ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semesterbegleitende Übung

Kryptologie: Blockverschlüsselung

Lena Csomor, Fabian Haller, Beat Jäckle

Inhalt

Auftrag Thema Kryptologie

Zielpublikum

3. Klasse Gymnasium

Form

LPU

Betreuung

Prof. Juraj Hromkovič

Datum

15. März 2025

Inhaltsverzeichnis

Teil I

Konzeption der Unterrichtseinheit

1 Vorwissen

Wir setzten voraus, dass die Lernenden folgendes Vorwissen mitbringen:

- Die Lernenden kennen die Caesar-Verschlüsselung.
- Die Lernenden kennen die Vigenère-Verschlüsselung.
- Die Lernenden haben von den stochastischen Gedanken (Kasiski-Test, Friedmansche Charakteristik) gehört und wissen, dass man damit Kryptosysteme wie Vigenère angreifen kann.

2 Stoffanalyse

2.1 Concept Map

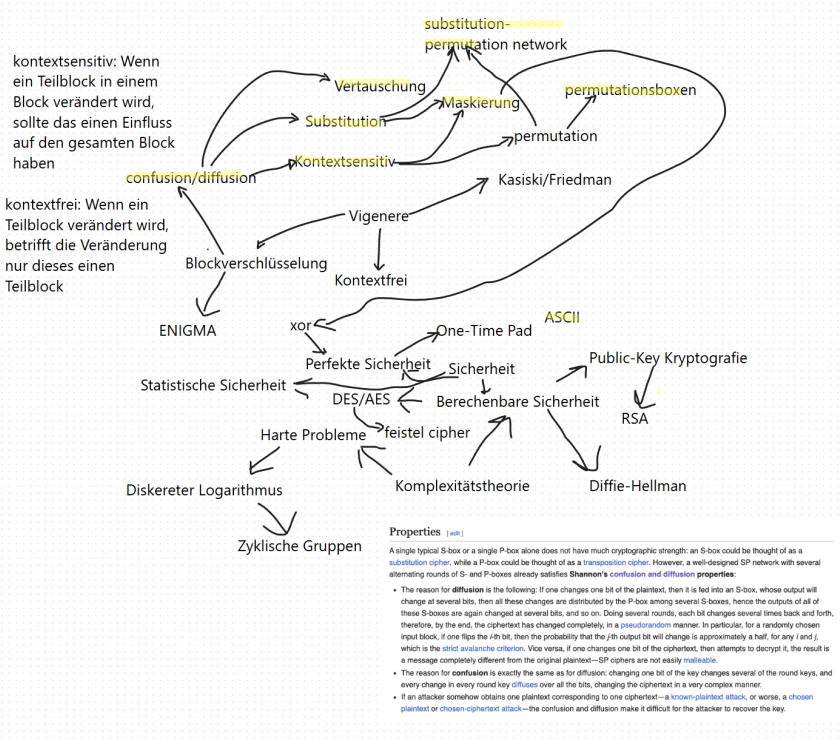


Abbildung 1: Recherche & Brainstorming

In der Abbildung 1 stehen die Begriffe, welche zu dieser Arbeit gehören.

In der Abbildung 2 ist die grob geplante Struktur des ersten Entwurfs der Unterrichtseinheit enthalten.

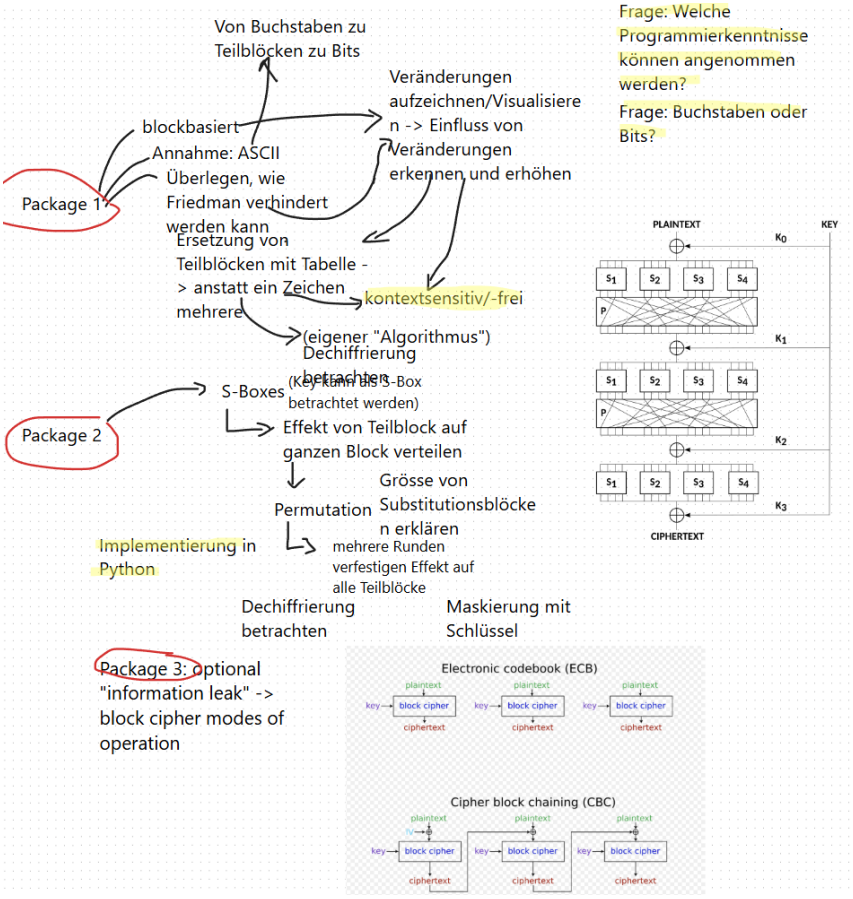


Abbildung 2: Grobplanung LPU

2.2 Leitidee

Viele moderne Verschlüsselungsalgorithmen beruhen auf kontextsensitiven Blockverschlüsselungen (Bsp. DES, AES). Deren Sicherheit basiert zu einem grossen Teil auf dem Lawineneffekt, ausgelöst durch rundenbasierte Substitution, Permutation und Maskierung. Aus diesem Grund sollten diese einzelnen Bestandteile sowie ihr Zusammenspiel aus kryptologischer Perspektive betrachtet und analysiert werden.

2.3 Dispositionsziele

1. Die Lernenden sind in der Lage, sich mit dem aus der Unterrichtseinheit erworbenen Wissen fortgeschrittenere Blockverschlüsselungen selbstständig anzueignen.
2. Die Lernenden sind in der Lage, mit ihrem Wissen über Kontextsensitivität ihnen zuvor unbekannte Kryptosysteme bezüglich ihrer Sicherheit gegen die erwähnten Angriffe (Kasiski / Friedman sowie verwandte, einfache Arten von Häufigkeitsanalysen) einzuschätzen.

2.4 Operationalisierte Lernziele

- Die Lernenden vergleichen was sich beim Chiffretext verändert, wenn sich beim Klartext nur ein Zeichen ändert.
- Die Lernenden verschlüsseln und entschlüsseln Teilblöcke anhand vorgegebenen Ta-

bellen.

- Die Lernenden können den Begriff 'k-lokale Geheimschrift' anhand Beispielen erklären.
- Die Lernenden kennen Methoden, wie Text binär kodiert werden kann und sind in der Lage, eine eigene Kodierung zu entwickeln.
- Die Lernenden kennen mit ASCII eine weit verbreitete Textkodierung.
- Die Lernenden kennen die Begriffe *Lookup Tabelle* und *Substitutionsbox* und wissen, wie eine Lookup Tabelle innerhalb einer Substitutionsbox verwendet wird.
- Die Lernenden können mithilfe der Beschreibung einer S-Box Text chiffrieren und dechiffrieren.
- Die Lernenden kennen die Stärken und Schwächen der S-Box bezüglich Sicherheit und Speicherverbrauch.
- Die Lernenden erklären den Begriff Lawineneffekt aus dem Stand und begründen seine Wichtigkeit für Block Ciphers mithilfe des Konzepts der Kontextsensitivität.
- Die Lernenden wählen selbstständig geeignete Permutationen so, dass ein möglichst starker Lawineneffekt gegeben ist.
- Die Lernenden können aus dem Stand aus zeigen, weshalb ein starker Lawineneffekt mindestens zwei Runden in einem Substitutions-Permutations-Netzwerk benötigt.
- Die Lernenden zeichnen aus dem Stand auf, wie das Substitutions-Permutations-Netzwerk vollständige Nicht-Lokalität garantiert.
- Die Lernenden erstellen selbstständig geeignete kontextsensitive Block Ciphers mithilfe von Substitutions-Permutations-Netzwerken.
- Die Lernenden erklären aus dem Stand, wann Maskierung mit einem Schlüssel sicher ist.

Teil II

Leitprogrammartige Unterrichtsmaterialien

Diese Seite wurde zum einfacheren Ausdrucken von Arbeitsblättern freigelassen.

Blockverschlüsselung

1 Schutz vor der Friedmanschen Methode

Die Friedmansche Methode erlaubt es, mithilfe von Statistik den Zusammenhang zwischen Klartext und Geheimtext herauszufinden. Eine Möglichkeit, dem entgegenzuwirken, besteht darin, zu analysieren, welche Buchstaben sich im Geheimtext ändern, wenn ein Buchstabe im Klartext modifiziert wird.

Beispiel 1.1. Wir chiffrieren die Nachricht: `Die Tiere wollen Futter.`
und `Die Tiere wollen Butter.`
Die Texte unterscheiden sich um nur einen Buchstaben:

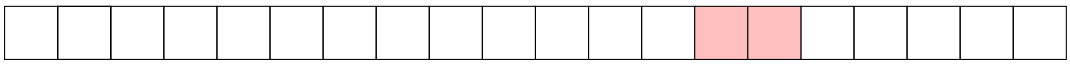
D	I	E	T	I	E	R	E	W	O	L	L	E	N	F	U	T	T	E	R
D	I	E	T	I	E	R	E	W	O	L	L	E	N	B	U	T	T	E	R

Verschlüsseln wir den Klartext mit Vigenère und dem Schlüssel: `DONTFEEDBUTTER`, so erhält man jeweils:

G	W	R	M	N	I	V	H	X	I	E	E	I	E	I	I	G	M	J	V
G	W	R	M	N	I	V	H	X	I	E	E	I	E	E	I	G	M	J	V

Es ändert sich im Geheimtext nur ein Zeichen.

Wir möchten eine Geheimschrift finden, sodass eine Buchstabenänderung im Klartext mehrere Buchstaben im Geheimtext ändert:



1.1 1-lokale Geheimschrift

Definition 1.1 (1-lokale Geheimschrift)**.** *Wir nennen eine Geheimschrift* 1-lokal*, wenn eine Veränderung von einem Buchstaben im Klartext genau einen veränderten Buchstaben im Geheimtext verursacht.*

Beispiel 1.2 (Caesar-Verschlüsselung mit Fallunterscheidung bei Vokalen)**.** Der Schlüssel besteht aus zwei Buchstaben. Für unser Beispiel nehmen wir (`J`,`B`). Als Klartext nehmen wir das Wort: `SCHLAUCH`.
Wir verschlüsseln die Buchstaben des Klartext einzeln und verwenden dabei entweder den ersten Schlüssel `J` oder den zweiten Schlüssel `B`. Welchen der zwei wir nehmen, hängt von dem vorherigen Buchstaben im Klartext ab. Wenn dort kein vorheriger Buchstabe ist oder ein Konsonant steht, verwenden wir `J`. Wenn der vorherige Buchstabe ein Vokal ist, verwenden wir den zweiten Schlüssel `B`.
Bis zu dem ersten Vokal `A` verwenden wir somit *Caesar(J)*:
`SCHLA` $\xrightarrow{Caesar(J)}$ `BLQUJ`
Weil nun ein Vokal (`A`) kam, nehmen wir für den nächsten Buchstaben des Klartextes `U` den zweiten Schlüssel für die Caesar-Verschlüsselung (`B`).
`U` $\xrightarrow{Caesar(B)}$ `V`
Da `U` im Klartext auch wieder ein Vokal war, nehmen wir auch für den nächsten Buchstaben wieder den zweiten Schlüssel.
`C` $\xrightarrow{Caesar(B)}$ `D`
Der letzte verschlüsselte Buchstabe im Klartext war `C`, ein Konsonant. Deswegen nehmen wir wieder den ersten Schlüssel.
`H` $\xrightarrow{Caesar(J)}$ `Q`.
Der Geheimtext lautet: `BLQUJVDQ`
Den Text kann man auch wieder von links nach rechts entschlüsseln. Jeweils, wenn ein Vokal im Klartext entschlüsselt wurde, müssen wir den nächsten Buchstaben mit dem zweiten Schlüssel zu entschlüsseln.

Aufgabe 1.1 (Caesar-Verschlüsselung mit Fallunterscheidung, Unterschied finden)**.** Das Wort `SIHLAUCH` ergibt keinen Sinn. Verschlüsseln Sie es trotzdem mit dem Schlüssel (`J`, `B`), denn `SIHLAUCH` und `SCHLAUCH` unterscheiden sich nur um einen Buchstaben.
Vergleichen Sie, wie viele Buchstaben sich im Geheimtext ändern.

Beispiel 1.3 (Caesar-Verschlüsselung mit Offset)**.** Der Schlüssel besteht aus einem Buchstaben und einer natürlichen Zahl n. Für unseres Beispiel nehmen wir (`J`,`3`). Als Klartext nehmen wir das Wort: `CAESAROFFSET`.
Zuerst verschlüsseln wir den Text mit der Caesar-Verschlüsselung mit dem Buchstaben aus dem Schlüssel.
Anschliessend verschlüsseln wir mit dem Vigenère-System. Dazu verwenden wir den Schlüssel beginnend mit n mal dem Buchstaben `A` und anschliessend dem Klartext. In unserem Beispiel ist das `AAACAESAROFFSET`

C	A	E	S	A	R	O	F	F	S	E	T
J	J	J	J	J	J	J	J	J	J	J	J
L	J	N	B	J	A	X	O	O	B	N	C
A	A	A	C	A	E	S	A	R	O	F	F
L	J	N	D	J	E	P	O	F	P	S	H

Der Geheimtext lautet: `LJNDJEPOFPSH`
Den Text kann man auch wieder von links nach rechts entschlüsseln. Den bekannten Klartext braucht man für die kommenden Buchstaben.

Aufgabe 1.2 (Caesar-Verschlüsselung mit Offset)**.** Verschlüsseln Sie mit dem Schlüssel (`C`, 4), den Klartext `UEBUNG` mit der *Caesar-Verschlüsselung mit Offset*.

Aufgabe 1.3 (Wissenssicherung 1-lokale Geheimschrift)**.** Welche von den vier Geheimschriften sind *1-lokale Geheimschriften*?

1. Caesar-Verschlüsselung
2. Vigènere-Verschlüsselung
3. Caesar-Verschlüsselung mit Fallunterscheidung
4. Caesar-Verschlüsselung mit Offset

Erklären Sie in eigenen Worten, wie Sie eine 1-lokale Geheimschrift erkennen.

2 Blockverschlüsselung

Das haben Sie bis hierher gelernt:

- Sie zählten die Veränderung im Geheimtext nach einer Veränderung im Klartext.
- Sie kennen den Begriff *1-lokale Geheimschrift*.

Um mehrere Buchstaben im Geheimtext zu ändern, werden oft ganze Teilblöcke ersetzt, anstatt wie bis anhin Buchstaben.

Beispiel 2.1 (Teilblöcke ersetzen)**.** Um es übersichtlicher zu machen, beschränken wir uns auf ein Alphabet, welches nur aus 5 Buchstaben besteht: $A = \{E, N, I, R, A\}$
Das Wort `Rennen` ist ein Wort aus diesem Alphabet.

R E	N N	E N

Verwenden wir eine Verschlüsselung wie folgt:

EN → EE	NE → NE	NR → RE	IA → EA	AI → NR	RN → NA
EI → RA	NN → NI	IE → ER	IR → AI	AA → RR	RI → EN
EA → AA	NI → IE	IN → EI	AE → IN	AR → AE	RA → IA
ER → NN	NA → RI	II → RN	AN → AR	RE → AN	RR → II

oder als Tabelle, wobei der erste Buchstabe vom Block in der Zeile und der zweite Buchstabe in der Spalte zu finden sind:

	E	N	I	A	R
E	IR	EE	RA	AA	NN
N	NE	NI	IE	RI	RE
I	ER	EI	RN	EA	AI
A	IN	AR	NR	RR	AE
R	AN	NA	EN	IA	II

Dann ist der Geheimtext:

A N	N I	E E

Aufgabe 2.1 (Textblock verschlüsseln)**.** Verschlüsseln Sie `NENNEN` mit der Verschlüsselung aus dem Beispiel 2.1.

Aufgabe 2.2 (Textblock entschlüsseln)**.** Entschlüsseln Sie die `IENNEE` mit der Verschlüsselung aus dem Beispiel 2.1.

	E	N	I	A	R
E	EN	RI	IN	IA	IE
N	NE	ER	NN	RN	AI
I	NI	AE	RR	RA	EE
A	AR	RE	IR	EA	AN
R	NR	II	NA	EI	AA

Tabelle 1: Entschlüsselungstabelle für die Verschlüsselung aus dem Besispiel 2.1

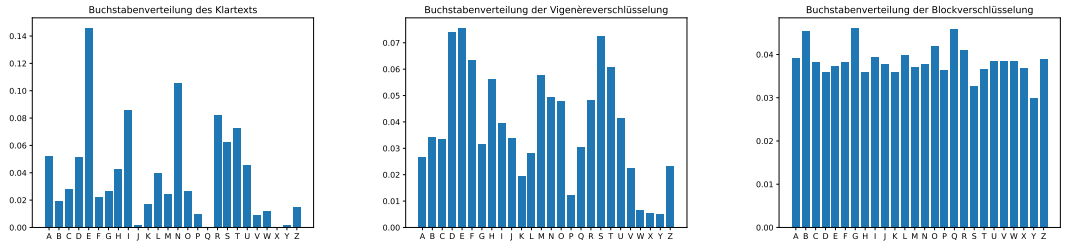


Abbildung 3: Buchstabenverteilungen vom Klartext, Vigènere und Blockverschlüsselung

Aufgabe 2.3 (Überprüfung der Änderung)**.** Zwischen dem Beispiel 2.1 und der Übung 2.1 hat sich im Klartext nur ein Buchstabe geändert.
Vergleichen Sie, wie viele Buchstaben sich in den entsprechenden Geheimtexten von `ANNIEE` zu `NENIEE` geändert haben.

Wie Sie vor dieser Lerneinheit erfahren haben, gibt es stochastische Methoden eine Verschlüsselung zu knacken. In der deutschen Sprache nutzt man den Buchstaben E am meisten. Die Caesarverschlüsselung verschiebt die Verteilung. Die Spitze existiert weiterhin und lässt uns den Schlüssel herausfinden.

Die Vigènere-Verschlüsselung mit einem Wort der Länge n ist von der Verteilung die Summe von n Caesar-Verschlüsselung. In der Abbildung 3 wurde der Schlüssel ABZ für die Vigènere-Verschlüsselung verwendet. Das `E` wird also auf `E`, F und `D` abgebildet. Man kann erkennen, dass diese drei Buchstaben in den häufigsten vier Zahlen des Geheimtexts vorkommen. Dieser Effekt wird kleiner, wenn das Schlüsselwort länger wird. Sie wissen jedoch, dass man mit dem Kasiski-Test und der Friedmansche Charakteristik auch andere Methoden gibt, die Vigènere-Verschlüsselung zu knacken.

Wir könnten jeden Block als ein neues Zeichen betrachten und somit erhalten wir ein viel grösseres Alphabet. Verschlüsseln wir einen Text mit der Blocklänge n, dann ersetzen wir nicht mehr 26 Symbole — wie bei einer *1-lokaler Geheimschrift* — sondern wir haben 26^n Symbole. Also gibt es 26^n Einträge in der Substitutionstabelle. Die Proportionen zwischen den Häufigkeiten von einzelnen Blöcken werden viel geringer als die Verhältnisse zwischen den Häufigkeiten von einzelnen Buchstaben.

Eine Tabelle für Blöcke der Länge 8 ist schon so gross, dass sie nicht mehr handhabbar wird. Es braucht deswegen eine einfachere Beschreibung der der Chiffrierung, die man effizient ausführen kann.

3 ASCII

Das haben Sie bis hierher gelernt:

- Sie wissen, was Substitutionstabellen sind.
- Sie wissen, wie dezimale Zahlen binär dargestellt werden und kennen das binäre System.

Im letzten Abschnitt haben wir gesehen, dass der Einfluss einer Änderung im Klartext auf den Geheimtext vergrössert werden muss, um zu verhindern, dass mittels einer Häufigkeitsanalyse der Geheimtext geknackt werden kann. In diesem Abschnitt werden wir das Verschlüsseln von Teilblöcken anhand Tabellen weiter formalisieren und analysieren. Dies wird uns die Grundlagen dafür geben, erste moderne Blockchiffrierungen zu bauen.

Alle Kryptosysteme, welche Sie bisher kennengelernt haben, wurden zu einer Zeit entwickelt, als es noch keine automatische Informationsverarbeitung gab. Damals wurde Chiffrierung und Dechiffrierung noch manuell von einem Menschen durchgeführt. Heute chiffriert und dechiffriert niemand mehr Nachrichten von Hand sondern diese Arbeit wird an einen Computer abdeligiert.
Wie Sie wissen, speichert ein Computer Daten binär ab. Dadurch stellt sich jedoch die Frage, wie ein Computer das lateinische Alphabet sowie Interpunktionszeichen binär darstellt.

Aufgabe 3.1. Der nachfolgende Text wurde mit Bitfolgen der Länge 5, die man auch als Zahlen aus dem Intervall 0 bis 31 ansehen kann. Dabei stellt die Null (00000) ein Leerzeichen dar und die nachfolgenden 26 Zahlen kodieren das lateinische Alphabet in aufsteigender Reihenfolge (00001 = 'A', 00010 = 'B', ... , 11010 = 'Z'). Erstellen Sie zuerst eine vollständige Übersetzungstabelle und versuchen Sie dann, den nachfolgenden Binärtext zu übersetzen.

00100 00001 10011 00000 01001 10011 10100 00000 00101 01001 01110 00000 00010
01001 01110 00001 00101 10010 10100 00101 11000 10100

Vielleicht ist Ihnen aufgefallen, dass die 5-Bit-Zeichenkodierung von Aufgabe 3.1 ziemlich limitiert ist und verschiedene Satz- und Interpunktionszeichen nicht beinhaltet und auch Klein- und Grossschreibung nicht unterscheidet. Zwar hätten wir in der 5-Bit-Zeichenkodierung noch Platz für fünf weitere Zeichen (11011 - 11111), jedoch würde das nicht reichen um zwischen Gross- und Kleinschreibung zu unterscheiden. Insgesamt können wir also nur $2^5 = 32$ unterschiedliche Zeichen mit 5 Bits kodieren. Um einen alltäglichen Text vernünftig zu kodieren, brauchen wir also eine grössere Bitweite.

Beispiel 3.1 (Zeichenkodierung ASCII)**.** ASCII (American Standard Code for Information Interchange) ist eine international anerkannte und standardisierte 7-Bit-Zeichenkodierung welche 1963 vom amerikanischen Normungsinstitut veröffentlicht wurde. ASCII beinhaltet $2^7 = 128$ Zeichen wobei 33 davon nicht druckbare Steuerzeichen sind. Viele der Steuerzeichen sind historisch entstanden und werden heute praktisch nicht mehr gebraucht. Beispielweise das Steuerzeichen *BEL* mit der numerischen Kodierung 7 wurde früher verwendet, um ein Tonsignal zu erzeugen (englisch Bell = Glocke). Ursprünglich wurde noch ein achtes Bit verwendet, um mögliche Übermittlungsfehler zu erkennen, dieses werden wir aber im Nachfolgenden nicht beachten.

Bin	ASCII	Bin	ASCII	Bin	ASCII	Bin	ASCII
000'0000	NUL	010'0000	SP	100'0000	@	110'0000	‘
000'0001	SOH	010'0001	!	100'0001	A	110'0001	a
000'0010	STX	010'0010	”	100'0010	B	110'0010	b
000'0011	ETX	010'0011	#	100'0011	C	110'0011	c
000'0100	EOT	010'0100	$	100'0100	D	110'0100	d
000'0101	ENQ	010'0101	%	100'0101	E	110'0101	e
000'0110	ACK	010'0110	&	100'0110	F	110'0110	f
000'0111	BEL	010'0111	’	100'0111	G	110'0111	g
000'1000	BS	010'1000	(	100'1000	H	110'1000	h
000'1001	TAB	010'1001	)	100'1001	I	110'1001	i
000'1010	LF	010'1010	*	100'1010	J	110'1010	j
000'1011	VT	010'1011	+	100'1011	K	110'1011	k
000'1100	FF	010'1100	,	100'1100	L	110'1100	l
000'1101	CR	010'1101	-	100'1101	M	110'1101	m
000'1110	SO	010'1110	.	100'1110	N	110'1110	n
000'1111	SI	010'1111	/	100'1111	O	110'1111	o
001'0000	DLE	011'0000	0	101'0000	P	111'0000	p
001'0001	DC1	011'0001	1	101'0001	Q	111'0001	q
001'0010	DC2	011'0010	2	101'0010	R	111'0010	r
001'0011	DC3	011'0011	3	101'0011	S	111'0011	s
001'0100	DC4	011'0100	4	101'0100	T	111'0100	t
001'0101	NAK	011'0101	5	101'0101	U	111'0101	u
001'0110	SYN	011'0110	6	101'0110	V	111'0110	v
001'0111	ETB	011'0111	7	101'0111	W	111'0111	w
001'1000	CAN	011'1000	8	101'1000	X	111'1000	x
001'1001	EM	011'1001	9	101'1001	Y	111'1001	y
001'1010	SUB	011'1010	:	101'1010	Z	111'1010	z
001'1011	ESC	011'1011	;	101'1011	[	111'1011	{
001'1100	FS	011'1100	¡	101'1100	\	111'1100	—
001'1101	GS	011'1101	=	101'1101	]	111'1101	}
001'1110	RS	011'1110	¿	101'1110	ˆ	111'1110	˜
001'1111	US	011'1111	?	101'1111	_	111'1111	DEL

Tabelle 2: ASCII Tabelle

Aufgabe 3.2. Der folgende Text wurde mit ASCII kodiert. Versuchen Sie den Text mithilfe der ASCII Tabelle in Abbildung 2 zu dekodieren. Arbeiten Sie dazu in Zweiergruppen.

1000001 1010011 1000011 1001001 1001001 0100000 1101011 1100001 1101110
1101110 0100000 1101011 1100101 1101001 1101110 1100101 0100000 1010101
1101101 1101100 1100001 1110101 1110100 1100101 0100000 1101011 1101111
1100100 1101001 1100101 1110010 1100101 1101110 0101110 0101110 0101110

Nun da Sie gelernt haben, wie Text binär kodiert werden kann, werden wir unseren Fokus wieder zurück auf das Chiffrieren von Blöcken legen. Ab diesem Punkt werden wir jedoch direkt Binärcodes anstelle von Buchstaben verschlüsseln. Mit anderen Worten, die Bits werden unsere neuen "Buchstaben" werden.

Aufgabe 3.3. Wir haben bereits das Kryptosystem von Vigenère kennengelernt. Versuchen Sie nun den folgenden Binärtext, welcher mit Vigenère verschlüsselt und mit ASCII kodiert wurde zu entschlüsseln und anschliessend zu dekodieren. Zuerst wurde der Text kodiert und anschliessend verschlüsselt. Der Schlüssel für Vigenère lautet 10110. Achten Sie darauf dass die Bitweite des Schlüssels und der Kodierung nicht identisch sind und gruppieren Sie die Bits allenfalls nach der Entschlüsselung neu.

00111 10000 11101 01111 10010 10101 10100 01100 11000 01011 00110 01010 00001
00100 01111 11100 10111 11101 11011 10001 10001 01111 11000 01011 00100 01000
11101 10011 11110 10000 11101 00001 11000 00101 00101 01010 10001 10011 00010
01010 11001 11110 01011 10100 10111 00100 11010 00001 11011 11110 10000 10011
00010 01000 00101 00001 01111 11000 00101 01111 01010 00001 11000 11110 10011
00111 10001 01100 11110

Interessanterweise entspricht das bitweise Anwenden von Vigenère genau dem sogenannten *XOR Gatter*. Das *XOR Gatter* ist eine Funktion, welche zwei Bits als Input nimmt und 0 ausgibt, falls die Bits identisch sind und 1 ausgibt, falls die Bits unterschiedlich sind, wie in der Tabelle 3 dargestellt.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 3: XOR Gatter

Die Anwendung des *XOR Gatters* mit einem Schlüssel wie in Aufgabe 3.3 wird auch *Maskierung* genannt und ist ein wichtiger Baustein für viele moderne Verschlüsselungsverfahren.

Die Verwendung der ASCII Kodierung erschwert das Brechen eines Kryptosystems grundsätzlich nicht, es dient lediglich der binären Darstellung von Buchstaben und Zeichen.

4 Substitutionsboxen

Das haben Sie bis hierher gelernt:

- Sie kennen Substitutionstabellen.
- Sie können Buchstaben und Texte kodieren und kennen mit ASCII ein Beispiel einer Kodierung.
- Sie kennen den Begriff *Lokalität*.

Wie wir im letzten Abschnitt festgestellt haben, besteht die hauptsächliche Schwäche von Vigenère darin, dass die Änderung eines Zeichens im Klartext nur einer Änderung im Geheimtext entspricht. Um den Effekt einer Änderung im Klartext zu vergrössern, müssen wir also dafür sorgen, dass bei der Veränderung eines Zeichens im Klartext mehrere Zeichen im Geheimtext verändert werden. Um das zu erreichen, können wir anstatt jedes Zeichen einzeln zu verschlüsseln auch mehrere Zeichen miteinander verschlüsseln. Die Zeichenkette, welche wir zusammen verschlüsseln, nennen wir **Block**. Für jede mögliche Zeichenkombination innerhalb eines Blockes müssen wir also eine Verschlüsselung definieren.

Aufgabe 4.1 (Anzahl von Zeichenkombinationen)**.** Wie viele Möglichkeiten gibt es, einen 5-Bit Schlüssel wie denjenigen aus Aufgabe 3.3 zu erstellen?
Wie viele Möglichkeiten gibt es für einen 3-stelligen Schlüssel mit Zeichen des lateinischen Alphabetes (26 Buchstaben)?

Wir müssen also jeder Zeichenkombination einen Geheimtext zuweisen. Eine solche Zuweisung von einem Input zu einem Output wird in der Informatik häufig auch *Lookup Tabelle* genannt. Eine *Lookup Tabelle* weist jedem Input einen Output zu. Ein Beispiel für eine *Lookup Tabelle* finden Sie in Tabelle 4.

Input	Output
000	100
001	110
010	001
011	011
100	111
101	010
110	000
111	101

Tabelle 4: Beispiel für eine Lookup Tabelle mit einem Input und Output von 3-Bit

Aufgabe 4.2. Der folgende mit ASCII kodierte Text wurde mit der Lookup-Tabelle von Tabelle 4 verschlüsselt. Versuchen Sie, den Text zu entschlüsseln. Hinweis: Die Bitweite der ASCII-Kodierung beträgt 7 Bits.
111 000 011 011 101 010 101 000 010 101 010 011 000 100 001 100 001 010 110 111 110 000 110 011 011 110 111 010 101 110 111

Aufgabe 4.3. Wie viele mögliche Anordnungen zwischen Input und Output gibt es für die Tabelle 4?
Wie viele mögliche Anordnungen würde es für eine Tabelle mit 5-Bits geben? Wie viele für n-Bits? Berechnen Sie dazu zuerst die Anzahl Einträge für grössere Tabellen.

Wie wir in Aufgabe 4.3 gesehen haben, wächst die Anzahl an möglichen Anordnungen der Lookup Tabelle mit steigender Bitweite rapide an. Bereits für eine Bitweite von drei, wie in Tabelle 4 dargestellt, gibt es 40320 mögliche Anordnungen. Mit anderen Worten müsste man 40320 verschiedene Möglichkeiten ausprobieren, um die korrekte Tabelle zu finden. Bei einer Bitweite von 8 gäbe es bereits mehr als 10^{506} Möglichkeiten.
Gleichzeitig vergrössert sich aber auch die Tabellengrösse exponentiell mit der Bitweite.

Definition 4.1 (Substitutionsbox (S-Box))**.** *Eine $m \times n$ S-Box ist eine im Allgemeinen nichtlineare Substitutionsoperation, bei der ein m-stelliger Binärtext durch einen n-stelligen Binärtext ersetzt wird.*

Wir werden uns in diesem Kapitel auf *symmetrische* und *invertierbare* S-Boxen beschränken. *Symmetrisch* heisst in diesem Kontext, dass die Bitweite des Inputs dieselbe ist wie diejenige des Outputs (sprich $n \times n$ S-Box) und *invertierbar* heisst, dass wenn man den Output sowie die Substitutionstabelle kennt, eindeutig auf den Input zurückgeschlossen werden kann. Ein Beispiel einer *symmetrischen* und *invertierbaren* $n \times n$ S-Box wurde bereits in Tabelle 4 vorgestellt, welche Sie in Aufgabe 4.2 verwendet haben, um die Geheimschrift zu dekodieren. Falls die Lookup-Tabelle nicht invertierbar gewesen wäre (sprich mehrere Inputs auf denselben Output abgebildet worden wären), dann hätten Sie den Geheimtext nicht mehr eindeutig dechiffrieren können.

Aufgabe 4.4. Sie haben eine geheime Nachricht mit dem folgenden Inhalt abfangen können:

0101010 1000000 1100011 1100001 1101100 1110101 0111111 1000010 0010011
0101101 0011001 0001010 1101100 1100110 1010001 1110000 0101100 1100101
0001011 0000011 1101100 1100101 0111110 0110111 0010000 0101100 0110001
1100101 1110010 1001001 1000010 1000001 0010001 0101100 1110101 1101011
1110101 1101101 1011111 1100010 1100101 0001010 0100010 1000001 1101100
1000000 1100111 0000001 1101011 0110110 0100001 0000111 0010110 1101100
0100110 1000001 1110101 1101101 1010101 1111111 0001100 0101101 1000011
1100101 0010001 0010101 1111111 0000001 1101100 1010101 1000010 1000111
1110010 1110101 0011111 1100010 1101000 0101100 0100011 1101110 1110010
1010101 1000011 1100110 0010001 1001000 1100111 0000011 1100000 1000000

Sie wissen, dass der Klartext mit 7-Bit ASCII kodiert und mit einer symmetrischen $n \times n$ S-Box chiffriert wurde. Leider kennen Sie n nicht genau, wissen aber, dass n kleiner als 6 ist. Da Sie den Empfänger der Nachricht kennen, wissen Sie auch dass der Anfang der Nachricht höchstwahrscheinlich mit dem Text 'Lieber Michael' beginnt. Versuchen Sie mit diesen Angaben den Klartext wiederherzustellen.

Aufgabe 4.5. In einer Verschlüsselung wird jeder Buchstabe mit k Bits kodiert und mit einer $n \times n$ S-Box verschlüsselt. Wie viele Buchstaben lang ist der kleinste Teilblock eines Klartextes, welcher immer auf denselben Teilblock im Geheimtext abgebildet wird?
Wie viele Buchstaben ändern sich maximal innerhalb eines Teilblockes des Geheimtextes, wenn ein einzelner Buchstabe im Klartext desselben Teilblockes verändert wird?
Beispiel für $k = 4, n = 8$: In diesem Fall ist ein Buchstabe mit 4 Bits kodiert und die S-Box verschlüsselt jeweils Blöcke von 8 Bits. In diesem Fall werden also immer zwei Buchstaben zusammen verschlüsselt (da $2k = n$). Wenn wir also einen Klartext mit dem Inhalt *hahaha* verschlüsseln würden, hätte der Geheimtext die Form *xyxyxy* wobei x und y beliebige Textzeichen sind. Somit würden jeweils immer 2 Buchstaben (im konkreten Fall *ha*) auf denselben Geheimtext abgebildet (*xy*). Sobald ein Buchstabe im Klartext verändert wird, können sich immer beide Buchstaben des Geheimtextes verändern und somit ist die Antwort auf die zweite Frage ebenfalls 2 in diesem Beispiel.
Finden Sie zuerst die Anzahl Buchstaben für den Fall $k = 7, n = 5$. Versuchen Sie dann eine Regel für beliebige ks und ns zu finden.
Achtung: Die zweite Frage ist im Allgemeinen anspruchsvoll und es muss eine Fallunterscheidung von 6 Fällen gemacht werden je nach n und k.

Wie Sie vielleicht in Aufgabe 4.5 bemerkt haben, ist die Nicht-Lokalität nur innerhalb der S-Box gewährleistet. Derselbe Input einer S-Box führt gezwungenermassen zum selben Output. Falls die Bitweite der S-Box klein ist (z.B. 7 Bits), dann kann die S-Box ziemlich einfach mit stochastischer Kryptoanalyse geknackt werden. Jedoch können wir die Bitweite der S-Box auch nicht unlimitiert vergrössern, da wir ansonsten eine exponentiell wachsende Lookup-Tabelle speichern müssten. Bereits eine symmetrische S-Box mit Bitweite 32 würde 0.5 Gigabyte Speicher benötigen. Wir benötigen also eine Alternative, um die Nicht-Lokalität auf längere Bitfolgen auszuweiten.

5 Permutationsboxen

Das haben Sie bis hierher gelernt:

- Sie wissen, was Substitutionsboxen sind.
- Sie wissen, welche Eigenschaften eine Substitutionsbox erfüllen muss, um für Chiffrierung und Dechiffrierung zu funktionieren.
- Sie verstehen, wie ein verändertes Bit einen ganzen Teilblock beeinflussen kann.

5.1 Der Lawineneffekt

In den vorherigen Abschnitten haben wir erfahren, dass wir uns gegen die statistischen Attacken auf Vigenère wehren können. Das bedingt, dass ein verändertes Zeichen im Klartext mehr als ein Zeichen im Geheimtext ändert. Dadurch verschleiern wir statistische Zusammenhänge zwischen dem Klartext und dem Geheimtext besser als bei Vigenère. Dies konnten wir in in Abbildung 3 besonders gut beobachten. Eine gleichmässigere Verteilung der Buchstaben im Geheimtext ist sehr viel schwieriger auf Auffälligkeiten zu untersuchen als eine ungleichmässige.

Definition 5.1 (k-lokale Geheimschrift)**.** *Wir nennen eine Geheimschrift* k-lokale Geheimschrift*, wenn eine Veränderung von einem Buchstaben im Klartext genau k veränderte Buchstaben im Geheimtext verursacht.*

Bisher erreichen wir **k-Lokalität** durch Substitutionsboxen: wenn wir nur schon ein einzelnes Bit in einem Teilblock verändern, wählen wir ein völlig anderes Element aus der Substitutionstabelle, als wir sonst ausgesucht hätten.
Das bedeutet, dass bei einer Substitutionsbox mit den Dimensionen $m \times n$ sich bis zu n Bits verändern können, wenn nur schon ein einziges der m Bits der Eingabe verändert wird.
Sehen wir uns die Tabelle vom Anfang dieses Kapitels nochmals an. Hier hat ein Teilblock die Länge 2. Der Teilblock ist beschränkt durch die Anzahl Zeichen, die wir mit der Tabelle auf einmal verschlüsseln können.

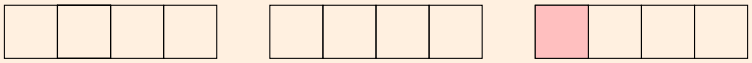
Beispiel 5.1. Sie möchten eigentlich den Klartext-Teilblock **EN** verschlüsseln, machen aber einen Tippfehler und der Teilblock wird zu **EI**. Dann erhalten Sie im Geheimtext anstatt **EE** den Text **RA**.

	E	N	I	A	R
E	IR	EE	RA	AA	NN
N	NE	NI	IE	RI	RE
I	ER	EI	RN	EA	AI
A	IN	AR	NR	RR	AE
R	AN	NA	EN	IA	II

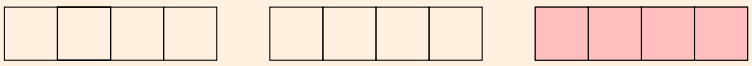
Die maximale Menge an veränderten Bits, die wir mit einer veränderten Eingabe erreichen können, beschränkt sich damit auf einen Teilblock. Alle Teilblöcke, bei denen die Eingabe nicht verändert wurde, bleiben so wie vorher.
Hier sehen Sie jeden Buchstaben als Quadrat. Veränderte Zeichen werden in pink dargestellt:

Klartext:

Klartext in Teilblöcken:



Geheimtext:



Wir haben festgestellt, dass das sicherer ist als Vigenère. Aber es geht noch weiter. Zwar sind die einzelnen Buchstaben besser geschützt, aber die Einteilung in Teilblöcke hat zur Konsequenz, dass sich wiederholende Teilblöcke angreifbar sind. Wir möchten aber, dass unser Geheimtext keine Informationen über den Klartext preisgibt. Sobald Informationen preisgegeben werden, können diese für einen Angriff verwendet werden.

Aufgabe 5.1. Sie wissen, dass der Klartext `RENNEN` verschlüsselt wurde als `ANEINA`. Sie kennen die zugehörigen Substitutionsboxen nicht, wissen aber, dass sie Länge 2 haben. Sie erhalten einen weiteren Geheimtext, dessen Klartext sie nicht kennen. Sie wissen aber, dass dieselben Substitutionsboxen verwendet wurden: `RREINA`.
Können Sie einen Teil davon entschlüsseln? Haben Sie eine Idee, um welches Wort es sich handeln könnte?

Aufgabe 5.2. Diskutieren Sie: Welche Konsequenzen hat diese Angriffsmöglichkeit für einen Angreifer, der einige solcher Klartext-Geheimtext-Paare besitzt? Kann sich der Empfänger einer Nachricht sicher sein, dass der Sender wirklich die Substitutionsboxen besitzt?

Aufgabe 5.3. Sie wissen, dass der Klartext `ERINA, RENN NIE IN ANNE` verschlüsselt wurde als `REARAINIEEINAREIIR`. Sie kennen die zugehörigen Substitutionsboxen nicht, wissen aber, dass sie Länge 2 haben. Konstruiere einen weiteren, gültigen Geheimtext. Notieren Sie auch dessen Entschlüsselung im Klartext.

Aufgabe 5.4. Diskutieren Sie: Wie können Sie diese Angriffsmöglichkeit verhindern?

Wir stellen fest: Zwei unterschiedliche Klartexte müssen **komplett unterschiedliche Geheimtexte** generieren, sonst sind sie angreifbar. Versuchen wir also, den Effekt von einem veränderten Bit auch auf andere Teilblöcke auszuweiten.

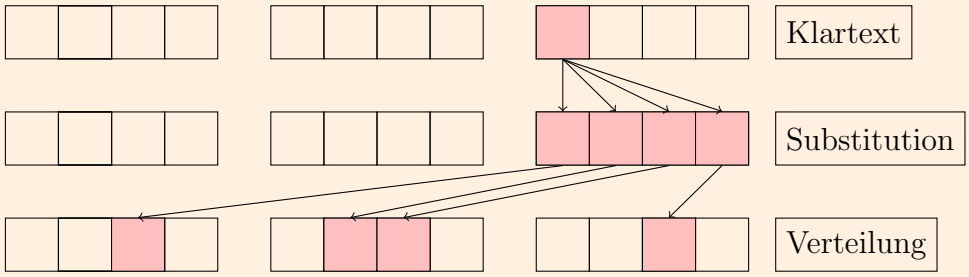
Definition 5.2 (Lawineneffekt)**.** *Wenn eine kleine Veränderung im Klartext eine grosse Veränderung im Geheimtext verursacht, sprechen wir von einem* ***Lawineneffekt****. Je grösser die Veränderung, desto* ***stärker*** *ist der Lawineneffekt.*

In den vorherigen Aufgaben haben wir erkannt, dass unser Lawineneffekt bis jetzt noch nicht stark genug ist, denn er bleibt innerhalb eines Teilblocks. Dies reicht nicht aus, um eine genügend hohe Sicherheit gegen Angriffe zu garantieren. Wir müssen im Folgenden überlegen, wie der Lawineneffekt verstärkt werden kann.

Aufgabe 5.5. Diskutieren Sie: Wir haben den Effekt von einem veränderten Zeichen auf einen Teilblock vergrössert. Wie können wir die Veränderung von diesem Teilblock auf den ganzen Block ausweiten? Wir haben jetzt zwar mehrere veränderte Zeichen, statt nur eines wie bei Vigenère, aber sie sind alle im selben Teilblock. Wir haben bereits gelernt, dass grosse Substitutionsboxen sehr ineffizient sind. Finden Sie eine andere Strategie?

Wir erkennen: um den Lawineneffekt zu verstärken, müssen wir die durch die Substitution veränderten Bits in verschiedene Teilblöcke unterbringen können.

Beispiel 5.2. Wir haben bisher jeweils markiert, wenn ein Bit von einer Veränderung betroffen war. Wir könnten also zum Beispiel die markierten Bits aus einem Teilblock auf die anderen Teilblöcke verteilen.

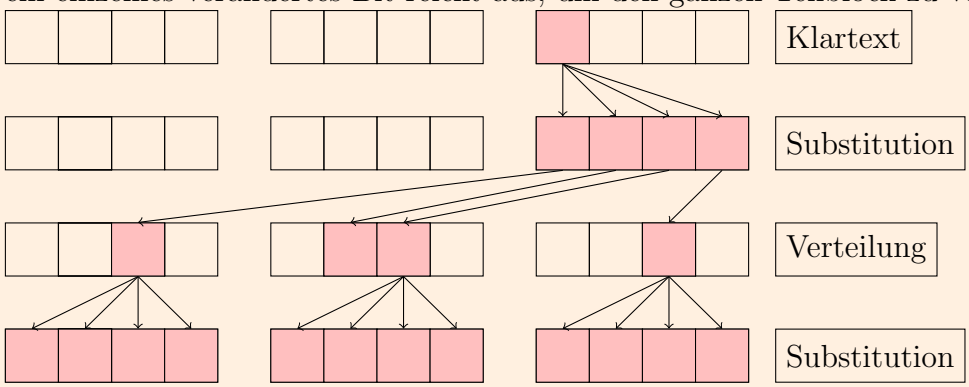


Beachten Sie: Um die Bits erfolgreich zu verteilen müssen wir sie mit anderen Bits tauschen. Um den Geheimtext am Schluss entschlüsseln zu können, müssen wir in der Lage sein, diesen Schritt rückgängig zu machen. Wie können wir dechiffrieren, wenn wir anfangen, Bits zu vertauschen? **Wir müssen uns nur die Vertauschungen merken**. Sie wird damit Teil des Schlüssels. Im Beispiel oben wäre die sogenannte **Permutation** für den letzten Teilblock: [3, 6, 7, 11].

Definition 5.3 (Permutation)**.** *Die* ***Permutation*** *ist eine Vertauschung, bzw. Neuanordnung von Elementen in einer Menge. Im Kontext von der Chiffrierung sind die Elemente in der Regel Buchstaben oder Bits, welche in eine neue Reihenfolge gebracht werden.*

Genau genommen hat sich durch die Permutation aber nicht viel an unserer Situation geändert: Die Anzahl der veränderten Bits ist gleich, nur ihr Ort hat sich verändert. Um den Lawineneffekt wirklich auszuweiten, müssen wir noch einen Schritt weitergehen.
Eigentlich wissen wir bereits, wie man aus einem einzelnen veränderten Bit in einem Teilblock einen ganzen Teilblock verändern kann: wir wenden erneut die Substitution an. Dafür können die S-Boxen wiederverwendet werden.

Beispiel 5.3. In diesem Beispiel sehen Sie anhand der eingefärbten Felder, wie sich die Veränderung eines einzelnen Bit im letzten Teilblock auf die anderen Teilblöcke auswirken könnte. Die Substitution sorgt dafür, dass der ganze Teilblock des Bits verändert wird. Die Verteilung, oder Permutation, trägt diesen Effekt dann in die anderen Teilblöcke. Wenden wir dann erneut die Substitution an, geschieht in den Teilblöcken, die neue Bits erhalten haben, dasselbe wie bei der ersten Substitution: ein einzelnes verändertes Bit reicht aus, um den ganzen Teilblock zu verändern.



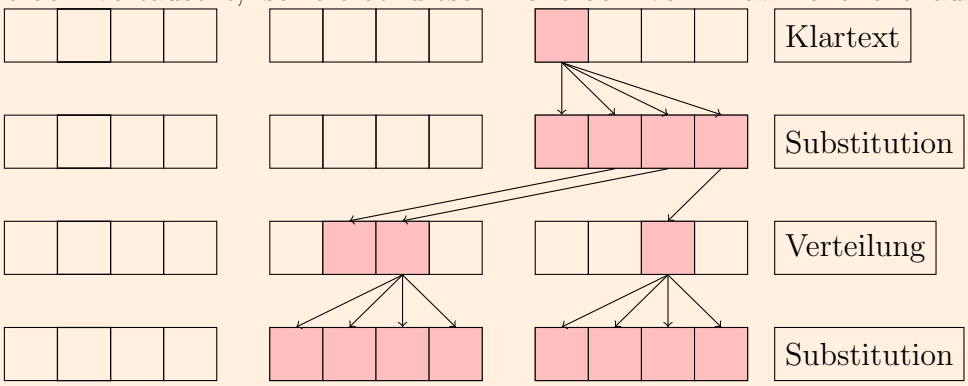
Aber was nun, wenn unser verändertes Bit in einem anderen Teilblock anstatt des letzten ist?

Die Vertauschung, die wir uns gemerkt haben, gilt ja nur für den letzten Teilblock. Wir brauchen also Vertauschungen für alle Teilblöcke und müssen sie uns merken. Dabei müssen wir nun aber aufpassen, dass keine Bits verloren gehen. Jede Position darf nur einmal besetzt werden. Ausserdem müssen wir die Vertauschung immer anwenden. Wir wissen nicht, wann wir möglicherweise unbeabsichtigt ein Zeichen im Klartext verändern und wann nicht.

Aufgabe 5.6. Überlegen Sie: Sind alle Permutationen gleich gut? Wissen Sie vielleicht sogar, wie viele Permutationen es gibt für einen Teilblock einer Länge k, oder den ganzen Text mit n Buchstaben?
Lassen Sie uns üben: Sie erhalten ein Kartenset. Jede Karte steht für ein Zeichen. Legen Sie sich eine Zeichenkette aus, zum Beispiel mit 12 Karten. Decken Sie alle Karten anfangs zu. Die Karten, die verändert werden, decken Sie nun auf. Spielen Sie nun jede Runde aus der Zeichnung 5.3 nach. In der Runde *Verteilung* dürfen Sie selbst aussuchen, wie Sie die betroffenen Karten verteilen. Denken Sie daran: Nur in Teilblöcken, in denen ein verändertes Zeichen ist, verändert sich der ganze Teilblock nach der Substitution. Was stellen Sie fest?

Bei der Erstellung von Permutationsboxen müssen wir darauf achten, den Lawineneffekt auf möglichst viele Teilblöcke auszuweiten. Dies bedeutet für eine geeignete Permutation, dass wir in so vielen Teilblöcken wie möglich mindestens ein Bit aus einem Teilblock unterbringen müssen.

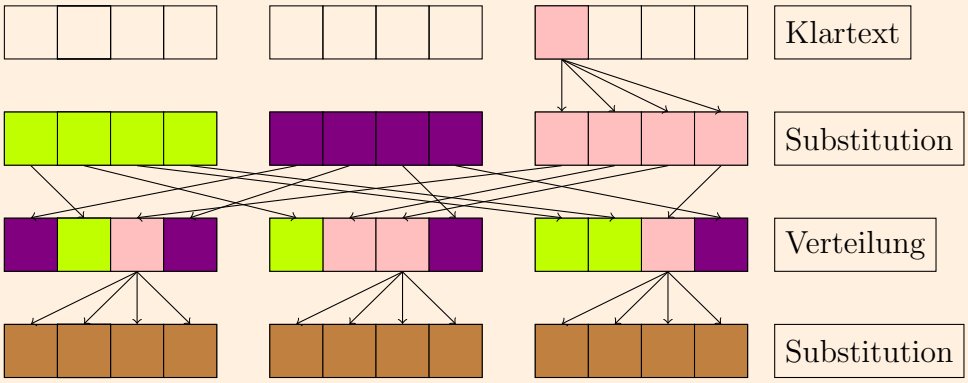
Beispiel 5.4. Betrachten Sie den Unterschied zwischen der Permutation unten und derjenigen aus Beispiel 5.3:
Wird bei der Permutation kein verändertest Bit in einen bestimmten Teilblock vertauscht, so bleibt dieser Teilblock vom Lawineneffekt ausgeschlossen.



Aufgabe 5.7. Wenn Sie eine Permutation für einen Teilblock festgelegt haben, wie lässt sich diese wieder rückgängig machen?

Wir sehen nun, dass wir unsere Substitutionsboxen einfach wiederverwenden können. Dazwischen wenden wir eine Permutation an. Dadurch bildet sich ein **Substitutions-Permutations-Netzwerk**. Man kann sich vorstellen dass, je mehr Substitutions-Permutations-Runden unser Netzwerk hat, desto stärker verfestigt sich der Lawineneffekt. Damit sorgen wir dafür, dass dieselben Teilblöcke nicht mehr denselben Geheimtext haben, da alle Teilblöcke nun miteinander vermischt werden.

Beispiel 5.5. Wir merken uns für jeden Teilblock, an welche Stelle welches Zeichen verschoben wird:
Teilblock 1: [2, 5, 9, 10]
Teilblock 2: [1, 4, 8, 12]
Teilblock 3: [3, 6, 7, 11]
Beachte, dass jede Position nur einmal vorkommen darf. Ansonsten überschreiben wir unsere eigenen Zeichen.



Definition 5.4 (Permutationsbox)**.** *Die Liste von Permutationen, die wir für die Teilblöcke definieren, nennen wir* ***Permutationsbox****. Die Permutationsbox für Beispiel 5.5 wäre also [2, 5, 9, 10, 1, 4, 8, 12, 3, 6, 7, 11]*

Aufgabe 5.8. Sie möchten folgenden Text permutieren: `ICH HEISSE LEA`. Benutzen Sie dazu die Permutationsbox aus dem Beispiel 5.5: [2, 5, 9, 10, 1, 4, 8, 12, 3, 6, 7, 11].
Diese Permutationsbox bedeutet, dass wie im Bild des Beispiels der erste Buchstabe an die zweite Position verschoben wird, der zweite Buchstabe an die fünfte Stelle, der dritte Buchstabe an die neunte Stelle und so weiter.

Aufgabe 5.9. Ein Satz wurde mit folgender Permutationsbox permutiert: [13, 11, 7, 2, 14, 3, 12, 5, 1, 9, 6, 16, 15, 4, 8, 10]. Das Resultat ist folgendes: `IAAR HBNE MNNG AMEE`. Invertieren Sie die Permutation.
Dafür muss der 13. Buchstabe des permutierten Texts an die erste Stelle, der 11. Buchstabe des permutierten Texts an die zweite Stelle, der 7. Buchstabe des permutierten Texts an die dritte Stelle und so weiter.

Beispiel 5.6. Selina hat einen Satz für Sie verschlüsselt, mit einem Substitutions-Permutations-Netzwerk wie in Beispiel 5.5. Der Satz, den sie verschlüsselt, lautet `MAMA MALT LAMA`. Sie benutzt eine Teilblocklänge von 4 und geht wie folgt vor:

1. Selina benutzt die S-Box aus der Tabelle (5.1) unten und wendet sie auf den Text an. Da erhält sie `ALMT TLLT TTTA`
2. Selina verwendet folgende Permutationsbox: [2, 5, 9, 10, 1, 4, 8, 12, 3, 6, 7, 11] und erhält `TATL LTTL MTAT`
3. Selina wendet dieselbe S-Box nochmals an und erhält: `TAMM AMLA AAMM`

Ihr verschlüsselter Satz ist damit `TAMM AMLA AAMM`.

Aufgabe 5.10. Selinas Freundin Elisa zweifelt noch an der Sicherheit der Substitutions-Permutations-Netzwerke. Zeigen Sie Elisa, dass sich in der Tat jeder einzelne Teilblock des Geheimtexts verändert, wenn sich nur ein einzelner Buchstabe im Klartext ändert. Ändern Sie dafür einen einzelnen Buchstaben im Text `MAMA MALT LAMA` (zum Beispiel zu `MAMA MALT LAMM`) und verschlüsseln Sie ihn genau so wie Selina (benutzen Sie dieselbe S-Box und Permutationsbox).

	MM	MA	ML	MT	AM	AL	AT	AA	LM	LA	LT	LL	TM	TA	TL	TT
MM	AATA	LALL	AAAL	MALT	AMLM	TTLL	LLLA	MMAT	AAMT	TLML	ALAT	MMLT	LMMT	AMLT	LLML	MTLA
MA	LLTM	ALMT	MAML	MTAM	AMLL	LATT	TATL	MLLM	LATL	TMLL	TLLT	AALA	MAAM	AMTA	LMTT	ALLA
ML	MTAA	TAAT	TTAM	ALMM	ATAT	MAAL	ATLM	LAAM	LAML	TLMA	LALT	TMAT	TLMT	LLLT	LAAT	ALTM
MT	LTML	LTLM	MATM	LMAT	AMMA	LLMA	AAMM	MTMA	MTTM	AAML	TALT	TTTL	LMLA	MALM	MATA	AMAL
AM	AMAM	TLLL	TAAA	LMLL	ATAM	LLAT	ALTA	TMTM	AATT	TTLM	LTAL	AATM	LLMM	LALM	LMMM	MMLA
AL	MTTT	LMLT	TALA	TATM	TTMT	LATA	AMAA	ATML	ALLL	TAAM	LLTA	MMAL	LTTL	LTMA	ATMA	AMTT
AT	TAML	TAMA	ATAL	LMAL	MMTL	MTML	MAAT	AALM	LLTT	LLAL	MTAT	MTTL	TMML	TTML	TMLT	TALM
AA	AAAM	LAAL	LAMA	ATLL	MTTA	TLTL	LTLL	AAMA	MALL	ATTA	MAMM	LTAA	LAAA	AALT	MLTT	LTAT
LM	LTTT	TMAA	TMTL	MMMA	LTTM	TLLA	LLMT	LTAM	LAMM	ALTL	ATTM	LLLM	TTLT	LMTL	MLLA	LLLL
LA	LMLM	TTTA	MTMT	LTTA	TTTM	LAMT	TTAA	ALAA	LLAM	MMTM	TMMA	TTMA	MTAL	TTAL	TLAA	ALAM
LT	ALTT	TMTT	LTMT	ALAL	ALLT	LMMA	MMMM	MMLM	MLTA	LTLA	ALMA	MMMT	TLAT	TMTA	AMLA	ALML
LL	TMMT	MLAL	MAMT	MLMM	MLAM	LMAM	MTLT	ATAA	TATA	MTLL	MMAA	MMLL	MLML	MLTM	MLMT	MTMM
TM	LLAA	MLMA	TLTM	AATL	LATM	AALL	AAAT	LLTL	AMTM	TMAL	AMMM	TAAL	TTLA	MMTT	MLAA	TTAT
TA	TLLM	TMLA	MMML	MMAM	TAMT	LALA	ATTT	TTTT	TLTT	LMAA	TAMM	MLLT	LMTA	MTLM	LMTM	MLLL
TL	TLMM	ATLA	MMTA	AMAT	ATMT	MMTL	TMAM	TMMM	MATL	TATT	TMLM	TLTA	TLAM	AAAA	AMTL	TTMM
TT	TLAL	MALA	ATMM	LTLT	TALL	MLAT	MAAA	ATLT	LTMM	MATT	MAMA	ATTL	ALLM	AMML	LMML	AMMT

6 Substitutions-Permutations-Netzwerke

Das haben Sie bisher gelernt:

- Sie wissen, was eine Permutation ist.
- Sie wissen, was der Lawineneffekt ist.
- Sie verstehen, wie durch die Kombination von Permutationsboxen und Substitutionsboxen der Lawineneffekt von einem Teilblock auf mehrere Teilblöcke ausgeweitet werden kann.

6.1 Block Cipher

Wir kennen nun Substitutions-Permutations-Netzwerke und verstehen, wie diese funktionieren. Nun möchten wir diese anwenden.
Dabei müssen wir beachten, dass wir bisher nur Klartexte von fixer Länge verschlüsseln können. Diese fixe Länge wird vom Substitutions-Permutations-Netzwerk vorgegeben und nennt sich **Block**.

Definition 6.1 (Substitutions-Permutations-Netzwerk)**.** *Ein Substitutions-Permutations-Netzwerk hat einen Klartext von fixer Länge (ein Block) als Eingabe. Mithilfe von Substitutions- und Permutationsboxen, welche rundenweise abwechselnd angewandt werden, wird die Eingabe chiffriert. Zur Dechiffrierung wird derselbe Vorgang rückwärts angewandt.*

Aufgabe 6.1. Entwerfen Sie Ihr eigenes Netzwerk und tauschen Sie es mit Ihrem Nachbarn. Entschlüsseln Sie gegenseitig geheime Sätze. Sie entwerfen dafür folgende Bestandteile: Eine Substitutionsbox, Permutationen für die einzelnen Teilblöcke sowie eine Zeichnung Ihres Netzwerks.
Geben Sie all diese Informationen zusammen mit Ihrem geheimen Satz Ihrem Banknachbarn.
Beachten Sie: Am Ende aller Runden wird immer nochmals eine Substitution angewandt, damit der Effekt der letzten Permutation auch richtig zum Zuge kommt.

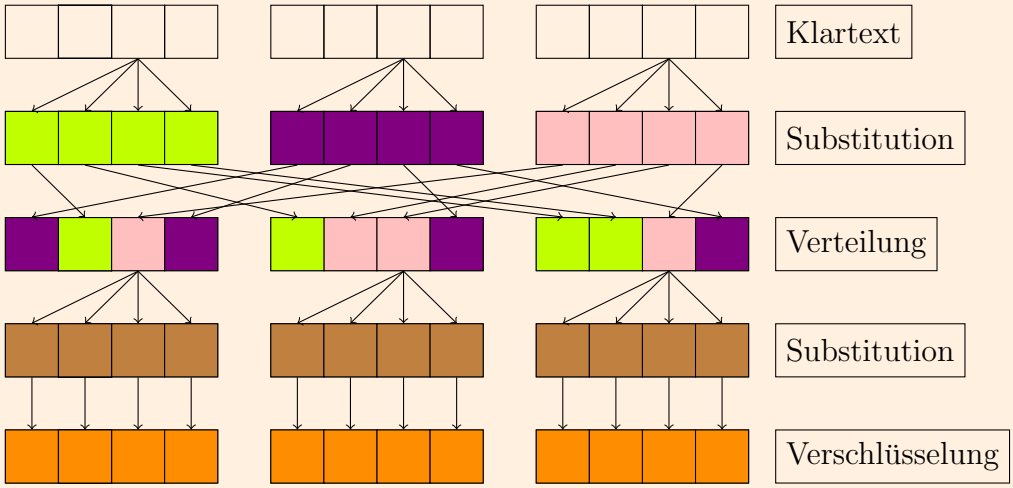
Vielleicht haben Sie bemerkt, dass wir in diesen Substitutions-Permutations-Netzwerken noch gar nicht über Schlüssel nachgedacht haben, wie wir es uns bisher gewohnt waren. Zeit, um nochmals über Sicherheit zu sprechen!

Aufgabe 6.2. Diskussion: Welche Teile des Netzwerks können öffentlich sein, ohne die Sicherheit des Systems zu mindern?
Zur Erinnerung: Kerckhoffs Prinzip besagt, dass nur der Schlüssel eines Kryptosystems geheim sein darf. Was ist hier also der Schlüssel?

Können wir auch einen weniger aufwändigen Schlüssel haben? Die Konstruktion der Substitutions-Permutations-Netzwerke ist ressourcenintensiv, wir hätten lieber einen Schlüssel mit zufälligen Bits.

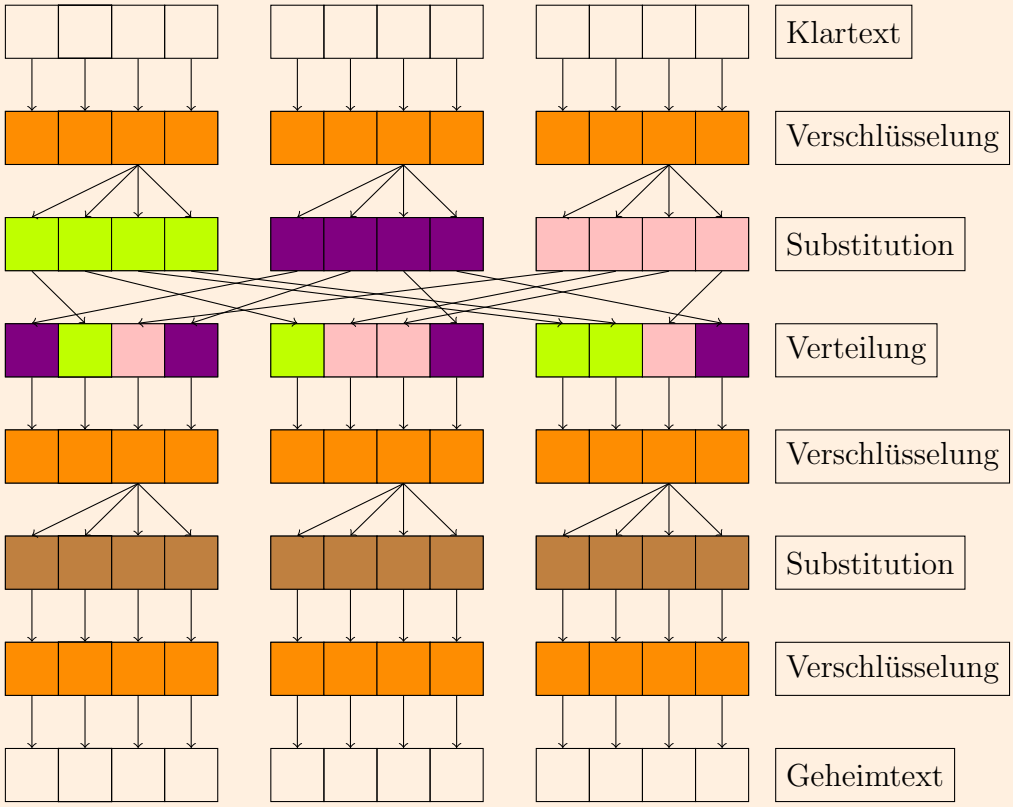
Aufgabe 6.3. Kreativaufgabe: Wo und wie könnte im Netzwerk ein Schlüssel eingesetzt werden, wenn das Netzwerk selbst öffentlich ist?

Beispiel 6.1. Nehmen wir zu unserem bereits bekannten Netzwerk einen Schlüssel dazu. Wir nehmen die Ausgabe der letzten Substitutionsbox und wenden darauf den Schlüssel an. Das Resultat daraus ist dann unser Geheimtext.



Aufgabe 6.4. Diskussion: Ist dieses System sicher? Warum (nicht)? Begründen Sie Ihre Antwort.

Beispiel 6.2. Versuchen wir es mit einem neuen System, welches mehrere Schlüssel verwendet:



Aufgabe 6.5. Diskussion: Ist dieses System sicher? Warum (nicht)? Begründen Sie Ihre Antwort.

Aufgabe 6.6. Verschlüsseln Sie das Wort `Hi` in ASCII **(8 Bit, erstes Bit ist 0, dann wie gewohnt die 7 Bits aus der bereits bekannten Tabelle)**. Sie bekommen drei 16-Bit-Schlüssel, die Sie in dieser Reihenfolge verwenden: [1101'1000 0010'1111, 1110'0110 1111'0010, 0010'1101 1100'1100]
Schlüssel werden als Maskierung angewandt, wie Sie es in Aufgabe 4.2 kennengelernt haben.
Sie bekommen die folgende S-Box, wobei Ihnen die ersten zwei Bits Ihres Teilblocks die Reihe angeben und die letzten zwei Bits die Spalte:

	00	01	10	11
00	1100	0101	1110	1011
01	1010	0010	0001	1101
10	0100	1111	0000	1001
11	0111	0011	0110	1000

Sie bekommen die folgende Permutation: [10, 4, 13, 8, 1, 15, 7, 5, 2, 12, 9, 6, 14, 11, 16, 3]
Wenden Sie das Netzwerk aus Beispiel 6.2 an.

Aufgabe 6.7. Die folgende Bitfolge wurde durch das Substitutions-Permutations Netzwerk aus Aufgabe 6.6 verschlüsselt (selber Schlüssel, Substititonstabelle und Permutation): [0010'0011 0010'1100]
Versuchen Sie, den Geheimtext zu dechiffrieren.
Welche Konsequenzen hätte es, wenn die Substitutionstabelle an der Stelle 1011 anstatt 1001, 1000 stehen würden? Könnten Sie damit verschlüsselten Geheimtext noch immer dechiffrieren?

Definition 6.2 (Block Cipher)**.** *Eine Block Cipher ist ein deterministisches Kryptosystem, welches Klartextblöcke von vordefinierter Länge (Blocklänge) verschlüsseln kann. Übersteigt ein Klartext die Länge eines solchen Blocks, wird er in mehrere Blöcke aufgeteilt.*

Wir wissen nun, wie man eine Block Cipher mit Schlüssel konstruiert. Sie können nun selbstständig Substitutionsboxen und Permutationsboxen erstellen, und daraus ein Substitutions-Permutations-Netzwerk kreieren. Sie verstehen, weshalb dieses Verfahren sicherer ist, als andere Kryptosysteme, welche Sie bereits kennengelernt haben. Sie wissen auch, weshalb ein Substitutions-Permutations-Netzwerk dank Kontextsensitivität gegen die Angriffe von Kasiski und Friedmann resistent ist.
Sollten Sie sich weiter für dieses Thema interessieren, würde es in einem nächsten Schritt darum gehen, auf welche Arten ein längerer Klartext auf Blöcke aufgeteilt und verschlüsselt werden kann. Denn wie Sie bereits wissen, werden auch hier dieselben Klartextblöcke immer noch zu denselben Geheimtextblöcken. Es liegen also viele neue Knobeleien vor Ihnen!

7 Mini-Exkurs: Betriebsmodus einer Block Cipher

Das haben Sie bisher gelernt:

- Sie kennen alle Komponenten eines Substitutions-Permutations-Netzwerks.
- Sie wissen, wie ein Substitutions-Permutations-Netzwerke funktioniert.
- Sie wissen, wie eine Block Cipher mit Schlüssel funktioniert.

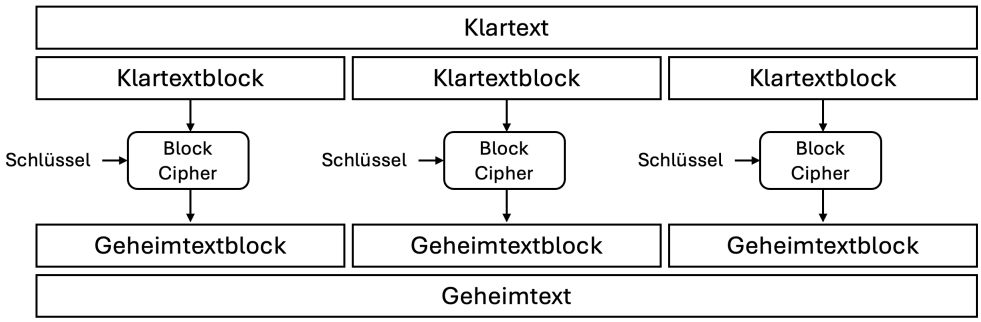
7.1 Mehrere Blöcke verschlüsseln

Bisher haben wir nur Texte limitierter Länge mit einer Block Cipher verschlüsselt. Wenn Sie sich aber an die Definition 6.2 einer Block Cipher erinnern, wird klar: Übersteigt die Länge eines Texts den eines Blocks, so wird der Text in mehrere Blöcke aufgeteilt.
Sollte die Länge des Texts nicht restlos durch die Blocklänge teilbar sein, fügen wir so lange Leerzeichen (oder Nullen bei Binärtexten) hinzu, bis ein Vielfaches der Blocklänge erreicht ist. Diesen Vorgang nennt man **Padding**.

Definition 7.1 (Padding)**.** *Das Ergänzen eines Texts mit Fülldaten. Diese können zusätzliche Informationen enthalten, müssen es aber nicht. In unserem Fall dient Padding dazu, den Text in ein für die Block Cipher geeignetes Format zu bringen, indem die Textlänge zu einem Vielfachen der Blocklänge wird.*

Wie wird nun ein solcher auf Blöcke aufgeteilter Text verschlüsselt? Die einfachste Möglichkeit ist es, die Block Cipher auf jeden Block einzeln anzuwenden. Die Art und Weise, wie wir eine Block Cipher benutzen, um einen Klartext beliebiger Länge zu verschlüsseln, nennen wir **Betriebsmodus**.

Definition 7.2 (Betriebsmodus: Electronic Codebook (ECB))**.** *Beim ECB wird der (bei Bedarf mit Padding versehene) Klartext auf Blöcke aufgeteilt und mit einer Block Cipher verschlüsselt. Die dabei entstehenden Geheimtextblöcke werden aus den einzelnen Blöcken wieder zum gesamten Geheimtext zusammengesetzt.*

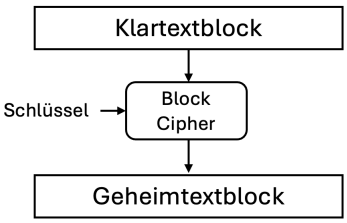


Aufgabe 7.1. Diskussion: Wie weit reicht im ECB-Modus der Lawineneffekt? Wie viele Blöcke sind von einer Veränderung betroffen?

Gleiche Blöcke sehen immer gleich aus als Konsequenz davon, dass der Lawineneffekt auf einen einzelnen Block beschränkt ist.

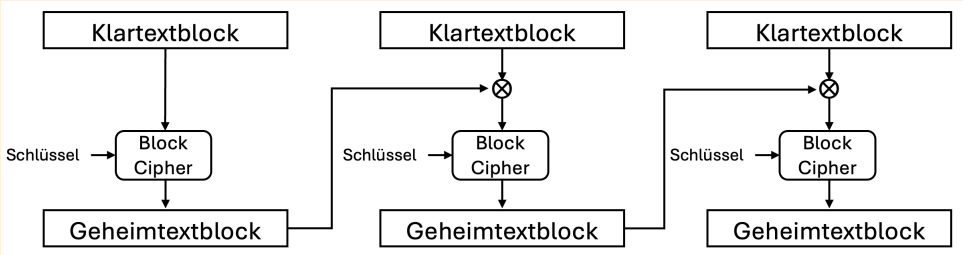
Aufgabe 7.2. Diskussion: Wie könnte der Lawineneffekt auf mehrere Blöcke ausgeweitet werden?

Wir können den Lawineneffekt ausweiten, ohne die Block Cipher dafür zu verändern. Wir können die Block Cipher als sogenannte **Black Box** verwenden, so wie es in Definition 7.2 bereits abgebildet wurde. Das bedeutet, dass wir nicht mehr betrachten müssen, wie diese funktioniert: wir geben ihr als Eingabe einen Text in Blocklänge und einen Schlüssel, und erhalten einen Geheimtext als Ausgabe.

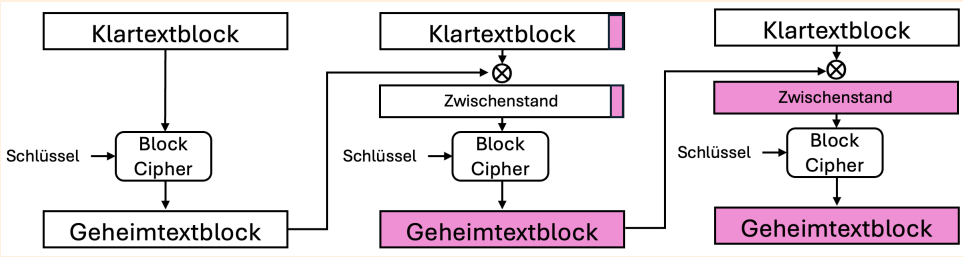


Um den Lawineneffekt erfolgreich auszuweiten, gibt es viele verschiedene Ansätze. Wir zeigen in diesem Exkurs einen: Dabei hängen wir die Blöcke wie Ketten zusammen.

Beispiel 7.1. Wir verschlüsseln von links nach rechts und benutzen den Geheimtext des jeweils vorangegangenen Blocks als Maske für den Klartext des nächsten Blocks.



So sorgen wir dafür, dass die Änderung eines einzelnen Blocks auf alle Blöcke in die Richtung der Verkettung propagiert wird. Dies sorgt dafür, dass der Lawineneffekt auf alle Blöcke rechts vom betroffenen Block ausgeweitet wird.



Bevor wir uns weiter mit der Sicherheit befassen, überlegen wir uns, ob wir überhaupt effizient entschlüsseln können. Den ersten Block kann man wie gewohnt effizient entschlüssln. Dadurch bekommt man die Maske für den zweiten Block. Dies erlaubt uns den folgenden Block effizient zu entschlüsseln, was uns wieder die Maske für den nächsten Block gibt. Wir können somit Block um Block effizient entschlüsseln.

Befassen wir uns nun weiter mit der Sicherheit. Es könnte nämlich sein, dass wir un-

seren Schlüssel wiederverwenden wollen über mehrere Nachrichten hinweg. Dann sehen gleiche Nachrichten weiterhin gleich aus, was Informationen an einen Angreifer verraten könnte.

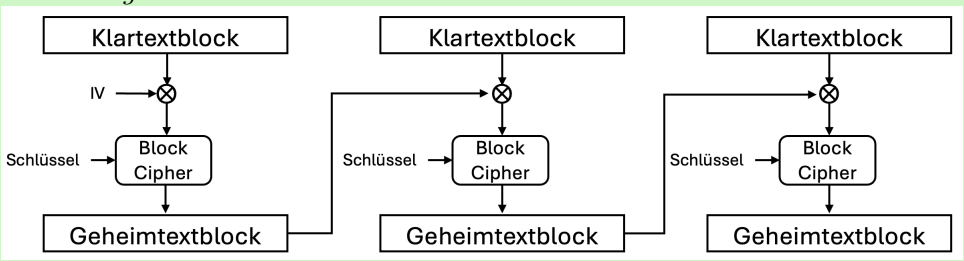
Aufgabe 7.3. Diskussion: In welchen Situationen werden (Teile von) Nachrichten mehrfach ausgetauscht? Überlegen Sie zu zweit.

Da wir bereits festgestellt haben, dass Veränderungen nach rechts weiterpropagiert werden und alle folgenden Geheimtextblöcke betreffen, fügen wir ganz links ein Element hinzu, welches sich bei jeder Nachricht verändert: der **Initialisierungsvektor**, im Folgenden auch IV genannt. Er dient als Maske für den ersten Klartextblock.

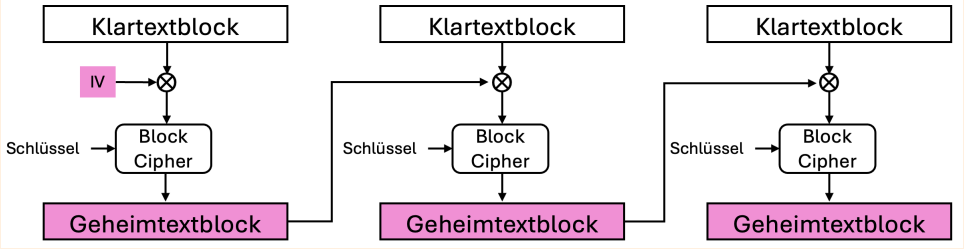
Definition 7.3 (Initialisierungsvektor (IV))**.** *Ein zufälliger Text in Blocklänge, der pro Nachricht neu generiert und als Teil der Nachricht mitgeschickt wird. Er dient dazu, den ersten Klartextblock zu maskieren und sorgt so dafür, dass auch bei wiederverwendetem Schlüssel und Klartext jede Nachricht anders aussieht.*

Wenn wir diesen IV wie in Definition 7.3 erwähnt als Maske für den ersten Klartextblock verwenden, erhalten wir einen sehr bekannten Betriebsmodus für Blockchiffren, der Cipher Block Chaining genannt wird.

Definition 7.4 (Cipher Block Chaining (CBC))**.** *In diesem Betriebsmodus verschlüsseln wir die Klartextblöcke von links nach rechts und benutzen den Geheimtext des vorangegangenen Blocks jeweils als Maske für den Klartext des nächsten Blocks, bevor dieser verschlüsselt wird. Um zu garantieren, dass selbst sich wiederholende Nachrichten mit gleichem Schlüssel nicht gleich aussehen, verwenden wir als Maske für den ersten Block den Initialisierungsvektor. Dieser wird dann als Teil der Nachricht mitgeschickt*



Beispiel 7.2. Sie sehen in der Grafik unten, wie sich bei gleichbleibender Nachricht und Schlüssel ein neuer IV so auswirkt, dass die gesamte Nachricht verändert wird. Der IV wird zur Entschlüsselung der Nachricht benötigt und deshalb mit der Nachricht mitgeschickt. Dies stellt jedoch kein Sicherheitsrisiko dar, da ein Angreifer ohne Schlüssel auch den ersten Geheimtextblock nicht entschlüsseln kann.



Es gibt auch noch viele weitere Betriebsmodi mit verschiedenen Sicherheitsüberlegungen, über die Sie sich weiter informieren könnten.

Anhang A Musterlösungen

Aufgabe 1.1

$\texttt{SI} \xrightarrow{Caesar(J)} \texttt{BR}$

$\texttt{H} \xrightarrow{Caesar(B)} \texttt{I}$

$\texttt{LA} \xrightarrow{Caesar(J)} \texttt{UJ}$

$\texttt{UC} \xrightarrow{Caesar(B)} \texttt{VD}$

$\texttt{H} \xrightarrow{Caesar(J)} \texttt{Q}$

$\texttt{SIHLAUCH} \rightarrow \texttt{BRIUJVDQ}$

Es ändert sich der 2. und 3. Buchstabe im Geheimtext. Somit ist die *Caesar-Verschlüsselung mit Fallunterscheidung* keine 1-lokale Geheimschrift.

Aufgabe 1.2 `WGXAQC`

U	E	B	U	N	G
C	C	C	C	C	C
W	G	D	W	P	I
A	A	U	E	B	U
W	G	X	A	Q	C

Aufgabe 1.3 1.–2. sind 1-lokale Geheimschriften. 3.–4. sind keine 1-lokale Geheimschriften.

Beispielerklärung: Wenn eine Änderung eines Buchstaben im Klartext zur Folge hat, dass sich mehrere Buchstaben im Geheimtext ändern, so ist es keine 1-lokale Geheimschrift.

Aufgabe 2.1 `NENIEE`

Aufgabe 2.2 `NIEREN`

Aufgabe 2.3 Es haben sich der ganze Block, also zwei Buchstaben geändert. Die Blockverschlüsselung ist auch keine 1-lokale Geheimschrift.

Aufgabe 3.1 Der dekodierte Text lautet `DAS IST EIN BINAERTEXT`.

Aufgabe 3.2 Der dekodierte Text lautet `ASCII kann keine Umlaute kodieren...`.

Aufgabe 3.3 Im ersten Schritt muss der Viginère Schlüssel auf den Geheimtext angewendet werden. Der entschlüsselte, in ASCII kodierte Binärstring lautet

1000100 1100101 1110010 0100000 1100010 1101001 1101110 1100001 1100101 1110010
1100101 0100000 1010110 1101001 1100111 1100101 1101110 1100101 1110010 1100101
0100000 1100101 1101110 1110100 1110011 1110000 1110010 1101001 1100011 1101000
1110100 0100000 1100100 1100101 1101101 0100000 1100010 1101001 1110100 1110111
1100101 1101001 1110011 1100101 1101110 0100000 1011000 1001111 1010010 00

. Beachten Sie, dass der Binärstring mit zwei Nullen endet, welche hinzugefügt wurden damit die Länge des Binärstrings einem Vielfachen der Schlüssellänge entspricht. Für die Dekodierung müssen also die letzten zwei Bits nicht beachtet werden.
Der dekodierte Klartext lautet `Der binaere Vigenere entspricht dem bitweisen XOR`.

Aufgabe 4.1 Jedes Bit im 5-Bit Schlüssel hat zwei mögliche Zustände: 0 oder 1. Somit beträgt die Gesamtanzahl von möglichen Schlüsseln $2^5 = 32$.
Jeder Buchstabe im 3-stelligen Zeichenschlüssel hat 26 mögliche Zustände. Somit beträgt die Gesamtanzahl von möglichen Schlüsseln $26^3 = 17576$.

Aufgabe 4.2 Zuerst muss die Lookup Tabelle angewendet werden. Da der Geheimtext bereits verschlüsselt ist, muss die Lookup Tabelle invertiert werden. Mit anderen Worten, die 3-stelligen Bitstrings im Geheimtext müssen mit dem Output der Tabelle verglichen und mit dem Input ersetzt werden. Der entschlüsselte, in ASCII kodierte Binärstring lautet

1001100 1101111 1101111 1101011 1110101 1110000 0100000 1010100 1100001 1100010
1101100 1100101 1110011

und der dekodierte Klartext lautet `Lookup Tables`.

Aufgabe 4.3 Für die erste Position der Tabelle gibt es 8 Möglichkeiten eine Substitution auszuwählen, für die zweite Position sind es noch 7 Möglichkeiten, da bereits eine Substitution ausgewählt wurde und so weiter. Somit gibt es insgesamt $8 \cdot 7 \cdot 6 \cdot \cdots \cdot 1 = 8! = 40320$ Anordnungen.
Für eine Tabelle mit k Einträgen gibt es dementsprechend $k!$ Anordnungen.
Die Anzahl Tabelleneinträge für eine $n-Bit$ Lookup Tabelle entspricht jeweils der Anzahl Möglichkeiten einen n-stelligen Bitstring zu bilden. Wie in Aufgabe 4.1 bereits gesehen, gibt es jeweils 2^n Möglichkeiten.
Kombinieren wir nun die Anzahl Tabelleneinträge (2^n) mit der Anzahl Möglichkeiten, diese anzuordnen $(k!)$ ergeben sich $(2^n)!$ Andordnungsmöglichkeiten für eine n-Bit Lookup Tabelle. Für eine 5-Bit Lookup Tabelle gibt es also $(2^5)! \approx 2.631 \cdot 10^{35}$ Anordnungsmöglichkeiten.

Aufgabe 4.4 Bei dieser Aufgabe geht es zuerst darum herauszufinden, mit welcher $n \times n$ S-Box der Klartext verschlüsselt wurde. Glücklicherweise kennen wir bereits den Klartext der ersten zwei Wörter (*Lieber Michael*) der verschlüsselten Nachricht. In ASCII kodiert ergibt sich der folgende Binärtext

1001100 1101001 1100101 1100010 1100101 1110010 0100000 1001101 1101001 1100011
1101000 1100001 1100101 1101100

welcher dem folgenden Abschnitt des Geheimtextes entspricht

0101010 1000000 1100011 1100001 1101100 1110101 0111111 1000010 0010011 0101101 0011001 0001010 1101100 1100110.

Als nächstes müssen wir versuchen, die S-Box zu rekonstruieren. Da wir die genaue Grösse der S-Box nicht kennen, versuchen wir es zuerst mit der 1×1 S-Box. Die ersten zwei Bits des (10) reichen bereits aus, um eine komplette Lookup Tabelle zu erzeugen wie in Tabelle 6 dargestellt.

Input	Output
0	1
1	0

Tabelle 6: Lookup Tabelle für 1×1 S-Box

Nun müssen wir aber auch überprüfen, ob diese Lookup Tabelle überhaupt konsistent mit dem Rest des Geheimtextes ist. Dabei fällt auf, dass bereits das dritte Bit (0 im Klartext und auch Geheimtext) nicht mit der Lookup Tabelle übereinstimmt (müsste laut Lookup Tabelle eine 1 im Geheimtext sein). Somit können wir die 1×1 S-Box ausschliessen und fahren mit der 2×2 S-Box weiter.
Die 2×2 S-Box lässt sich bereits nach den ersten zwei Blöcken ausschliessen, da zwei unterschiedliche Blöcke des Klartextes (10 und 01) auf denselben Block im Geheimtext (01) abgebildet werden. Damit wäre die S-Box nicht mehr invertierbar und der Klartext könnte nicht mehr eindeutig wiederhergestellt werden.
Auf dieselbe Weise können auch die 3×3 und 5×5 S-Boxen ausgeschlossen werden. Lediglich für die 4×4 S-Box ergibt sich eine konsistente Lookup Tabelle wie in Tabelle 7 dargestellt. Beachten Sie, dass der Klartext Abschnitt keinen Block mit 0111 enthält.

Input	Output	Input	Output	Input	Output	Input	Output
0000	1111	0100	1100	1000	1011	1100	1101
0001	1010	0101	0111	1001	0101	1101	0010
0010	0001	0110	1000	1010	0000	1110	1110
0011	0110	0111	0011	1011	1001	1111	0100

Tabelle 7: Lookup Tabelle für 4×4 S-Box

Da aber alle anderen Blöcke vorkommen und wir somit alle anderen Tabelleneinträge vervollständigen konnten, ergibt sich nur noch eine Möglichkeit für den fehlenden Tabelleneintrag ohne die Invertierbarkeit der S-Box zu verletzen.
Da die S-Box nun bekannt ist kann der restliche Teil der Nachricht auch entschlüsselt werden. Der dekodierte Klartext lautet `Lieber Michael,`*`LF`*` Morgen um Mitternacht am vereinbarten Ort - eine bedeutsame Entdeckung!`.
LF in der Nachricht entspricht einem Zeilenumbruch (ASCII 000′1010).

Aufgabe 4.5 Der kleinste Teilblock, welcher immer gleich verschlüsselt wird, ist immer das kleinste gemeinsame Vielfache zwischen n und k ($kgv(n, k)$) Buchstaben lang.
Zur zweiten Frage gibt es sechs verschiedene Fälle. Die Variable x stellt dabei jeweils immer eine beliebige ganze Zahl dar.

- $k = nx$: Falls k ein Vielfaches von n ist, dann verschlüsselt jede S-Box jeweils einen Teil desselben Buchstaben. Es gibt keine S-Boxen welche Teile von zwei Buchstaben gleichzeitig verschlüsseln. Daher lautet die Antwort 1.
- $2k = nx \wedge k \neq nx$: Falls $2k$ ein Vielfaches von n ist, dann gibt es S-Boxen, welche das Ende eines Buchstabens und den Anfang eines anderen Buchstaben verschlüsseln. Somit kann die Veränderung eines Buchstaben auch die Verschlüsselung des darauffolgenden beeinflussen. Jedoch gibt es keine Buchstaben, bei welchen sowohl der Anfang als auch das Ende teilweise mit dem vorangehenden und nächsten Buchstaben verschlüsselt wird da $2k$ ein Vielfaches von n ist. Daher lautet die Antwort 2.
- $k > n \wedge k \neq nx \wedge 2k \neq nx$: Falls k grösser als n ist aber weder k noch $2k$ ein Vielfaches von n sind, dann gilt dasselbe wie beim letzten Fall wobei es aber auch Buchstaben geben kann, wo sowohl der Anfang als auch das Ende teilweise mit dem vorangehenden bzw. dem nächsten Buchstaben verschlüsselt werden. Somit lautet die Antwort 3.
- $n = kx$: Falls n ein Vielfaches von k ist, dann beginnen alle S-Boxen immer am Anfang eines Buchstaben. Jede S-Box verschlüsselt immer $\frac{n}{k}$ Buchstaben gleichzeitig und vollständig. Es werden nie Teilstücke eines Buchstaben verschlüsselt. Daher lautet die Antwort $\frac{n}{k}$.
- $2n = kx \wedge n \neq kx$: Falls $2n$ ein Vielfaches von k ist, dann gibt es S-Boxen, welche entweder nur den Anfang des letzten oder nur das Ende des ersten Buchstaben innerhalb des S-Box verschlüsseln. Daher lautet die Antwort $\lfloor \frac{n}{k} \rfloor + 1$ ($\lfloor * \rfloor$ ist die Abrundungsfunktion).
- $n > k \wedge 2n \neq kx \wedge n \neq kx$: Falls $2n$ ein Vielfaches von k ist, dann gibt es S-Boxen, welche sowohl nur den Anfang des letzten als auch nur das Ende des ersten Buchstaben innerhalb des S-Box verschlüsseln. Daher lautet die Antwort $\lfloor \frac{n}{k} \rfloor + 2$.

Aufgabe 5.1 Das Wort ist `NENNEN`. Die bekannten Blöcke sind `NN` und `EN`.

Aufgabe 5.2 Wenn genügend Teilblöcke bekannt sind, und darunter auch häufige Buchstabenkombinationen sind, können unter Umständen grosse Teile des Geheimtexts entschlüsselt werden. Dadurch können aufgrund von Sprachmustern eventuell weitere Teilblöcke und Textfragmente entschlüsselt werden. Mit demselben Wissen ist es auch möglich, neue Geheimtexte zu erstellen und so zu fälschen. Wir sprechen von Fälschen, wenn einem Angreifer die Substitutionsboxen nicht bekannt sind, dieser aber trotzdem einen gültigen Geheimtext erstellen kann, indem bekannte Teilblöcke zusammengefügt werden.

Aufgabe 5.3 Wir erhalten folgende Substitutionsboxen:

Klartext	ER	IN	AR	EN	NN	IE	AN	NE
Substitution	RE	AR	AI	NI	EE	IN	EI	IR

Ein möglicher gefälschter Geheimtext wäre: `IRAREIIRIREEREAR`, was im Klartext zu `NEIN, ANNE, NENN ERIN` wird.

Aufgabe 5.4 Der direkte Zusammenhang zwischen Klar- und Geheimtext muss gebrochen werden. Dafür gibt es viele verschiedene Möglichkeiten. Eine Lösung ist natürlich die Permutation einzelner Bits über verschiedene Teilblöcke hinweg, wie sie später eingeführt wird. Diese Aufgabe soll die Lernenden auf diese Überlegungen vorbereiten.

Aufgabe 5.5 Das Ziel ist es, in jedem Teilblock ein verändertes Bit unterzubringen, um dann wieder Substitution darauf anzuwenden.

Aufgabe 5.6 Es sollte sich nach der Permutation in jedem Teilblock ein verändertes Bit befinden.

Aufgabe 5.7 Wenn wir eine Permutationsbox erhalten, setzen wir den Buchstaben, der an der Position des Wertes des aktuellen Elements in der Permutationsbox steht, an den Index des aktuellen Elements in der Permutationsbox. Haben wir zum Beispiel den Text `ERINA` und die Box [3, 1, 5, 4, 2]. Dann setzen wir den Buchstaben, der im Text an der dritten Position steht (erstes Element in der Permutationsbox) an die erste Position (Index des ersten Elements in der Permutationsbox). Das Resultat der gesamten Permutation ist entsprechend: `IEANR`.
Um diese Prozedur umzukehren, müssen wir den Buchstaben an der Position des Index des aktuellen Elements in der Permutationsbox an die Position setzen, die dem Wert des aktuellen Elements in der Permutationsbox entspricht.
Beim Text `IEANR` und der Box [3, 1, 5, 4, 2] muss also der erste Buchstabe (Index des ersten Elements in der Permutationsbox) an die dritte Stelle (erstes Element in der Permutationsbox). Dadurch erhalten wir am Schluss wieder den Text `ERINA`.

Aufgabe 5.8 `EIEICLESHHAS`

Aufgabe 5.9 `ANNA MAG HIMBEEREN`

Aufgabe 5.10 `MLLT LLLM LMAT`

Aufgabe 6.1 Lösungen individuell.

Aufgabe 6.2 Das gesamte Netzwerk ist der Schlüssel und sollte deshalb geheim bleiben.

Aufgabe 6.3 Lösungen individuell. Die Aufgabe soll die Lernenden auf die folgenden Sicherheitsgedanken vorbereiten. Entscheidend ist die Idee, dass der Schlüssel als Maske verwendet wird.

Aufgabe 6.4 Das System ist nicht sicher: Wenn die Eingabe bekannt ist, kann der öffentliche Teil des Netzwerks genutzt werden, um bis zum vorletzten Schritt (Substitution) zu rechnen. Hat ein Angreifer die dazu passende Ausgabe, kann der Schlüssel unter Umständen wiederhergestellt werden.

Aufgabe 6.5 Dieses System ist sicherer, da die Zwischenzustände ohne Kenntnis des Schlüssels nicht erstellt werden können. Die soweit vorgestellten Netzwerke sind aber nicht im kryptographischen Sinne sicher. Viele einfachere Block Ciphers sind angreifbar mit linearer oder differenzieller Kryptoanalyse. Um der Sicherheit modernen Block Ciphers näher zu kommen, müssten die Lernenden in der Lage sein, das Konstrukt sogenannter Feistel-Ciphers zu verstehen. Diese beinhalten allerdings nonlineare Transfomationen und wurden deshalb in diesem Kapitel aussen vor gelassen. Eine verhältnismässig einfache Block Cipher, die den Lernenden gezeigt werden könnte, ist PRESENT: https://www.iacr.org/archive/ches2007/47270450/47270450.pdf https://en.wikipedia.org/wiki/PRESENT https://www.youtube.com/watch?v=hMoWNQbe5SI

Aufgabe 6.6 Hi in ASCII ist: 0x48 0x69 = 01001000 01101001. Verschlüsselt mit dem ersten Schlüssel ergibt dies: 1001 0000 0100 0110. Wir wenden darauf die Substitution an: 1111 1100 1010 0001. Die Permutation gibt uns: 1111000111001010. Dies wird wieder verschlüsselt, mit dem zweiten Schlüssel: 0001 0111 0011 1000. Durch die Substitution erhalten wir: 0101 1101 1011 0100. Mit der letzten Verschlüsselung erhalten wir: 0111000001111000

Aufgabe 6.7 Nach dem Dechiffrieren des Geheimtextes erhält man als Klartext `Du`. Würde man den Eintrag in der Substitutionstabelle abändern, wäre die Substitutionsbox nicht mehr invertierbar (da der Eintrag 1000 bereits einmal vorkommt). Somit könnte Geheimtext nicht mehr eindeutig dechiffriert werden.

Aufgabe 7.1 Nur ein Block ist von der Veränderung betroffen, da die Blöcke unabhängig voneinander verschlüsselt werden.

Aufgabe 7.2 Individuell. Zwischen den Blöcken könnte zum Beispiel irgendeine Art Zusammenhang hergestellt werden. Diese Diskussion ist eine Vorbereitung für den CBC-Modus.

Aufgabe 7.3 Viele standardisierte Nachrichten, wie zum Beispiel Statusmeldungen, Fehlermeldungen oder Logs bei Internetdiensten, sind wiederholt gleich oder sehr ähnlich. Ein Angreifer kann so versuchen, etwas über das Innenleben eines Internetdiensts zu lernen, in dem aktiv wiederholt die gleichen oder gezielt verschiedene Nachrichten ausgelöst werden.

Anhang B Programmieraufgabe: PRESENT Chiffre

In dieser Programmieraufgabe geht es darum, dass Sie selbstständig eine Blockchiffre implementieren um damit Text zu verschlüsseln und entschlüsseln.

Die Blockchiffre welche Sie implementieren werden heisst *PRESENT* und wurde 2007 von Orange Labs, der Ruhr-Universität Bochum und der Dänemarks Technischen Universität entwickelt. Bis zu diesem Zeitpunkt (Januar 2025) konnte die Chiffre nicht gebrochen werden und gilt daher anno dato als sicher.

B.1 Aufbau der PRESENT Chiffre

PRESENT ist ein Beispiel für ein Substitutions-Permutations Netzwerk und besteht aus 31 Runden. Die Chiffre verschlüsselt 64-Bit Blöcke und unterstützt Schlüssel der Grösse 80- oder 128-Bit.

Zu Beginn des Chiffrieralgorithmus werden 32 *Rundenschlüssel* $[K_1, K_2, ..., K_{32}]$ aus dem Schlüssel generiert. Alle Rundenschlüssel haben - genau wie die Blöcke - eine Bitweite von 64.

Jede der 31 Runden beginnt mit der Maskierung des Inputs mit einem der Rundenschlüssel (in der $i$then Runde wird der Rundenschlüssel K_i angewendet). Auf die Maskierung folgt die Anwendung von 16 parallelen 4×4 S-Boxen. Zuletzt wird eine P-Box auf den ganzen Block angewendet. Ganz am Ende der 31 Runden findet nochmals eine Maskierung mit Rundenschlüssel K_{32} statt.

Eine grafische Darstellung befindet sich in Abbildung 4 [1].

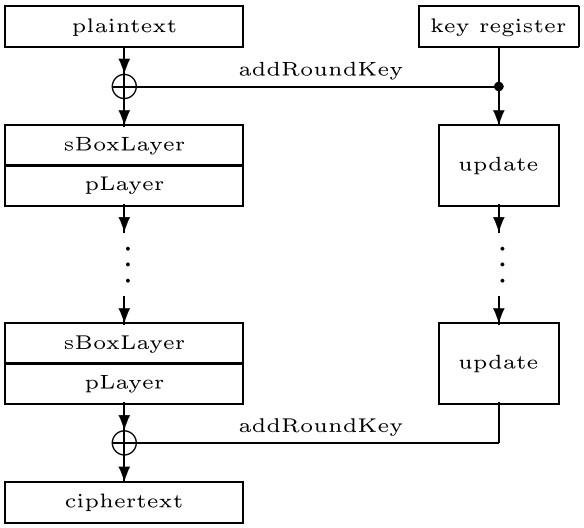


Abbildung 4: Überblick der Blockchiffre *PRESENT*

S-Box Layer Wie bereits in der Einleitung erwähnt, werden 16 4×4 S-Boxen parallel angewendet. Das heisst, die erste S-Box wird auf die ersten 4 Bits (Bits 0 bis 3) des 64-Bit Blocks angewendet, die zweite S-Box auf die nächsten 4-Bits (Bits 4 bis 7) etc. Die Substitutionstabelle aller S-Boxen ist identisch und in Tabelle 8 dargestellt.

[1] Quelle: https://www.iacr.org/archive/ches2007/47270450/47270450.pdf

Input	Output	Input	Output	Input	Output	Input	Output
0000	1100	0100	1001	1000	0011	1100	0100
0001	0101	0101	0000	1001	1110	1101	0111
0010	0110	0110	1010	1010	1111	1110	0001
0011	1011	0111	1101	1011	1000	1111	0010

Tabelle 8: S-Box der *PRESENT* Blockchiffre

P-Box Layer Die P-Box wird auf den kompletten 64-Bit Block angewendet. Die Permutationstabelle der P-Box ist in Tabelle 9 dargestellt.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Tabelle 9: P-Box der *PRESENT* Blockchiffre

Weitere Informationen zu dem Aufbau der *PRESENT* Chiffre finden Sie im folgenden Artikel: https://www.iacr.org/archive/ches2007/47270450/47270450.pdf

Aufgabe B.1. Wie viele Runden der Blockchiffre *PRESENT* sind notwendig, bis die Änderung eines einzelnen Bits im Input alle Bits im Output verändern kann?

B.2 Setup

Für die Implementierung erhalten Sie zwei Dateien: *presentBlockCipher.py* und *test.py*. Die Datei *presentBlockCipher.py* enthält Skeleton Code, das heisst die Struktur auf welcher Sie die *PRESENT* Blockchiffre implementieren. Verschiedene Funktionen sind bereits gegeben, jedoch fehlt für die meisten die Implementierung. Es ist Ihre Aufgabe, diese zu vervollständigen (es sind jeweils Kommentare enthalten, wo Sie Ihren Code hinzufügen müssen). Ebenfalls gegeben sind zwei Python Listen mit der Substitutionstabelle aus Tabelle 8 und der Permutationstabelle aus Tabelle 9, damit Sie diese nicht abtippen müssen.

In der zweiten Datei *test.py* sind Testdaten sowie Validierungsfunktionen enthalten. Sie brauchen diese Datei nicht zu modifizieren. Dieser Code dient Ihnen lediglich dazu, Ihre Implementierungen zu überprüfen.

Zusätzlich können Sie Ihre Implementierungen auch selbst mit Funktionsaufrufen im *presentBlockCipher.py* Skript selbst mit eigenen Funktionsparametern überprüfen.

Aufgabe B.2. Führen Sie das Validierungsskript (*test.py*) aus. Wenn alles funktioniert, sollten Sie in der Konsole eine Übersicht der getesteten Funktionen sehen. Da Sie die Funktionen noch nicht implementiert haben, wird die Validierung für alle Funktionen fehlschlagen. Achtung: Das Validierungsskript muss sich auf derselben Ordnerebene wie *presentBlockCipher.py* befinden.

B.3 S-Box

Aufgabe B.3. Implementieren Sie als erstes die Funktion *sBox*. Diese sollte eine einzelne 4×4 S-Box der *PRESENT* Chiffre anwenden und den Output als Rückgabewert zurückgeben. Sie können dazu die Python Liste *SboxTable*, welche die Tabelle 8 beinhaltet, verwenden. Überprüfen Sie Ihre Implementierung mit dem Validierungsskript.

Aufgabe B.4. Implementieren Sie als nächstes den kompletten S-Box Layer (*sBoxLayer*) mit den 16 parallelen S-Boxen. Dafür werden Ihnen die *set4BitBlock* und die *get4BitBlock* Funktionen hilfreich sein. Diese müssen Sie aber zuerst auch noch selbst implementieren. Sie können die Implementierungen der drei Funktionen wieder mit dem Validierungsskript überprüfen.

Aufgabe B.5. Nachdem Sie den kompletten S-Box Layer für die Verschlüsselung implementiert haben, müssen Sie auch den S-Box Layer für die Entschlüsselung schreiben. Implementieren Sie dazu die Funktion *sBoxLayerInverse*. Die Funktionen *sBoxInverse* und *calculateInverseSboxTable* werden Ihnen dabei helfen, aber Sie müssen diese zuerst noch vervollständigen. Die Funktion *calculateInverseSboxTable* wird einmalig beim Start des Skripts als Initialisierung ausgeführt und Sie können das Resultat in der Python Liste *SboxTableInv* speichern und dann in der Funktion *sBoxInverse* wiederverwenden. Überprüfen Sie anschliessend Ihre Resultate mit dem Validierungsskript.

B.4 P-Box & Maskierung

Aufgabe B.6. Versuchen Sie als nächstes die Funktion *pBoxLayer* zu vervollständigen. Sie können dazu die Python Liste *PboxTable*, welches die Tabelle 9 beinhaltet, verwenden.

Aufgabe B.7. Wie für die S-Box müssen Sie auch für die P-Box eine inverse Funktion für die Entschlüsselung schreiben. Komplementieren Sie die Funktion *pBoxLayerInverse* und nutzen Sie die Funktion *calculateInversePboxTable*. *calculateInversePboxTable* wird analog zu *calculateInverseSboxTable* zu Beginn des Skripts ausgeführt.

Aufgabe B.8. Der letzte Baublock der Chiffre, welcher noch fehlt, ist die Maskierung. Vervollständigen Sie die Funktion *mask*.
Hinweis: Der bitweise XOR Operator (^) könnte Ihnen dabei hilfreich sein.

B.5 Ver- & Entschlüsselung

Aufgabe B.9. Als letztes gilt es, alle Bausteine, welche Sie in der vorangegangenen Aufgaben erstellt haben, zusammenzusetzen. Komplementieren Sie die Verschlüsselung mit der Funktion *encryptBlock*. Die 32 Rundenschlüssel wurden für Sie bereits berechnet. Überprüfen Sie die Funktion mit dem Validierungsskript.

Aufgabe B.10. Nun fehlt lediglich das Entschlüsseln. Vervollständigen Sie dazu auch diese Funktion. Beachten Sie, dass die Rundenschlüssel in umgekehrter Reihenfolge angewendet werden müssen.
Versuchen Sie selbst einen Block Text zu verschlüsseln, tauschen Sie Ihn mit Ihrem Banknachbar aus und versuchen Sie dessen Text zu entschlüsseln.

Aufgabe B.11. Falls Sie den Mini-Exkursion über die Betriebsmodus einer Block Cipher gefolgt sind, können Sie zuallerletzt noch versuchen, das Cipher Block Chaining (CBC) zu implementieren.
Achtung: Diese Aufgabe ist relativ anspruchsvoll.