zur Geheimtextentschlüsselung herauszufinden. Es genügt einen gleichwertigen Schlüssel zu finden, der die gleichen Shift-Werte generiert und denselben Additionswert n hat.
Um einen gleichwertigen Schlüssel zu finden, wird zuerst die Häufigkeitsanalyse durchgeführt. Mit Hilfe des dort herausgefundenen Parameters n werden nun die Geheimtext-Binärzahlen auf Gleichheit analysiert. Dabei ist der hintere Teil aller Zahlen immer gleich und nur die ersten sechs bis acht Stellen, bei $n > 255$ auch mehr Stellen, sind unterschiedlich. Diese Stellen repräsentieren das verschlüsselte Klartextzeichen. Für das Herausfinden eines gleichwertigen Schlüssels, wie dem der zum Verschlüsseln verwendet wurde, ist nur der hintere, gleichbleibende Teil der Binärzahlen entscheidend. Waren beim Verschlüsseln die durchgeführten Bit-Shifts immer größer wie die Stelligkeit von n, so kann der Additionswert in dem gleichbleibenden Teil der Binärzahlen, umschlossen von 0er-Blöcken, gefunden werden. Von links beginnend, werden nun die Stellen des gleichbleibenden Binärteils bis einschließlich der Stellen des ersten Additionswertes gezählt. So erhält man den ersten Shift-Wert. Werden alle weiteren Stellen bis einschließlich des jeweils nächsten Additionswertes genauso gezählt, erhält man alle Shift-Werte des verwendeten Schlüssels. Für den letzten Shift-Wert wird nur die Anzahl der vorkommenden Nullen gezählt, da am Ende des Verschlüsselns nur ein Bit-Shift und keine Addition mehr durchgeführt wird.
Der maximale Shift-Wert b wird anschließend genau um eine Zahl größer als der größte herausgefundene Shift-Wert festgelegt, um alle benötigten Shift-Werte durch die Modulo-Rechnung erhalten zu können. Für das Schlüsselwort werden jetzt passende ASCII-

Zeichen gewählt, die Modulo b gerechnet, die zuvor herausgefundenen Shift-Werte ergeben. Mit dem so herausgefundenen Schlüssel kann der gesamte Geheimtext entschlüsselt werden.

Ob der Schlüssel auch herausgefunden werden kann, wenn der Shift-Wert beim Verschlüsseln nicht immer größer als die Stelligkeit von n ist, konnte im zeitlichen Rahmen dieser Arbeit nicht weiter untersucht werden.

Beispiel 4.3 Die Geheimtext-Binärzahl `1100110001010000100010010010000010000` aus dem Beispiel 4.1 konnte mit Hilfe der Häufigkeitsanalyse dem Klartextzeichen „e“ zugeordnet und so auch der Additionswert $n = 1$ bestimmt werden. Mit diesen Informationen kann jetzt ein Schlüssel gefunden werden, mit welchem sich der gesamte Geheimtext entschlüsseln lässt. Dafür wird das Vorkommen des Wertes n in der Geheimtext-Binärzahl untersucht:

`1100110`|`001`|`01`|`00001`|`0001`|`001`|`001`|`00001`|`0000`

Die ersten 7-Bit-Stellen repräsentieren das Klartextzeichen und sind deshalb nicht zu beachten. Werden danach alle Stellen bis einschließlich der des ersten Additionswertes gezählt, erhält man für den ersten Shift-Wert die Zahl 3. Wird dieses Prinzip so fortgeführt und zählt man am Ende der Binärzahl noch die übrigen Nullen, erhält man folgendes Array mit allen Shift-Werten: $[3, 2, 5, 4, 3, 3, 5, 4]$. Anschließend wird der maximale Shift-Wert $b = 6$ gesetzt und damit für jeden Shift-Wert ein passendes ASCII-Zeichen bestimmt. In diesem Fall wurde für den ersten Shift-Wert der Buchstabe „E“ gewählt, da dieser Buchstabe in ASCII durch die Zahl 69 repräsentiert wird und Modulo 6 gerechnet die Zahl 3 ergibt. Hat man so das gesamte Schlüsselwort bestimmt, erhält man den Schlüssel: („EDGFEEGF“, 6, 1). Mit diesem kann jetzt der gesamte Geheimtext entschlüsselt werden. Ursprünglich wurde die Nachricht eigentlich mit dem Schlüssel: („Schlüssel“, 8, 1) verschlüsselt. Es zeigt sich also, dass der gleiche Geheimtext durch verschiedene Schlüssel erzeugt werden kann.

Bemerkung 4.4 Abschließend ist noch anzumerken, dass das Bit-Shift-Verfahren eine Amateur-Verschlüsselung ist, die nur aus didaktischen Gründen erstellt wurde. Wie im Abschnitt 4.4 zu sehen, kann bei der Bit-Shift-Verschlüsselung nicht nur der Klartext, sondern auch ein zugehöriger Schlüssel herausgefunden werden. Dieses Verfahren ist daher nicht sicher und sollte auch nicht für geschützte Kommunikation verwendet werden. Möchte man dennoch ein symmetrisches Verschlüsselungsverfahren nutzen, so sollte dafür auf ein modernes Verfahren, wie Advanced Encryption Standard (kurz AES), zurückgegriffen werden.

5 Webapp

Um dem Benutzer eine attraktive und zeitgemäße Website zu bieten, wird das bisherige CrypTool-Online gerade auf eine verbesserte Benutzeroberfläche umgestellt. Dafür wird primär React und die dazu passende Chakra UI-Bibliothek [10] verwendet. React stellt die Website nicht nur auf allen Geräten optimal dar, sondern ermöglicht es auch, die Seiten schnell zu laden und während des Renderns reaktionsfähig zu halten. Durch die ständige Weiterentwicklung von React bleibt die entwickelte Web-App, hinsichtlich der Browser-Darstellung, immer auf dem neuesten Stand. [17]

5.1 Status Quo

Die alte Bit-Shift-App in dem bisherigen CrypTool-Online besteht aus zwei Reitern. Auf dem einen wird eine kurze Erklärung des Bit-Shift-Verfahrens angeboten, auf dem anderen die Verschlüsselungstechnik selbst. Die Nutzeroberfläche besteht in der bisherigen Bit-Shift-App nur aus wenigen Feldern: Einem Eingabe- und einem Ausgabetextfeld, über die entweder der Klartext oder der Geheimtext eingegeben werden kann. Darunter liegt das Eingabefeld für den Schlüssel. Hier kann der Schlüssel zum Ver- und Entschlüsseln eingegeben werden. Am unteren Rand der App ist der Bereich für Optionen. Zwischen den beiden Textfeldern gibt es einen Pfeil, der die Richtung des Verfahrens angibt. Die Pfeilrichtung kann manuell durch Anklicken geändert werden. Zeigt der Pfeil vom Klartext zum Geheimtext, wird verschlüsselt, zeigt er in die andere Richtung, wird entschlüsselt. Im Bereich für Optionen befindet sich lediglich eine Checkbox, mit der die verschlüsselte Nachricht in Fünfer-Blöcken dargestellt werden kann.

Wie in Abbildung 5.1 zu sehen, ist die Option „5er Blöcke“ aktiviert, aber die verschlüsselte Nachricht ist nicht in Fünfer-Blöcke unterteilt. Dies liegt daran, dass diese Option erst dann wirksam wird, wenn die eingegebene Nachricht verändert wurde. Gleiches gilt auch für das Ausschalten der Fünfer-Blöcke-Darstellung und das Ändern der Pfeilrichtung. Durch das Verwenden von React können diese Veränderungen automatisch erkannt und umgesetzt werden.
In dem zweiten Reiter der Website, wird die Funktionsweise des Bit-Shift-Verfahrens genauer erläutert, siehe Abbildung 5.2. Zu Beginn wird nur kurz die Funktionsweise der Bit-Shift-Verschlüsselung aufgezeigt und anschließend gibt es jeweils ein knappes

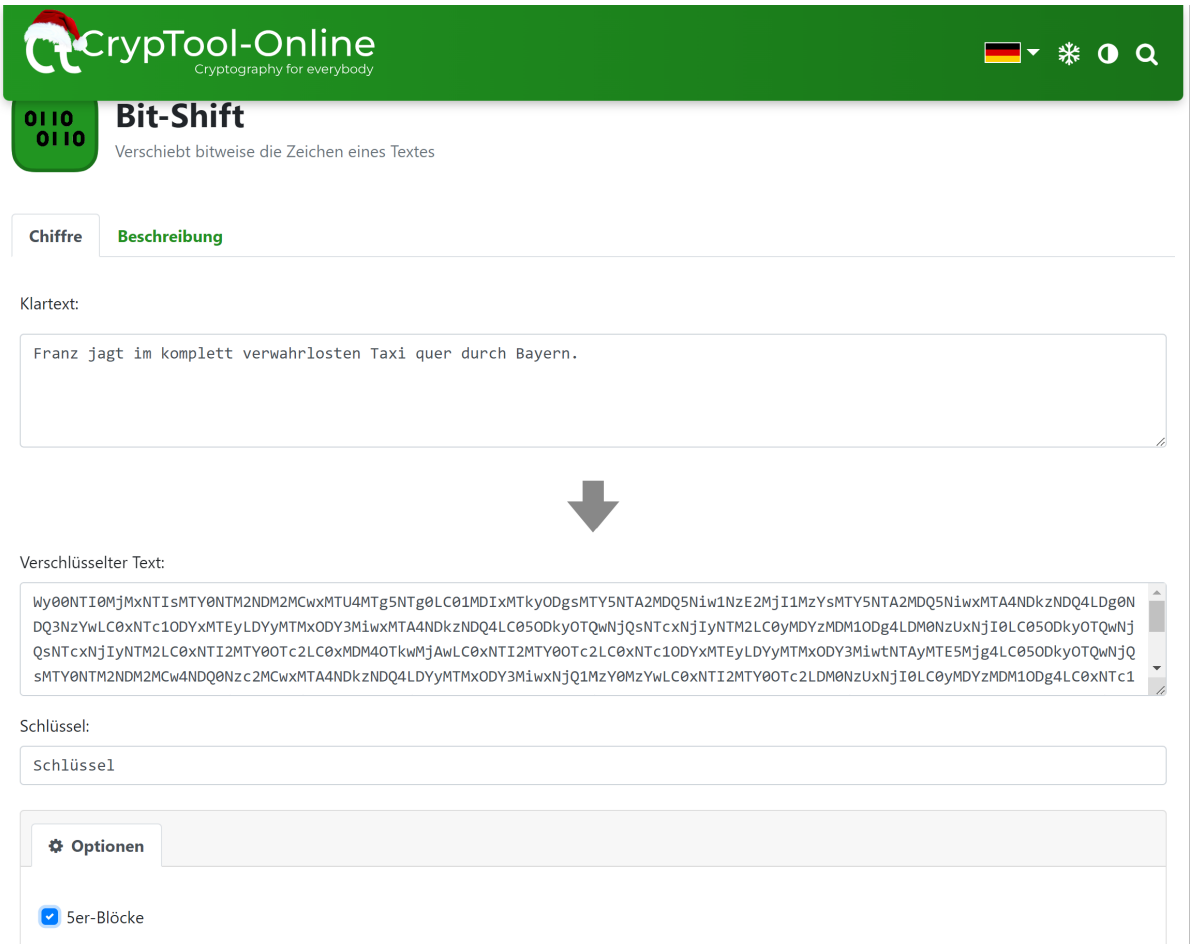
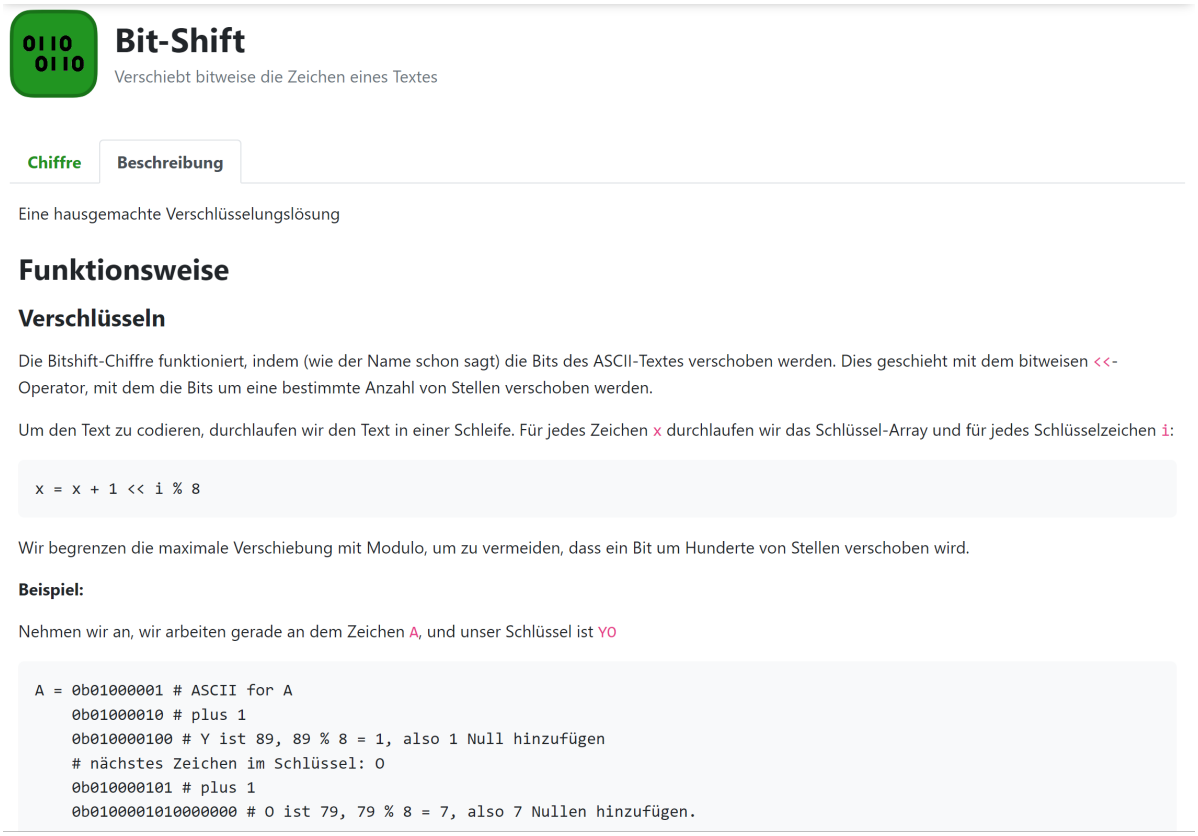


Abbildung 5.1: Alte Bit-Shift-App



Bit-Shift

Verschiebt bitweise die Zeichen eines Textes

Chiffre | Beschreibung

Eine hausgemachte Verschlüsselungslösung

Funktionsweise

Verschlüsseln

Die Bitshift-Chiffre funktioniert, indem (wie der Name schon sagt) die Bits des ASCII-Textes verschoben werden. Dies geschieht mit dem bitweisen `<<`-Operator, mit dem die Bits um eine bestimmte Anzahl von Stellen verschoben werden.

Um den Text zu codieren, durchlaufen wir den Text in einer Schleife. Für jedes Zeichen `x` durchlaufen wir das Schlüssel-Array und für jedes Schlüsselzeichen `i`:

```
x = x + 1 << i % 8
```

Wir begrenzen die maximale Verschiebung mit Modulo, um zu vermeiden, dass ein Bit um Hunderte von Stellen verschoben wird.

Beispiel:

Nehmen wir an, wir arbeiten gerade an dem Zeichen `A`, und unser Schlüssel ist `YO`

```
A = 0b01000001 # ASCII for A
    0b01000010 # plus 1
    0b010000100 # Y ist 89, 89 % 8 = 1, also 1 Null hinzufügen
    # nächstes Zeichen im Schlüssel: O
    0b010000101 # plus 1
    0b0100001010000000 # O ist 79, 79 % 8 = 7, also 7 Nullen hinzufügen.
```

Abbildung 5.2: Alte Bit-Shift-Beschreibung

Beispiel zur Ver- und Entschlüsselung. Beide Beispiele sind nicht sehr verständlich beschrieben, weshalb die Technik unter Umständen auch nach mehrmaligem Lesen nicht vollständig verstanden werden kann.

5.2 Implementierung

Die neue Bit-Shift-App wird, wie man in der Abbildung 5.3 sehen kann, in ein schon vorhandenes Projekt von CrypTool-Online eingebettet. Die App an sich wurde dabei auf mehrere Dateien aufgeteilt. Die BitShiftAlgorithm.js-Datei berechnet die Ver- und Entschlüsselung mit den zugehörigen Zwischenschritten. Diese Werte werden in der Decryption.jsx- und Encryption.jsx-Datei verarbeitet und eine Webkomponente erstellt, welche an die BitShiftComponent.jsx-Datei zurückgegeben wird. Diese Datei fasst am Ende alle Webkomponenten zu einer zusammenhängenden Website zusammen. Zudem verwaltet sie die Werte, die vom Nutzer auf der Website eingegeben werden können. Die KeyInputValue.jsx-Datei kümmert sich um die Darstellung der Schlüsselparametereingabe inklusive der Verarbeitung eventuell auftretender Fehlermeldungen. Damit die in Abbildung 5.6 dargestellte Zwischenschrittauswahl nicht mehrmals implementiert werden musste, wurde diese in die SliderComponent.jsx-Datei ausgelagert.
Für die deutsche und englische Version der App gibt es in allen jsx-Dateien eine Translations-Variable, die anhand der übergebenen Sprache den Anzeigetext auf Deutsch oder Englisch zurückgibt. Für beide Sprachen ist zusätzlich jeweils die passende readme-Datei vorhanden, welche weitere Informationen bezüglich der Bit-Shift-Verschlüsselung enthält.

5.2.1 Gestaltung

Statt nur die Ver- und Entschlüsselung mit dem entsprechenden Output auszuführen zeigt die neue Bit-Shift-App das Verfahren schrittweise an und bietet deutlich mehr Optionen. Wie in der Abbildung 5.4 zu sehen, wird die Eingabe der Schlüsselparameter an den Anfang gestellt, da diese immer als Erstes vor dem Ver- und Entschlüsseln einer Nachricht definiert werden müssen.

Damit die Ver- und Entschlüsselung genauso einfach funktioniert, wie früher das Klicken auf den Pfeil, wird die Ver- und Entschlüsselung in zwei separaten Registerkarten angezeigt, zwischen denen der Nutzer durch Anklicken hin und her wechseln kann, siehe Abbildung 5.4. Damit der Benutzer den generierten Geheimtext nicht händisch in das Entschlüsselungsregister kopieren muss, wird dieser automatisch in die Geheimtexteingabe der Entschlüsselungsbeschreibung kopiert. Die useMemo()-Methode von React überprüft, ob sich der verschlüsselte Text im Verschlüsselungsreiter geändert hat und kopiert

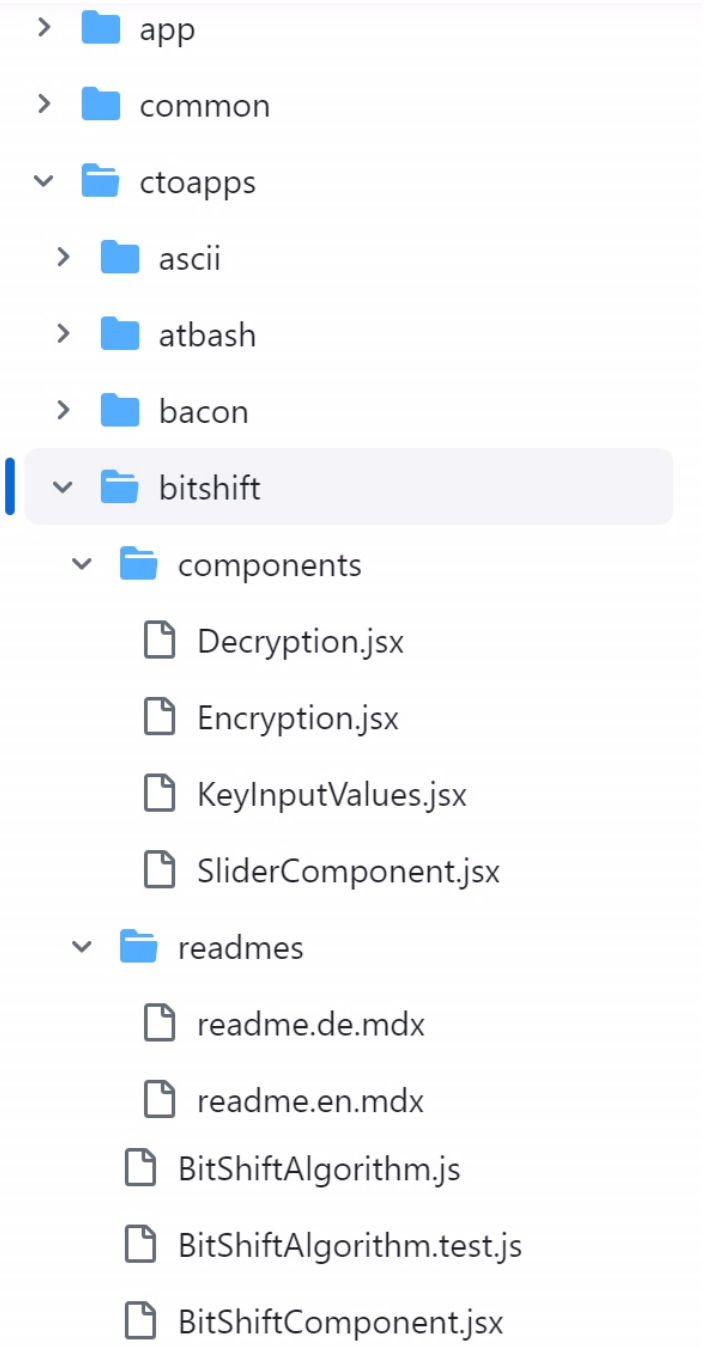


Abbildung 5.3: Projektstruktur

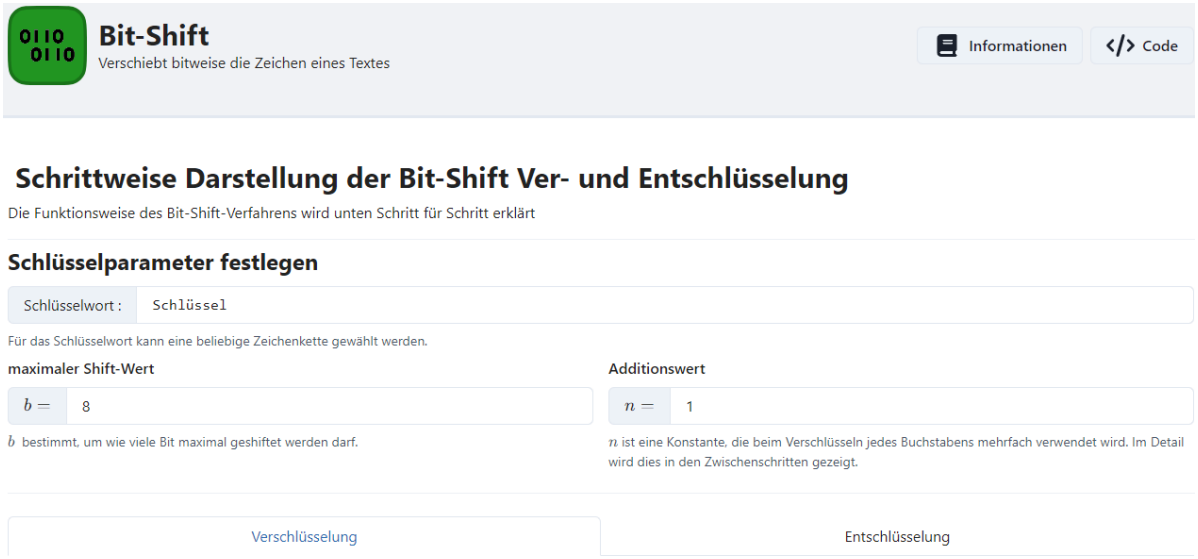


Abbildung 5.4: Neue Bit-Shift-App

```
1         useMemo(() =>
2         setCiphertext(encryptedString), [encryptedString])
```

Listing 5.1: Geheimtext automatisch setzen

diesen anschließend über die setCiphertext()-Methode in den Geheimtext des Entschlüsselungsbereiches, siehe Listing 5.1.

Damit der Benutzer das Verfahren einfacher nachvollziehen kann, werden nicht mehr nur der Klar- und Geheimtext angezeigt, sondern auch die in Kapitel 3 beschriebenen Zwischenschritte. Die Bit-Shift-Verschlüsselung kann so direkt bei der Anwendung verstanden werden, wodurch keine zusätzliche Anleitung notwendig ist. Bei längeren Nachrichten und großen Shift-Werten würde die neue Website dadurch jedoch schnell unübersichtlich. Um dies zu verhindern, werden die einzelnen Zwischenschritte, wie in Abbildung 5.5 zu sehen, durch ausklappbare Website-Inhalte realisiert. Jeder Nutzer kann so selbst entscheiden, ob er sich alle oder nur einzelne Zwischenschritte anzeigen lassen möchte. Mit der Accordion-Componente von Chakra UI kann dies auch einfach im Code umgesetzt werden. Wie im Listing 5.2 zu sehen, muss dafür nur der Zwischenschrittname, im AccordionButton und der zugehörige Inhalt, im AccordionPanel, deklariert werden.

Um es dem Nutzer zu ermöglichen, sich den Bit-Shift-Algorithmus für jedes Nachrichten- und Schlüsselzeichen anzeigen zu lassen, wurden zwei Zahleneingabefelder mit dazugehöriger Zahlenskala, wie in Abbildung 5.6a dargestellt, auf der Website hinzugefügt. Sowohl über die Zahleneingabe als auch die Zahlenskala kann der Nutzer eine gewünschte Schlüssel- und Nachrichtenstelle auswählen. In der darunterliegenden Ausgabe wird dann

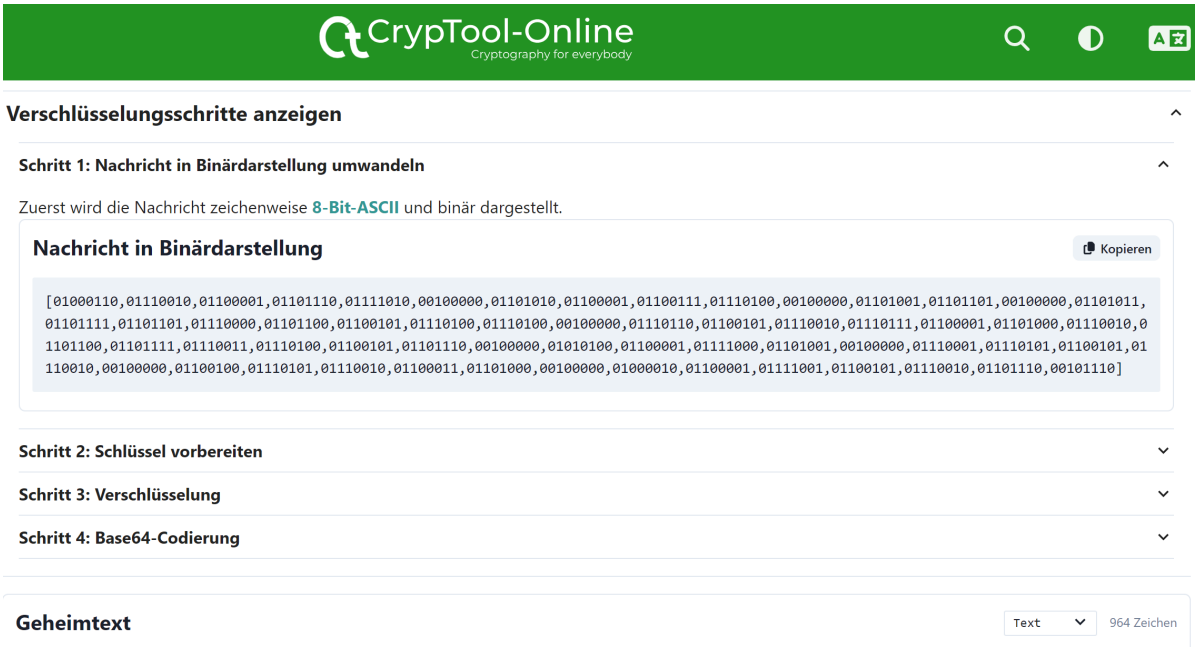


Abbildung 5.5: Zwischenschrittdarstellung Website

```
    <AccordionItem>
        <h2>
            <AccordionButton px={0}>
                <Box as="b" flex="1" textAlign="left">
                    {translation.step1Label}
                </Box>
                <AccordionIcon />
            </AccordionButton>
        </h2>
        <AccordionPanel px={0} pb={4}>
            {translation.step1Text}
            <CustomCodeBlock title={translation.step1CodblockTitle}>
                [{props.plainTextArrayBinary?.toString()}]
            </CustomCodeBlock>
        </AccordionPanel>
    </AccordionItem>

```

Listing 5.2: Zwischenschrittdarstellung Code

der Bit-Shift-Schritt mit dem passenden Klartext- und Schlüsselwortzeichen dargestellt. Das Zahleneingabefeld mit der nebenstehenden Zahlenskala gestaltet sich für Anwender mit kleinem Bildschirm oder Toucheingabe auf mobilen Endgeräten eher schwierig. Deshalb wurde für diese Fälle eine andere Eingabemöglichkeit gewählt. Wie in Abbildung 5.6b zu sehen, gibt es hier für die Auswahl der Stellen jeweils nur ein Nummerneingabefeld und zwei Buttons, um die Zahl zu erhöhen oder zu verringern. Der Wechsel zwischen diesen beiden Ansichten erfolgt anhand der Bildschirmgröße automatisch.

Darüberhinaus wurde die Website, wie in Abbildung 5.7 dargestellt, noch um eine Anzeige mit zusätzlichen Informationen aus den Kapiteln 3 und 4 ergänzt.

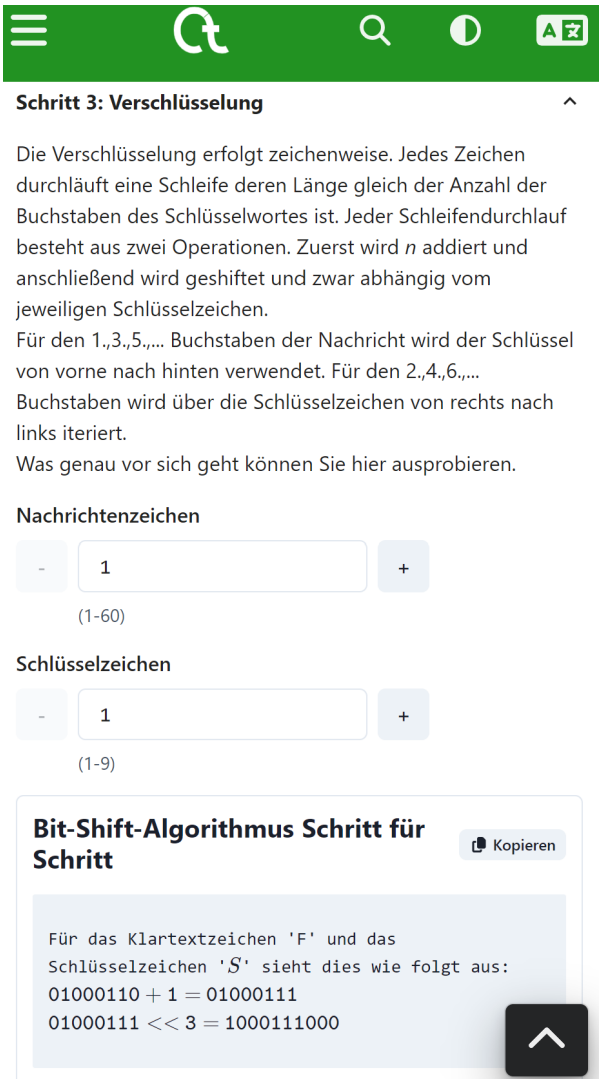
5.2.2 Funktionalität

Der implementierte Bit-Shift-Algorithmus aus der alten App ist wie in Listing 5.3 zu sehen sehr kompakt gehalten. Dadurch war es nicht möglich, diesen ohne vorherige Anpassung in der neuen App wiederzuverwenden, denn es gibt keine Ausgabe der Zwischenschritte, die jetzt für die Anzeige auf der neuen Website benötigt werden. Zudem war der Additionswert n mit 1 und der maximale Shift-Wert b mit 8 fest vorgegeben. In der neuen App kann der Nutzer diese Werte frei wählen (siehe Zeile 7 im Listing 5.3). Deshalb hat sich der Autor dazu entschieden, den Bit-Shift-Algorithmus neu zu implementieren. Im neuen Algorithmus werden alle Zwischenschritte in einem geschachtelten Array gespeichert. Dabei beinhaltet das innerste Array, im Code „bitKeyShiftArray“ genannt, die Werte vor der Addition, vor dem Bit-Shift und danach. Das „characterBitShiftArray“ beinhaltet alle Arrays für alle Schlüsselzeichen und das oberste Arrray („interimStepArray“) beinhaltet alle Zwischenschritte für alle Zeichen des Klartextes. Dies kann man im Listing 5.4 in den Zeilen 4, 7, 12, 14 und 18 sehen. Außerdem wird in Zeile 6 jetzt der variable Additionswert n, hier „increaseValueBig“ genannt, dazu addiert und nicht mehr die bisher feste Zahl eins.

Der alte Algorithmus verwendet in Zeile 5 im Listing 5.3 die parseInt()-Methode, welche einen Integer zurückgibt. Der Integer-Datentyp von JavaScript unterstützt nur Zahlen bis $2^{53} - 1$. [15] Durch den im Verfahren verwendeten Bit-Shift kann das Ergebnis aber schnell größer als die maximal möglichen 53 Bitstellen werden. Die Abbildung 5.8 zeigt die Entschlüsselung des Standardbeispiels der alten Bit-Shift-App und es wird deutlich, dass hier beim Verschlüsseln des Beispieltextes der Zahlenbereich übergelaufen ist, da im Klartextfeld kein sinnvoller Text ausgegeben wird.

Zur Behebung dieses Problems wird der neue Algorithmus mit BigInt-Variablen implementiert. Dieser Datentyp ermöglicht das Rechnen mit ganzen Zahlen beliebiger Größe. Dabei muss aber darauf geachtet werden, dass dieser Datentyp nicht mehr alle vordefinierten Funktionen unterstützt. Um diese zu umgehen, wurden für die CTO-Apps der Datentyp BigInt mit allen relevanten Funktionen, wie Division mit Rest, Addition,

(a) Desktop-Ansicht



(b) Mobile-Ansicht

Abbildung 5.6: Zwischenschrittauswahl im Schritt 3

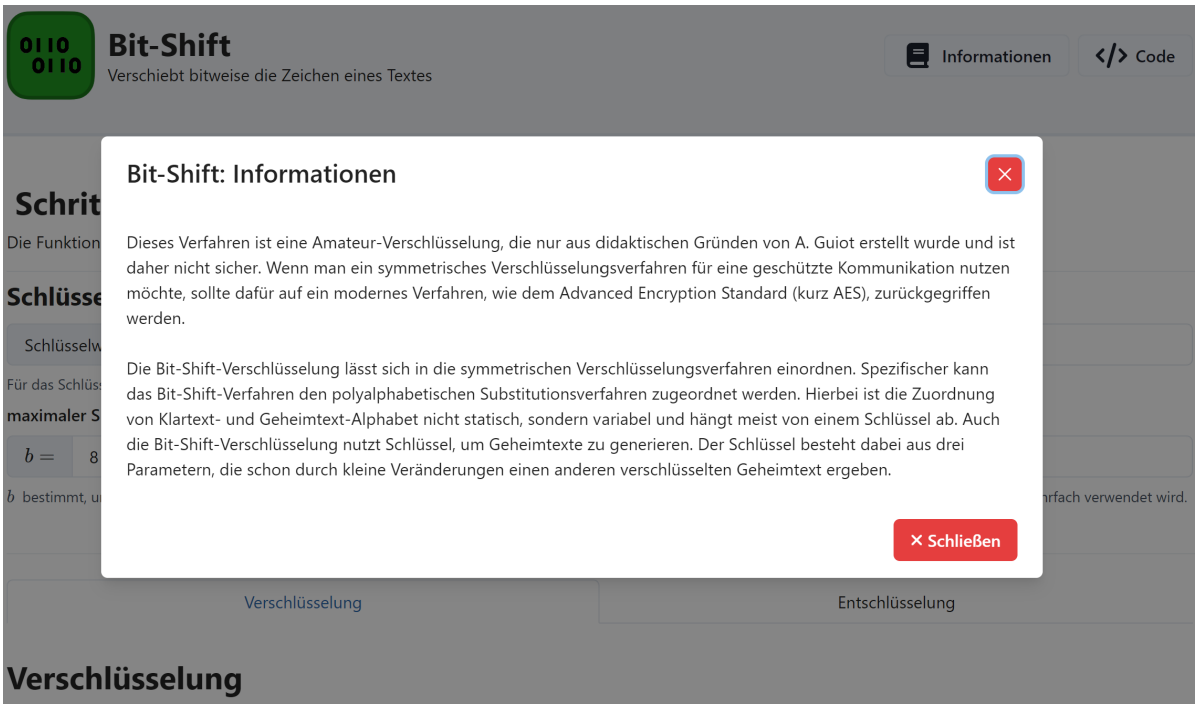


Abbildung 5.7: Anzeige weiterer Informationen

```
1   String.prototype.encrypt = function(key) {
2   const encoded = this.encode();
3   const keyEncoded = key.encode();
4   let array = encoded.map(x => {
5       x = parseInt(x)
6       for (let i of keyEncoded) {
7           x = x + 1 << i % 8
8       }
9       keyEncoded.reverse()
10      return x;
11  })
12  if (typeof btoa === 'undefined') {
13      global.btoa = str =>
14      new Buffer(str, 'binary').toString('base64');
15  }
16  return btoa(JSON.stringify(array))
17  }
```

Listing 5.3: Alter Bit-Shift-Algorithmus

```
1     import bigInt from "big-integer" // math functions on large numbers
2     function shift() {
3     for (let x = 0; x < plainTextArrayShiftedDezimal.length; x++) {
4       for (let i = 0; i < keyArrayModulo.length; i++) {
5         bitKeyShiftArray.push(plainTextArrayShiftedDezimal[x])
6         plainTextArrayShiftedDezimal[x] =
7           plainTextArrayShiftedDezimal[x].add(increaseValueBig)
8         bitKeyShiftArray.push(plainTextArrayShiftedDezimal[x])
9
10        plainTextArrayShiftedDezimal[x] =
11        plainTextArrayShiftedDezimal[x].shiftLeft(keyArrayModulo[i])
12        bitKeyShiftArray.push(plainTextArrayShiftedDezimal[x])
13
14        characterBitShiftArray.push(bitKeyShiftArray)
15        bitKeyShiftArray = []
16      }
17      keyArrayModulo.reverse()
18      interimStepArray.push(characterBitShiftArray)
19      characterBitShiftArray = []
20    }
21    }
```

Listing 5.4: Neuer Bit-Shift-Algorithmus

Abbildung 5.8: Fehler beim Entschlüsseln in alter App

```
1         <TeX>{latex`\mathtt{${interimStepArrayEncrypt?.[
2             sliderValueString - 1]?.[sliderValueKey - 1]?.[0]
3             .toString(2)
4             .padStart(8, "0")} +
5             ${increaseValue} = \:}
6             `}</TeX>
```

Listing 5.5: Binärzahlendarstellung der Zwischenschritte auf der Website

Subtraktion und Bit-Shift, im Projekt als Interface schon vordefiniert (siehe Listing 5.4).

Die im Algorithmus berechneten Zahlen für die Anzeige der Zwischenschritte auf der Website sind daher auch BigInt-Werte. Deshalb werden diese Zahlen vor dem Anzeigen auf der Website mit der toString()-Methode in einen Binärstring umgewandelt. Diese Methode gibt allerdings keine führenden Nullen mit an. Da für das Bit-Shift-Verfahren aber die 8-Bit-ASCII-Darstellung gewählt wurde, sollten die Binärzahlen auch führende Nullen haben. Dies wird mit der padStart()-Methode im Listing 5.5 in Zeile 4 realisiert. Diese Methode fügt allen Binärzahlen, kleiner acht Stellen, die entsprechende Anzahl an führenden Nullen hinzu. Zudem werden im Listing 5.5 zwei nicht allgemein bekannte Zeichen verwendet, das „\:“ und „?“. Das „\:“-Zeichen erzeugt ein Leerzeichen nach dem „=“ und das „?“ verhindert einen Absturz der Website bei undefiniertem „interimStepArrayEncrypt“.

Die Bit-Shift-Verschlüsselung erlaubt als Nachricht und Schlüsselwort nur 8-Bit-ASCII-Zeichen. Deshalb wird die Eingabe dieser beiden Felder auf der Website beschränkt. Die eingegebene Zeichenkette wird nach der Eingabe mit der im Listing 5.6 beschriebenen Methode auf 8-Bit-ASCII-Zeichen geprüft. Damit es hierbei nicht zu unerwarteten Fehlern kommt, wird die Überprüfung mit der codePointAt()-Methode, welche alle Unicode-Zeichen in UTF-16-Codierung umwandeln kann, durchgeführt. Diese Funktion gibt für die ASCII-Zeichen genau die gleichen Dezimalzahlen wie bei der ASCII-Codierung zurück. Im Anschluss überprüft eine If-Abfrage in einer Schleife, ob die Dezimalzahlen ASCII-konform sind und sich daher im Zahlenbereich zwischen 0 und 256 befinden. Der Rückgabewert entscheidet dann darüber, ob die Zeicheneingabe des Nutzers akzeptiert, die Berechnung des Geheimtextes durchgeführt oder stattdessen eine Fehlermeldung ausgegeben wird.

Das Gleiche gilt auch für die Eingabe des Geheimtextes, dabei wird dieser aber nicht direkt überprüft, sondern bei der Berechnung des Klartextes wird ein entsprechender Fehler ausgegeben und die Berechnung abgebrochen. Die Fehlermeldung wird dann auf der Website unter dem Geheimtexteingabefeld angezeigt, um den Nutzer darauf aufmerksam zu machen, dass die eingegebene Nachricht nicht mit Base64 codiert wurde.

```
1     function isASCIINorm(str) {
2     const text = String(str)
3     for (let i = 0; i != text.length; i++) {
4         if (text.codePointAt(i) < 0 || text.codePointAt(i) > 256) {
5             return false
6         }
7     }
8     return true
9     }
```

Listing 5.6: ASCII-Zeichen Überprüfung

5.2.3 Problem

Im Listing 5.5 wird die Ausgabe des ersten Bit-Shift-Algorithmusschrittes, die Addition mit n, dargestellt. Dafür wird zuerst die im Algorithmus berechnete Dezimalzahl aus dem bereitgestellten Array ausgewählt. Dies geschieht anhand der im Abschnitt 5.2.1 beschriebenen Nachrichten- und Schlüsselzeichenauswahl. Anschließend wird diese mit der toString()-Methode in einen Binärstring umgewandelt und mit der padStart()-Methode überprüft, ob die Binärzahl 8-Bitstellen hat. Wenn nicht wird die entsprechende Anzahl an führenden Nullen ergänzt. Zum Schluss wird der gesamte Text noch mit TeX in die KaTeX-Formatierung umgewandelt. KaTeX ist eine JavaScript-Bibliothek, die schon im CTO-Projekt eingebunden ist. Diese sorgt für eine saubere Darstellung der mathematischen Notationen auf Websites.

Nachrichten, die mit dem Bit-Shift-Verfahren verschlüsselt werden, können bei großen Shift-Werten schnell sehr groß werden. Dies liegt daran, dass ein Klartextzeichen für alle Schlüsselzeichen verschoben und dadurch auf eine sehr große Binärzahl abgebildet wird. Diese wird anschließend mit Base64 in viele Zeichen umgewandelt. Schon bei dem Schlüssel („Schlüssel“, 8, 1) wird aus einem Klartext mit 65 Zeichen ein Geheimtext mit 1044 Zeichen. Das ist schon mehr als das 16-fache der Originallänge. Dementsprechend wird auch die Ausgabe der Zwischenschritte sehr groß, was die Website dadurch nicht nur unübersichtlich macht, sondern auch zu Abstürzen der Website führen kann.

Sowohl die zuvor beschriebene Umwandlung in den Binärstring mit der anschließenden Überprüfung der 8-Bitstellen, als auch die beschriebene KaTeX-Darstellung benötigen vergleichsweise viel Zeit und Rechenleistung. Dadurch kann es bei zu großen Schlüsselwörtern und Shift-Werten zu Verzögerungen bis hin zum Absturz der Website kommen. Dieses Problem konnte im Rahmen dieser Arbeit nur eingeschränkt, jedoch nicht vollständig behoben werden. So konnte an einigen Stellen die aufwändige KaTeX-Umgebung durch einfache CustomCodeBlocks, wie im Listing 5.2 zu sehen, ersetzt werden. Hierdurch wurde die Performance um einiges verbessert.

6 Schluss

In dieser Arbeit wurde zu Beginn in Kapitel 3 die neue Version des Bit-Shift-Verfahrens beschrieben. Das alte Verfahren wurde dabei um die Schlüsselparameter b und n erweitert. Zudem wurde auch die Reihenfolge im Bit-Shift-Algorithmus angepasst, damit die Entschlüsselung auch für Werte $n > 1$ zuverlässig funktioniert.

Darauffolgend wurde im Kapitel 4 die Sicherheit des Verfahrens gegen Brute-Force-Angriffe und Angriffe mittels Häufigkeitsanalyse untersucht. Der Brute-Force-Angriff ist wie zu erwarten eine sehr schlechte Methode, um die Schlüsselparameter herauszufinden, da das Durchprobieren aller Möglichkeiten mit aktueller Rechenleistung (2024) viel zu lange dauert. Es hat sich aber gezeigt, dass es mit Hilfe der Häufigkeitsanalyse möglich ist, die Verschlüsselung zu brechen und sowohl den Geheimtext als auch einen zugehörigen Schlüssel herauszufinden. Daher sollte Bit-Shift nicht für geheime Kommunikationen genutzt werden, denn die Sicherheit des Verfahrens ist, wie beschrieben, nicht gewährleistet.

Im Kapitel 5 der Arbeit wurden die alte Webapp und die neu implementierte App beschrieben. In der neu implementierten App werden nun zur einfacheren Verständlichkeit der Verschlüsselung Zwischenschritte mit zugehöriger Erklärung angezeigt. Für die bessere Übersichtlichkeit wurde die Eingabe der Schlüsselparameter an den Anfang gestellt, die Ver- und Entschlüsselung in zwei Registerkarten aufgeteilt und die Anzeige der Zwischenschritte in ausklappbaren Website-Inhalten realisiert. Hervorzuheben ist dabei, dass in der neuen Bit-Shift-App jetzt jeder Ver- und Entschlüsselungsschritt einzeln angezeigt werden kann. Zudem wurde auch die Funktionalität der App verbessert. So wird beispielsweise die Eingabe des Klartextes auf ASCII-Zeichen beschränkt und gegebenenfalls eine Fehlermeldung, bei Eingabe nicht konformer Zeichen, ausgegeben. Auch wurde das Problem mit dem Überlauf des Zahlenbereichs, durch Verwendung von BigInt, behoben.

Eine große Herausforderung der Bit-Shift-Verschlüsselung sind Klar- und Schlüsseltexte mit hoher Zeichenzahl, denn dadurch kann es zu Verzögerungen bis hin zum Absturz der App kommen. Dieses Problem konnte im zeitlichen Rahmen dieser Arbeit nicht vollständig behoben werden. Bei weiterer Beschäftigung mit diesem Verschlüsselungsverfahren wäre es wichtig, sich diesem Problem zu widmen und eine adäquate Lösung dafür zu finden. Zudem könnte auch die Sicherheit des Verfahrens noch verbessert und die Sicherheitsanalyse erweitert werden.

Bit-Shift wurde ursprünglich von Arthur Guiot aus didaktischen Gründen erstellt, implementiert und hatte keine Sicherheitsanalyse. Die zugehörige Bit-Shift-App war aber leider nur schwer verständlich und funktionierte, bei großen Schlüsselworten, nicht zuverlässig. In der vorliegenden Arbeit wurde eine verbesserte Version des Bit-Shift-Verfahrens entwickelt und dessen Sicherheit analysiert. Zudem wurde die zugehörige, verbesserte Bit-Shift-App in CrypTool-Online implementiert.

Literatur

[1] *ASCII-Tabelle nach Windows-1252*. URL: https://www.ascii-code.com/de. Abgerufen am 21.01.2024 (siehe S. 5).

[2] *Base64*. URL: https://developer.mozilla.org/en-US/docs/Glossary/Base64. Abgerufen am 17.02.2024 (siehe S. 6, 9).

[3] F. L. Bauer. *Entzifferte Geheimnisse. Methoden und Maximen der Kryptologie*. Springer-Verlag, 1995 (siehe S. 19, 21, 22).

[4] K-H. Best. „Glottometrics 11“. In: RAM-Verlag, 2005. Kap. Zur Häufigkeit von Buchstaben, Leerzeichen und anderen Schriftzeichen in deutschen Texten (siehe S. 22).

[5] *Blindtextgenerator*. URL: https://www.blindtextgenerator.de/. Abgerufen am 17.02.2024 (siehe S. 42).

[6] Dr. N. S. Borenstein und N. Freed. *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*. RFC 1521. Sep. 1993. DOI: 10.17487/RFC1521. URL: https://www.rfc-editor.org/info/rfc1521. Abgerufen am 21.01.2024 (siehe S. 5).

[7] *Brute Force Attacken: Trivialer und brutaler Passwortklau*. 2022. URL: https://www.psw-group.de/blog/brute-force-attacken-trivialer-und-brutaler-passwortklau/9532#:~:text=Ein%20hochmoderner%20Rechner%20schafft%20es,verschiedene%20Passw%C3%B6rter%20erstellen%20und%20durchprobieren.. Abgerufen am 21.01.2024 (siehe S. 17).

[8] *BSI TR-02102-1 Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile. Abgerufen am 21.01.2024 (siehe S. 17).

[9] J. Buchmann. *Einführung in die Kryptographie*. 6. Aufl. Springer-Verlag, 2016 (siehe S. 16).

[10] *Chakra UI*. URL: https://chakra-ui.com/. Abgerufen am 10.02.2024 (siehe S. 25).

[11] *CrypTool-Online*. URL: https://www.cryptool.org/de/cto/. Abgerufen am 21.01.2024 (siehe S. 3, 18).

[12] *CrypTool-Online*. URL: https://www.cryptool.org/de/cto/bitshift. Abgerufen am 16.02.2024 (siehe S. 3).

[13] B. Esslinger, Hrsg. *Das CrypTool-Buch: Kryptographie lernen und anwenden mit CrypTool und SageMath*. 12. Aufl. CrypTool-Projekt, 2018 (siehe S. 8).

[14] A. Guiot. *BitShiftCipher*. URL: https://github.com/CrypTools/BitShiftCipher. Abgerufen am 21.01.2024 (siehe S. 3).

[15] *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/MAX_SAFE_INTEGER. Abgerufen am 10.02.2024 (siehe S. 32).

[16] M. Miller. *Symmetrische Verschlüsselungsverfahren*. Vieweg+Teubner Verlag, 2003 (siehe S. 19).

[17] *React*. URL: https://react.dev/. Abgerufen am 21.01.2024 (siehe S. 25).

[18] *Statistical Distributions of English Text*. URL: https://web.archive.org/web/20170918020907/http://www.data-compression.com/english.html#first. Abgerufen am 25.02.2024 (siehe S. 22).

[19] A. Tanenbaum und D. Wetherall. *Computernetzwerke*. Pearson Deutschland, 2012 (siehe S. 5).

A Vollständiger Klartext

Dieser Klartext wurde in Kapitel 4.1 mit Hilfe der Häufigkeitsanalyse entschlüsselt:
„Jemand musste Josef K. verleumdet haben, denn ohne dass er etwas Böses getan hätte, wurde er eines Morgens verhaftet. »Wie ein Hund!« sagte er, es war, als sollte die Scham ihn überleben. Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt. Und es war ihnen wie eine Bestätigung ihrer neuen Träume und guten Absichten, als am Ziele ihrer Fahrt die Tochter als erste sich erhob und ihren jungen Körper dehnte. »Es ist ein eigentümlicher Apparat«, sagte der Offizier zu dem Forschungsreisenden und überblickte mit einem gewissermaßen bewundernden Blick den ihm doch wohlbekannten Apparat. Sie hätten noch ins Boot springen können, aber der Reisende hob ein schweres, geknotetes Tau vom Boden, drohte ihnen damit und hielt sie dadurch von dem Sprunge ab. In den letzten Jahrzehnten ist das Interesse an Hungerkünstlern sehr zurückgegangen. Aber sie überwanden sich, umdrängten den Käfig und wollten sich gar nicht fortrüh“ [5]

Schriftliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, die Zitate ordnungsgemäß gekennzeichnet und keine anderen, als die im Literatur/Schriftenverzeichnis angegebenen Quellen und Hilfsmittel benutzt zu haben.

Ferner habe ich vom Merkblatt über die Verwendung von studentischen Abschlussarbeiten Kenntnis genommen und räume das einfache Nutzungsrecht an meiner Bachelorarbeit der Universität der Bundeswehr München ein.

.....
(Unterschrift)