Verschlüsselung

3

Der klassische Gegenstand der Kryptographie sind Verschlüsselungsverfahren. Solche Verfahren braucht man um die Vertraulichkeit von Nachrichten oder gespeicherten Daten zu schützen. In diesem Kapitel behandeln wir symmetrische Verschlüsselungsverfahren. Wir führen Grundbegriffe ein, die die Beschreibung solcher Verfahren ermöglichen. Außerdem besprechen wir einige Beispiele. Ausführlich werden affin lineare Chiffren diskutiert und ihre Unsicherheit gezeigt.

3.1 Symmetrische Verschlüsselungsverfahren

Wir beginnen mit der Definition von *symmetrischen Verschlüsselungsverfahren*.

Definition 3.1 Ein *symmetrisches Verschlüsselungsverfahren* oder *symmetrisches Kryptosystem* ist ein Tupel (**K**, **P**, **C**, **KeyGen**, **Enc**, **Dec**) mit folgenden Eigenschaften:

1. **K** ist eine Menge. Sie heißt *Schlüsselraum*. Ihre Elemente heißen *Schlüssel*.
2. **P** ist eine Menge. Sie heißt *Klartextraum*. Ihre Elemente heißen *Klartexte*.
3. **C** ist eine Menge. Sie heißt *Chiffretextraum*. Ihre Elemente heißen *Chiffretexte* oder *Schlüsseltexte*.
4. **KeyGen** ist ein probabilistischer Algorithmus. Er heißt *Schlüsselerzeugungsalgorithmus*. Wird er aufgerufen, gibt er einen Schlüssel $K \in \mathbf{K}$ zurück.
5. **Enc** ist ein probabilistischer Algorithmus, der bei Eingabe eines Schlüssels und eines Klartextes einen Chiffretext zurückgibt. Er heißt *Verschlüsselungsalgorithmus* und kann zustandsbehaftet sein.
6. **Dec** ist ein deterministischer Algorithmus, der bei Eingabe eines Schlüssels und eines Chiffretextes einen Klartext zurückgibt. Er heißt *Entschlüsselungsalgorithmus*.
7. Das Verschlüsselungsverfahren entschlüsselt korrekt: gibt der Verschlüsselungsalgorithmus bei Eingabe eines Schlüssels K und eines Klartextes P einen Schlüs-

J. Buchmann, *Einführung in die Kryptographie*, Springer-Lehrbuch,
DOI 10.1007/978-3-642-39775-2_3

seltext C aus, so gibt der Entschlüsselungsalgorithmus bei Eingabe von K und C den Klartext P zurück.

Die gewählten Bezeichnungen entsprechen den englischen Wörtern: **K** wie *key space*, **P** wie *plaintext space*, **C** wie *ciphertext space*, **KeyGen** wie *key generation algorithm*, **Enc** wie *encryption algorithm* und **Dec** wie *decryption algorithm*.

Der Schlüsselraum ist normalerweise endlich und der typische Schlüsselerzeugungsalgorithmus ist Algorithmus 3.1, der einen Schlüssel zufällig und gleichverteilt zurückgibt.

Algorithmus 3.1 (KeyGen)

(Schlüsselerzeugung durch zufällige Wahl)

$$K \xleftarrow{\$} \mathbf{K}$$
$$\textbf{return } K$$

Hierbei bedeutet das Symbol $K \xleftarrow{\$} \mathbf{K}$, dass K zufällig mit Gleichverteilung aus **K** gewählt wird.

Ist der Verschlüsselungsalgorithmus **Enc** deterministisch, so nennen wir die Funktion

$$\mathbf{K} \times \mathbf{P} \to \mathbf{C}, \quad P \mapsto \mathbf{Enc}(K, P) \tag{3.1}$$

Verschlüsselungsfunktion. Analog nennen wir

$$\mathbf{K} \times \mathbf{P} \to \mathbf{C}, \quad P \mapsto \mathbf{Dec}(K, C) \tag{3.2}$$

Entschlüsselungsfunktion.

Verschlüsselungsverfahren werden verwendet, um *Vertraulichkeit* zu ermöglichen, also Klartexte vor unbefugtem Zugriff zu schützen. Dazu benötigen die Kommunikationspartner, sie werden im Folgenden mit Alice und Bob bezeichnet, denselben Schlüssel $K \in \mathbf{K}$. Vor Anwendung des Verschlüsselungsverfahrens tauschen Alice und Bob einen Schlüssel aus und halten ihn dann geheim. Es gibt unterschiedliche Methoden, Schlüssel sicher auszutauschen. Alice und Bob können sich treffen und dabei den Schlüssel einander übergeben. Sie können auch einen Kurier mit der Übermittlung des Schlüssels beauftragen. Wenn Emails verschlüsselt werden sollen, wird der Schlüssel manchmal per SMS verschickt. Dann muss aber sicher sein, dass Angreifer die Mobilfunkverbindung nicht abhören können. Oft verwendet man für den Schlüsselaustausch auch kryptographische Verfahren, wie sie in Kap. 8 besprochen werden.

Wenn Alice und Bob einen gemeinsamen Schlüssel K vereinbart haben, kann Alice vertrauliche Klartexte $P \in \mathbf{P}$ an Bob schicken. Sie berechnet die entsprechenden Schlüsseltexte $C \leftarrow \mathbf{Enc}(K, P)$ und übermittelt sie. Bob erhält die Klartexte als $P =$

$\mathbf{Dec}(K, C)$. Wenn Alice vetrauliche Daten $P \in \mathbf{P}$ speichern möchte, wählt sie einen Schlüssel $K \leftarrow \mathbf{KeyGen}$, speichert $C \leftarrow \mathbf{Enc}(K, P)$ und bewahrt den Schlüssel so auf, dass Unbefugte keinen Zugriff darauf haben. Benötigt sie die Daten wieder, bestimmt sie $P = \mathbf{Dec}(K, P)$.

Vertraulichkeit ist nicht das einzig mögliche Schutzziel. Wer Daten übermittelt, möchte zum Beispiel auch ihre *Integrität* gewährleisten. Integrität bedeutet, dass die Daten bei der Übermittlung nicht verändert wurden. Verschlüsselung gewährleistet keine Integrität. Ein Angreifer kann einfach den Chiffretext ändern. Darum werden später noch weitere kryptographische Verfahren besprochen, die anderen Schutzzielen dienen.

3.2 Verschiebungschiffre

Als erstes Beispiel für ein Verschlüsselungsverfahren beschreiben wir die *Verschiebungschiffre*. Sie wird auch *Caesar-Chiffre* genannt, weil der römische Feldherr Gaius Julius Caesar (100 v.Chr.–44 v.Chr.) sie verwendet haben soll, um seine militärische Korrespondenz geheim zu halten.

Der Klartext-, Chiffretext- und Schlüsselraum der Verschiebungschiffre ist $\Sigma = \{A, B, \ldots, Z\}$. Wir identifizieren die Buchstaben $A, B, \ldots, Z$ gemäß Tab. 3.1 mit den Zahlen $0, 1, \ldots, 25$.

Diese Identifikation ermöglicht es, mit Buchstaben zu rechnen als wären es Zahlen. Der Schlüsselerzeugungsalgorithmus wählt einen Schlüssel zufällig und gleichverteilt. Der Verschlüsselungsalgorithmus 3.2 addiert K modulo 26 zu einem Klartext P. Entsprechend subtrahiert der Entschlüsselungsalgorithmus 3.3 den Schlüssel K modulo 26 von einem Schlüsseltext C.

Algorithmus 3.2 ($\mathbf{Enc}(K, P)$)

(Verschlüsselungsalgorithmus der Verschiebungschiffre)

$C \leftarrow (P + K) \bmod 26$

return C

Algorithmus 3.3 ($\mathbf{Dec}(K, C)$)

(Entschlüsselungsalgorithmus der Verschiebungschiffre)

$P \leftarrow (C - K) \bmod 26$

return P

Tab. 3.1 Entsprechung von Buchstaben und Zahlen

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Gaius Julius Caesar soll die Verschiebungschiffre mit dem Schlüssel $K = 3$ verwendet haben.

Aus der Verschiebungschiffre kann man leicht ein Verschlüsselungsverfahren machen, dessen Klartext- und Chiffretextraum die Menge aller Folgen $\vec{w} = (w_1, w_2, \ldots, w_k)$ mit Einträgen w_i aus Σ ist, $1 \leq i \leq k$. Der Schlüsselraum ist wieder $\mathbb{Z}_{26}$. Der Verschlüsselungsalgorithmus **Enc** ersetzt jeden Buchstaben w_i durch $w_i + K \bmod 26$, $1 \leq i \leq k$. Der Entschlüsselungsalgorithmus macht diese Operation rückgängig. Dieses Verschlüsselungsverfahren nennt man ebenfalls Verschiebungschiffre.

Beispiel 3.1 Wendet man die Verschiebungschiffre mit Schlüssel $K = 5$ auf das Wort KRYPTOGRAPHIE an, so erhält man PWDUYTLWFUMNJ.

Der Schlüsselraum der Verschiebungschiffre hat nur 26 Elemente. Chiffretexte der Verschiebungschiffre kann man entschlüsseln, indem man alle möglichen Schlüssel ausprobiert und prüft, welcher Schlüssel einen sinnvollen Text ergibt. Man erhält auf diese Weise nicht nur den Klartext aus dem Chiffretext, sondern auch den verwendeten Schlüssel. Diesen Angriff nennt man *vollständige Suche*. Er wird in Abschn. 3.4.2 genauer besprochen.

3.3 Asymmetrische Verschlüsselungsverfahren

Die in diesem Kapitel behandelten Verschlüsselungsverfahren heißen symmetrisch, weil derselbe Schlüssel zur Ver- und Entschlüsselung verwendet wird. Dies hat zur Folge, dass Schlüssel auf sichere Weise ausgetauscht werden müssen. In großen Kommunikationsnetzen ist das sehr aufwändig, wenn nicht sogar unmöglich. Daher wurden seit Ende der 1970ziger Jahre *asymmetrische Verschlüsselungsverfahren* entwickelt. Sie werden in Kap. 8 beschrieben. Bei solchen Verfahren entfällt die Notwendigkeit des Schlüsselaustauschs, weil die Schlüssel zum Verschlüsseln und Entschlüsseln verschieden sind und die Schlüssel zum Entschlüsseln nicht aus den Schlüsseln zum Verschlüsseln berechnet werden können. Die Verschlüsselungsschlüssel können also öffentlich gemacht werden, während die Entschlüsselungsschlüssel geheim gehalten werden. Alice kann ein asymmetrisches Verschlüsselungsverfahren folgendermaßen verwenden, um eine vertraulichen Klartext P an Bob zu schicken. Sie besorgt sich Bobs öffentlichen Verschlüsselungsschlüssel, verschlüsselt P damit und schickt den Chiffretext C an Bob. Bob verwendet seinen geheimen Entschlüsselungsschlüssel, um den Klartext P aus dem Chiffretext C zu rekonstruieren.

Asymmetrische Verschlüsselungsverfahren machen aber symmetrische nicht überflüssig, weil die bekannten asymmetrischen Verfahren viel langsamer sind als die besten symmetrischen. Darum wird in der Praxis *hybride Verschlüsselung* eingesetzt. Sie funktioniert so: Alice will eine vertrauliche Nachricht P an Bob schicken. Sie erzeugt einen Schlüssel K für ein symmetrisches Verschlüsselungsverfahren. Sie verwendet ein asymmetrisches Verfahren, um K mit Bobs öffentlichem Schlüssel zu K' zu verschlüsseln.

Danach verschlüsselt sie P symmetrisch unter Verwendung des Schlüssels K und schickt K' und C an Bob. Bob kann K' zu K entschlüsseln, wenn er seinen asymmetrischen Entschlüsselungsschlüssel verwendet. Anschließend kann er mit Hilfe von K den Schlüsseltext C zum Klartext P entschlüsseln. Bei diesem Verfahren muss der symmetrische Schlüssel also nicht vorher ausgetauscht werden.

Symmetrische Verschlüsselungsverfahren werden auch *Private-Key-Kryptosysteme* genannt. Asymmetrische Verschlüsselungsverfahren heißen auch *Public-Key-Kryptosysteme*.

3.4 Sicherheit von Verschlüsselungsverfahren

Verschlüsselungsverfahren sollen gespeicherte oder gesendete Daten geheim halten. Um entscheiden zu können, ob ein Verschlüsselungsverfahren dies tatsächlich tut, ist es nötig, die *Sicherheit* von Verschlüsselungsverfahren zu definieren. In diesem Abschnitt diskutieren wir den Begriff der Sicherheit von Verschlüsselungsverfahren zunächst informell. Eine mathematische Modellierungen der Sicherheit symmetrischer Kryptosysteme erläutern wir in Kap. 4. Die entsprechenden Modelle für asymmetrische Verfahren beschreibt Kap. 8.

Soll Geheimhaltung definiert werden, müssen zwei Fragen beantwortet werden: Was ist das Ziel eines Angreifers und welche Mittel stehen dem Angreifer zur Verfügung, um sein Ziel zu erreichen? Diese Fragen werden in den folgenden Abschnitten behandelt.

3.4.1 Angriffsziele

Angreifer eines Kryptosystems wollen aus Chiffretexten möglichst viel über die entsprechenden Klartexte lernen. Dazu können sie versuchen, geheime Schlüssel zu erfahren. Wer sie kennt, kann alle damit erzeugten Schlüsseltexte entschlüsseln. Angreifer können sich aber auch darauf beschränken, einzelne Nachrichten zu entschlüsseln ohne den entsprechenden Entschlüsselungsschlüssel zu ermitteln. Oft genügt es dem Angreifer sogar, nicht die gesamte Nachricht zu entschlüsseln, sondern nur eine spezielle Information über die Nachricht in Erfahrung zu bringen. Dies wird an folgendem Beispiel deutlich.

Beispiel 3.2 Ein Bauunternehmen macht ein Angebot für die Errichtung eines Bürogebäudes. Das Angebot soll vor der Konkurrenz geheimgehalten werden. Die wichtigste Information für die Konkurrenz ist der Preis, der im Angebot genannt wird. Die Details des Angebots sind weniger wichtig. Ein Angreifer kann sich also darauf beschränken, den Preis zu erfahren und muss nicht den gesamten Chiffretext entschlüsseln. Das ist möglicherweise einfacher.

Sicherheit liefert ein Verschlüsselungsverfahren also erst dann, wenn ein Angreifer aus dem Schlüsseltext nichts über den entsprechenden Klartext lernen kann.

3.4.2 Angriffstypen

Als Nächstes diskutieren wir die Fähigkeiten, die Angreifer eines Kryptosystems haben können. Zwei Szenarien sind möglich. Im ersten Szenario schickt Alice Nachrichten an Bob, die mit einem symmetrischen Kryptosystem verschlüsselt wurden. Dafür haben Alice und Bob vorher einen Schlüssel ausgetauscht, der nur ihnen bekannt ist. Im zweiten Szenario verschlüsselt Alice gespeicherte Daten. Solche Daten können zum Beispiel in einem Cloud-Speicher liegen. Den entsprechenden Schlüssel bewahrt Alice an einem sicheren Ort auf, so dass nur sie Zugang dazu hat.

Bei unserer Sicherheitsdiskussion gehen wir davon aus, dass folgende Sicherheitsannahmen erfüllt sind:

1. Angreifer kennen das verwendete Verschlüsselungsverfahren.
2. Angreifer haben Zugang zu den Chiffretexten.
3. Der Schlüsselaustausch ist sicher, d. h. Angreifer haben keine Möglichkeit, während des Schlüsselaustausch etwas über den geheimen Schlüssel zu erfahren.
4. Die Schlüsselaufbewahrung ist sicher, d. h. Angreifer haben keinen Zugang zu den geheim aufbewahrten Schlüsseln.

Wir diskutieren diese Annahmen nun im Einzelnen.

Statt von einem öffentlich bekannten Verschlüsselungsverfahren auszugehen, könnte man auch versuchen, das Verschlüsselungsverfahren geheimzuhalten und so die Sicherheit weiter zu steigern. Dies wird zum Beispiel bei manchen militärischen Anwendungen versucht. In der Welt des Internets ist das aber nicht möglich. Sicherheitsprotokolle, zum Beispiel das Transport Layer Security Protokoll (TLS), sind standardisiert, damit die beteiligten Systeme (Computer, Smartphones etc.) mit den Protokollen umgehen können. Verwenden die Protokolle Verschlüsselungsverfahren, wie das bei TLS der Fall ist, legen die Standards die erlaubten Verschlüsselungsverfahren fest. Die Standards und die Verschlüsselungsverfahren sind öffentlich. Aber selbst wenn Verschlüsselungsverfahren nicht in öffentlichen Standards bekannt gemacht werden, ist unklar, wie sie geheim gehalten werden können. Ein Angreifer hat viele Möglichkeiten, zu erfahren, welches Kryptosystem verwendet wird. Er kann verschlüsselte Nachrichten abfangen und daraus Rückschlüsse ziehen. Er kann beobachten, welche technischen Hilfsmittel zur Verschlüsselung benutzt werden. Jemand, der früher mit dem Verfahren gearbeitet hat, gibt die Information preis. Es ist also unklar, ob es gelingen kann, das verwendete Kryptosystem wirklich geheimzuhalten.

Die zweite Annahme besagt, dass Angreifer Zugang zu Chiffretexten haben. Verschlüsselungsverfahren sollen ja gerade so beschaffen sein, dass aus Chiffretexten keine Informationen über die zugehörigen Klartexte gewonnen werden können. Sicherheit von Verschlüsselungsverfahren bedeutet also, dass diese Eigenschaft erfüllt ist, wenn die Angreifer Zugang zu Chiffretexten haben.

Tab. 3.2 Mindestgröße des Schlüsselraums

Schutz bis	Mindestgröße
2020	2^{96}
2030	2^{112}
2040	2^{128}
für absehbare Zukunft	2^{256}

Gemäß der dritten Einnahme haben Angreifer keine Möglichkeit, während des Schlüsselaustausch Informationen über den Schlüssel zu gewinnen. Diese Annahme bedeutet nicht, dass es leicht ist, vertraulich Schlüssel auszutauschen. Tatsächlich ist Schlüsselaustausch ein eigenes Thema der Kryptographie. Die Sicherheit des Schlüsselaustausches ist aber nicht Teil der Sicherheit eines Kryptosystems, sondern muss unabhängig davon gewährleistet werden. Entsprechendes gilt für den Schutz vertraulicher privater Schlüssel.

Wir diskutieren jetzt die wichtigen Typen von Angriffen auf Kryptosysteme.

Ciphertext-Only-Angriff

Bei einem Ciphertext-Only-Angriff kennt der Angreifer nichts anderes als einen Chiffretext. Aus diesem Chiffretext versucht er Informationen über den verwendeten Schlüssel oder den entsprechenden Klartext abzuleiten.

Ein einfacher Ciphertext-Only-Angriff ist *vollständige Suche*. Dabei entschlüsselt der Angreifer den Schlüsseltext mit allen Schlüsseln aus dem Schlüsselraum. Unter den wenigen sinnvollen Texten, die sich dabei ergeben, befindet sich der gesuchte Klartext. Welcher das ist, kann zum Beispiel mithilfe von Kontextinformationen ermittelt werden. Diese Methode führt zum Erfolg, wenn der Schlüsselraum des Verschlüsselungsverfahrens klein genug ist. Dies ist offensichtlich bei der Caesar-Chiffre der Fall. Deren Schlüsselraum hat ja nur 26 Elemente. Vollständige Suche funktioniert heutzutage aber auch beim Data Encryption Standard DES, der von 1976 bis 2001 ein wichtiger amerikanischer Verschlüsselungsstandard war (siehe Kap. 5). Sein Schlüsselraum hat die Größe 2^{56}. DES wurde 1998 von dem speziell für diesen Zweck gebauten Computer Deep Crack der Electronic Frontier Foundation in 56 Stunden mittels vollständiger Durchsuchung gebrochen. Heute kann DES von jedem handelsüblichen PC in noch kürzerer Zeit gebrochen werden.

Der Schlüsselraum eines sicheren Verschlüsselungsverfahrens muss also eine Mindestgröße haben. Diese Größe hängt von der Leistungsfähigkeit der jeweils vorhandenen Computer ab. Nach dem Moorschen Gesetz verdoppelt sich die Anzahl der Operationen, die Computer pro Zeiteinheit ausführen können, alle 18 Monate. Die Mindestgröße der Schlüsselräume muss also entsprechend vergrößert werden. Tab. 3.2 zeigt, wie sich diese Mindestgröße im Laufe der Zeit entwickelt. Die Werte wurden im Rahmen des EU-Projekts ECRYPT II ermittelt (siehe [73]).

Andere Ciphertext-Only-Angriffe nutzen statistische Eigenschaften der Sprache, aus der die Klartexte stammen. Wird zum Beispiel die Verschiebungschiffre zum Verschlüsseln benutzt, so wird bei festem Schlüssel jedes Klartextzeichen durch das gleiche Schlüsseltextzeichen ersetzt. Das häufigste Klartextzeichen entspricht also dem häufigsten

Schlüsseltextzeichen, das zweithäufigste Klartextzeichen entspricht dem zweithäufigsten Schlüsseltextzeichen usw. Genauso wiederholt sich die Häufigkeit von Zeichenpaaren, Tripeln usw. Diese statistischen Eigenschaften können ausgenutzt werden, um Chiffretexte zu entschlüsseln und den Schlüssel zu finden. Weitere Beispiele für solche statistischen Angriffe finden sich in [35], [72] und in [6].

Angreifer, die nur Ciphertext-Only-Angriffe ausführen können, heißen *passive Angreifer*. Angreifer, die einen der im Folgenden beschriebenen Angriffe ausführen können, heißen *aktive Angreifer*.

Known-Plaintext-Angriff

Bei einem Known-Plaintext-Angriff kennt der Angreifer Klartexte und die zugehörigen Schlüsseltexte. Dies ermöglicht es ihm in vielen Situationen, in denen Ciphertext-Only-Angriffe nicht zum Erfolg führen, seine Angriffsziele zu erreichen. Das nächste Beispiel zeigt, dass dieser Angriffstyp realistisch ist.

Beispiel 3.3 Im zweiten Weltkrieg verwendeten die Deutschen die ENIGMA-Verschlüsselungsmaschine, um Nachrichten an deutsche U-Boote zu verschlüsseln. Solche Nachrichten waren häufig stereotyp abgefasst und enthielten viele leicht zu erratende Textabschnitte wie zum Beispiel OBERKOMMANDODERWEHRMACHT. Die Alliierten kannten also diese Klartexte und die entsprechenden Schlüsseltexte. Außerdem wurden auch Routinemeldungen verschlüsselt, wie Wetterberichte, die jeden Morgen pünktlich zur selben Zeit und vom selben Ort gesendet wurden. Diese Routinemeldungen waren öffentlich. Die Allierten brauchten also nur die entsprechenden Chiffretexte abzuhören, um Known-Plaintext-Angriffe anwenden zu können.

Known-Plaintext-Angriffe können erfolgreich gegen deterministische Verschlüsselungsverfahren eingesetzt werden. Angenommen nämlich, der Angreifer kennt ein Klartext-Schlüsseltext-Paar (P, C). Wenn er später den Schlüsseltext C abfängt, weiß er, dass der entsprechende Klartext P ist, jedenfalls solange derselbe Schlüssel verwendet wird. Das verwendete Verschlüsselungsverfahren ist ja deterministisch, das heißt aus demselben Klartext und Schlüssel ergibt sich immer derselbe Chiffretext. Diese Angriffsmöglichkeit ist auch der Grund dafür, dass deterministische Verschlüsselungsverfahren immer unsicher sind.

Known-Plaintext-Angriffe können besonders erfolgreich gegen alle affin-linearen Chiffren eingesetzt werden, wie in Abschn. 3.19 gezeigt wird.

Chosen-Plaintext-Angriff

Bei diesem Angriff kann der Angreifer sogar selbst gewählte Klartexte verschlüsseln. Man kann sich das zum Beispiel so vorstellen:

Beispiel 3.4 Im zweiten Weltkrieg kannten die Alliierten die Nachrichten, die von deutschen Schiffen gesendet wurden, wenn der Abwurf einer Wasserbombe beobachtet wurde.

Die Nachrichten enthielten den Zeitpunkt und den Abwurfort. Um die jeweils aktuellen Schlüssel zu finden, warfen die Alliierten also Wasserbomben ab, ohne deutsche Schiffe zu treffen. Sie fingen die verschlüsselten Meldungen der U-Boote ab und hatten so Chiffretexte zu selbst gewählten Nachrichten, weil sie ja den Abwurfort und den Abwurfzeitpunkt bestimmt hatten. Diese Kombinationen aus Klar- und Chiffretexten halfen den Alliierten, geheime Schlüssel zu finden.

Wir geben noch ein anderes Beispiel.

Beispiel 3.5 Bob ist unachtsam und vergisst sein Smartphone in einem Cafe. Mit dem Smartphone kann er seine vertraulichen Nachrichten verschlüsseln. Ein Angreifer findet das Smartphone. Er kann den geheimen Schlüssel nicht auslesen, weil er unzugänglich im Smartphone gespeichert ist. Er kann das Smartphone aber benutzen, um sich selbst verschlüsselte Nachrichten seiner Wahl zu schicken. Er verwendet diese, um den geheimen Schlüssel zu bestimmen, wie etwa in der Methode aus Abschn. 3.19. Danach legt er das Gerät zurück. Bob kommt zurück und ist froh, sein Smartphone wieder zu haben. Aber jetzt kennt der Angreifer den geheimen Schlüssel und kann Bobs geheime Nachrichten entschlüsseln.

In Beispiel 3.5 hat der Angreifer sogar noch stärkere Möglichkeiten. Er kann jeden Klartext, den er wählt, von seinen vorherigen Berechnungen abhängig machen. Es könnte ja sein, dass er schon einige Informationen über den geheimen Schlüssel gesammelt hat und dass er für weitere Informationen Klartexte von besonderer Gestalt verschlüsseln muss. Der Angriff heißt dann *adaptiver Chosen-Plaintext-Angriff*. Normalerweise meint man solche Angriffe, wenn man von Chosen-Plaintext-Angriffen spricht.

Chosen-Plaintext-Angriffe sind gerade bei deterministischen Public-Key-Verschlüsselungsverfahren leicht möglich. Jeder kennt den öffentlichen Verschlüsselungsschlüssel und kann darum beliebige Klartexte verschlüsseln. Wenn ein Angreifer zum Beispiel einen Schlüsseltext C kennt und weiß, dass der entsprechende Klartext P entweder „ja“ oder „nein“ ist, muss er nur „ja“ zu C_{ja} und „nein“ zu C_{nein} verschlüsseln. Wenn $C = C_{\text{ja}}$ ist, dann ist $P =$ „ja“. Andernfalls ist $P =$ „nein“. Hier zeigt sich wieder das Risiko deterministischer Verschlüsselung.

Chosen-Ciphertext-Angriff

Bei diesem Angriff kann der Angreifer selbst gewählte Schlüsseltexte entschlüsseln, ohne den Entschlüsselungsschlüssel zu kennen. Beispiel 3.5 beschreibt ein Szenario, in dem das möglich ist. In der dort beschriebenen Situation kann der Angreiferer das Smartphone auch benutzen, um selbst gewählte Chiffretexte zu entschlüsseln. Abschn. 8.4.6 zeigt, wie ein solcher Angriff bei einem Public-Key-Verfahren erfolgreich sein kann.

3.5 Alphabete und Wörter

Für die weitere Beschreibung von Kryptosystemen sind noch einige Begriffe nötig, die in diesem Abschnitt eingeführt werden. Um Texte aufzuschreiben, braucht man Zeichen aus einem Alphabet. Unter einem *Alphabet* verstehen wir eine endliche, nicht leere Menge Σ. Die *Länge* von Σ ist die Anzahl der Elemente in Σ. Die Elemente von Σ heißen *Zeichen*, *Buchstaben* oder *Symbole* von Σ.

Tab. 3.3 Der ASCII-Zeichensatz

0	NUL	1	SOH	2	STX	3	ETX
4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT
12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3
20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC
28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	“	35	#
36	$	37	%	38	&	39	’
40	(	41	)	42	*	43	+
44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3
52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;
60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C
68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K
76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S
84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[
92		93	]	94	^	95	_
96	‘	97	a	98	b	99	c
100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k
108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s
116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{
124	|	125	}	126	~	127	DEL

Beispiel 3.6 Ein bekanntes Alphabet ist

$$\Sigma = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}, \mathrm{G}, \mathrm{H}, \mathrm{I}, \mathrm{J}, \mathrm{K}, \mathrm{L}, \mathrm{M}, \mathrm{N}, \mathrm{O}, \mathrm{P}, \mathrm{Q}, \mathrm{R}, \mathrm{S}, \mathrm{T}, \mathrm{U}, \mathrm{V}, \mathrm{W}, \mathrm{X}, \mathrm{Y}, \mathrm{Z}\}.$$

Es hat die Länge 26.

Beispiel 3.7 In der Datenverarbeitung wird das Alphabet $\mathbb{Z}_2$ verwendet. Es hat die Länge 2.

Beispiel 3.8 Ein häufig benutztes Alphabet ist der ASCII-Zeichensatz. Dieser Zeichensatz samt seiner Kodierung durch die Zahlen von 0 bis 127 findet sich in Tab. 3.3.

Da Alphabete endliche Mengen sind, kann man ihre Zeichen mit nicht negativen ganzen Zahlen identifizieren. Hat ein Alphabet die Länge m, so identifiziert man seine Zeichen mit den Zahlen in $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$. Für das Alphabet $\{A, B, \ldots, Z\}$ und den ASCII-Zeichensatz haben wir das in den Tab. 3.1 und 3.3 dargestellt. Wir werden daher meistens das Alphabet $\mathbb{Z}_m$ verwenden, wobei m eine natürliche Zahl ist.

Die folgenden Definition verwendet endliche Folgen. Ein Beispiel für eine endliche Folge ist

$$(2, 3, 1, 2, 3).$$

Sie hat fünf Folgenglieder. Das erste ist 2, das zweite 3 usw. Manchmal schreibt man die Folge auch als

$$23123.$$

Aus formalen Gründen definiert man auch die *leere Folge* (). Sie hat null Folgenglieder.

Definition 3.2 Sei Σ ein Alphabet.

1. Als *Wort* oder *String* über Σ bezeichnet man eine endliche Folge von Zeichen aus Σ einschließlich der leeren Folge, die mit ε bezeichnet und *leeres Wort* genannt wird.
2. Die *Länge* eines Wortes $\vec{w}$ über Σ ist die Anzahl seiner Zeichen. Sie wird mit $|\vec{w}|$ bezeichnet. Das leere Wort hat die Länge 0.
3. Die Menge aller Wörter über Σ einschließlich des leeren wird mit Σ^* bezeichnet.
4. Sind $\vec{v}, \vec{w} \in \Sigma^*$, dann ist der String $\vec{v}\vec{w} = \vec{v} \circ \vec{w}$, den man durch Hintereinanderschreiben von $\vec{v}$ und $\vec{w}$ erhält, die *Konkatenation* von $\vec{v}$ und $\vec{w}$. Insbesondere ist $\vec{v} \circ \varepsilon = \varepsilon \circ \vec{v} = \vec{v}$.
5. Ist n eine nicht negative ganze Zahl, dann bezeichnet Σ^n die Menge aller Wörter der Länge n über Σ.

Wie in Übung 3.5 gezeigt wird, ist $(\Sigma^*, \circ)$ eine Halbgruppe. Deren neutrales Element ist das leere Wort.

Beispiel 3.9 Ein Wort über dem Alphabet aus Beispiel 3.6 ist COLA. Es hat die Länge vier. Ein anderes Wort über Σ ist COCA. Die Konkatenation von COCA und COLA ist COCACOLA.

3.6 Permutationen

Um eine sehr allgemeine Klasse von Verschlüsselungsverfahren, die Blockchiffren (siehe Abschn. 3.7), zu charakterisieren, wird der Begriff der Permutation benötigt.

Definition 3.3 Sei X eine Menge. Eine *Permutation* von X ist eine bijektive Abbildung $f : X \to X$. Die Menge aller Permutationen von X wird mit $S(X)$ bezeichnet.

Beispiel 3.10 Sei $X = \mathbb{Z}_5$. Wir erhalten eine Permutation von X, wenn wir jedem Element von X in der oberen Zeile der folgenden Matrix die Ziffer in der unteren Zeile zuordnet.

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 4 & 3 & 5 & 0 \end{pmatrix}$$

Auf diese Weise lassen sich Permutationen endlicher Mengen immer darstellen. Zur Vereinfachung können wir die erste Zeile auch weglassen. Dann werden Permutation von $\mathbb{Z}_n$ als Folgen der Länge n dargestellt.

Die Menge $S(X)$ aller Permutationen von X zusammen mit der Hintereinanderausführung bildet eine Gruppe, die im allgemeinen nicht kommutativ ist. Ist n eine natürliche Zahl, dann bezeichnet man mit S_n die Gruppe der Permutationen der Menge $\mathbb{Z}_n$.

Beispiel 3.11 Die Gruppe S_2 hat die Elemente $\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$.

Theorem 3.1 *Die Gruppe S_n hat genau $n! = 1 * 2 * 3 * \cdots * n$ Elemente.*

Beweis Wir beweisen die Behauptung durch Induktion über n. Offensichtlich hat S_1 nur ein Element. Angenommen, S_{n-1} hat $(n-1)!$ viele Elemente. Betrachte jetzt Permutationen der Menge $\{1, \ldots, n\}$. Wir zählen die Anzahl dieser Permutationen, die 1 auf ein festes Element x abbilden. Bei einer solchen Permutation werden die Zahlen $2, \ldots, n$ bijektiv auf die Zahlen $1, 2, \ldots, x-1, x+1, \ldots, n$ abgebildet. Nach Induktionsannahme gibt es $(n-1)!$ solche Bijektionen. Da es aber n Möglichkeiten gibt, 1 abzubilden, ist $n(n-1)! = n!$ die Gesamtzahl der Permutationen in S_n. □

Sei $X = \mathbb{Z}_2^n$ die Menge aller Bitstrings der Länge n. Eine Permutation von X, in der nur die Positionen der Bits vertauscht werden, heißt *Bitpermutation*. Um eine solche

Bitpermutation formal zu beschreiben, wählt man $\pi \in S_n$. Dann setzt man

$$f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n, b_1 \dots b_n \mapsto b_{\pi(1)} \dots b_{\pi(n)}.$$

Dies ist tatsächlich eine Bitpermutation und jede Bitpermutation lässt sich in eindeutiger Weise so schreiben. Es gibt also $n!$ Bitpermutationen von Bitstrings der Länge n.

Spezielle Bitpermutationen sind z. B. *zirkuläre Links- oder Rechtsshifts*. Ein zirkulärer Linksshift um i Stellen macht aus dem Bitstring $(b_0, b_1, \ldots, b_{n-1})$ den String $(b_{i \bmod n}, b_{(i+1) \bmod n}, \ldots, b_{(i+n-1) \bmod n})$. Zirkuläre Rechtsshifts sind entsprechend definiert.

3.7 Blockchiffren

Blockchiffren sind deterministische Verschlüsselungsverfahren, die Blöcke fester Länge auf Blöcke derselben Länge abbilden. Wie wir in Abschn. 3.10 sehen werden, kann man sie in unterschiedlicher Weise benutzen, um beliebig lange Texte zu verschlüsseln.

Definition 3.4 Unter einer *Blockchiffre* versteht man ein deterministisches Verschlüsselungsverfahren, dessen Klartext- und Schlüsseltextraum die Menge Σ^n aller Wörter der Länge n über einem Alphabet Σ sind. Die *Blocklänge* n ist eine natürliche Zahl. Der Schlüsselerzeugungsalgorithmus einer Blockchiffre wählt einen Schlüssel gleichverteilt zufällig aus dem Schlüsselraum. Blockchiffren der Länge 1 heißen *Substitutionschiffren*.

Wir beweisen folgende Eigenschaft von Blockchiffren:

Theorem 3.2 *Die Verschlüsselungsfunktionen von Blockchiffren sind Permutationen.*

Beweis Weil die Blockchiffre korrekt entschlüsselt, ist die Funktion injektiv. Eine injektive Abbildung $\Sigma^n \to \Sigma^n$ ist bijektiv. □

3.8 Permutationschiffren

Nach Theorem 3.2 können wir die allgemeinste Blockchiffre folgendermaßen beschreiben. Wir fixieren die Blocklänge n und ein Alphabet Σ. Als Klartext- und Schlüsseltextraum verwenden wir $\mathbf{P} = \mathbf{C} = \Sigma^n$. Der Schlüsselraum ist die Menge $S(\Sigma^n)$ aller Permutationen von Σ^n. Die Verschlüsselungsfunktion ist

$$\mathbf{Enc} : S(\Sigma^n) \times \Sigma^n \to \Sigma^n, (\pi, \vec{v}) \mapsto \pi(\vec{v}).$$

Die entsprechende Entschlüsselungsfunktion ist

$$\mathbf{Dec} : S(\Sigma^n) \times \Sigma^n \to \Sigma^n, (\pi, \vec{v}) \mapsto \pi^{-1}(\vec{v}).$$

Der Schlüsselraum dieses Verfahrens ist sehr groß. Er enthält $(|\Sigma|^n)!$ viele Elemente. Das Verfahren ist aber nicht besonders praktikabel, weil es eine explizite Darstellung des Schlüssels, also der Permutation π, benötigt. Wir können die Permutation darstellen, indem wir zu jedem Klartext $\vec{w} \in \Sigma^n$ den Wert $\pi(\vec{w})$ notieren, also eine Tabelle mit $|\Sigma|^n$ Werten verwenden. Die ist aber zu groß. Es ist daher vernünftig, als Ver- und Entschlüsselungsfunktionen nur eine Teilmenge aller möglichen Permutationen von Σ^n zu verwenden. Diese Permutationen sollen durch kurze Schlüssel leicht erzeugbar sein.

Ein Beispiel für eine solche Vereinfachung ist die *Permutationschiffre*. Sie verwendet nur solche Permutationen, die durch Vertauschen der Positionen der Zeichen entstehen. Der Schlüsselraum der Permutationschiffre ist die Permutationsgruppe S_n. Jedes Element $\pi \in S_n$ kann durch die Folge $(\pi(1), \ldots, \pi(n))$ dargestellt werden. Dies ist eine viel kürzere Darstellung als die Wertetabelle der entsprechenden Permutation in $S(\Sigma^n)$. Die Verschlüsselungsfunktion der Permutationschiffre ist

$$\mathbf{Enc} : S_n \times \Sigma^n \to \Sigma^n, \quad (\pi, (v_1, \ldots, v_n)) \mapsto (v_{\pi(1)}, \ldots v_{\pi(n)}).$$

Die zugehörige Entschlüsselungsfunktion ist

$$\mathbf{Dec} : S_n \times \Sigma^n \to \Sigma^n, \quad (\pi, (v_1, \ldots, v_n)) \mapsto (v_{\pi^{-1}(1)}, \ldots v_{\pi^{-1}(n)}).$$

Der Schlüsselraum der Permutationschiffre hat $n!$ viele Elemente. Jeder Schlüssel lässt sich als eine Folge von n Zahlen kodieren. Damit verhindert die Größe des Schlüsselraums (für genügend großes n) den Angriff durch vollständige Durchsuchung des Schlüsselraums. Gleichzeitig haben die Schlüssel eine kurze Darstellung als Permutation der Folge $(1, \ldots, n)$. Trotzdem ist die Permutationschiffre unsicher, wie wir in Abschn. 3.19 zeigen werden.

Beispiel 3.12 Wir wählen das Alphabet $\Sigma = \mathbb{Z}_2$ und die Blocklänge $n = 3$. Der Schlüsselraum der entsprechenden Permutationschiffre ist $S_3 = \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$. Die Verschlüsselungsfunktion ist also

$$\mathbf{Enc} : S_3 \times \mathbb{Z}_2^3 \to \mathbb{Z}_2^3, ((i,j,k), (v_1, v_2, v_3)) \mapsto (v_i, v_j, v_k).$$

Die entsprechende Entschlüsselungsfunktion ist

$$\mathbf{Dec} : S_3 \times \mathbb{Z}_2^3 \to \mathbb{Z}_2^3, ((i,j,k), (v_i, v_j, v_k)) \mapsto (v_1, v_2, v_3).$$

3.9 Mehrfachverschlüsselung

Will man die Sicherheit einer Blockchiffre steigern, so kann man sie mehrmals hintereinander mit verschiedenen Schlüsseln verwenden. Gebräuchlich ist die E-D-E-Dreifach-Verschlüsselung (Triple Encryption). Einen Klartext P verschlüsselt man zu

$$C \leftarrow \mathbf{Enc}(K_1, \mathbf{Dec}(K_2, \mathbf{Enc}(K_3, P))).$$

Dabei sind K_1, K_2, K_3 drei Schlüssel, **Enc** ist die Verschlüsselungsfunktion und **Dec** ist die Entschlüsselungsfunktion der Blockchiffre. Wir erreichen auf diese Weise eine erhebliche Vergrößerung des Schlüsselraums. Soll die Schlüssellänge nur verdoppelt werden, werden K_1 und K_3 identisch gewählt.

3.10 Verschlüsselungsmodi

Bevor wir weitere klassische Beispiele für Verschlüsselungsverfahren besprechen, zeigen wir erst, wie mit Hilfe von *Verschlüsselungsmodi* Blockchiffren zur Verschlüsselung von längeren Dokumenten verwendet werden können. Dabei verwenden wir eine Blockchiffre mit Alphabet $\mathbb{Z}_2$, Blocklänge $n \in \mathbb{N}$ und Schlüsselraum $\mathbf{K}$. Die Verschlüsselungsfunktion sei E und die Entschlüsselungsfunktion D. Die Bezeichnungen **Enc** und **Dec** werden für die Ver- und Entschlüsselungsverfahren reserviert, die aus den Verschlüsselungsmodi entstehen. Die in den folgenden Abschnitten behandelten Verschlüsselungsmodi erlauben die Konstruktion von Verschlüsselungsverfahren mit Klartext- und Schlüsseltextraum $\mathbb{Z}_2^*$. Der Schlüsselerzeugungsalgorithmus wählt jeweils zufällig und gleichverteilt einen Schlüssel aus dem Schlüsselraum der Blockchiffre. In der Beschreibung der Algorithmen werden Vektoren in $\mathbb{Z}_2^n$ mit den durch sie dargestellten ganzen Zahlen identifiziert. Umgekehrt werden ganze Zahlen zwischen 0 und $2^n - 1$ mit den Bitstrings der Länge n identifiziert, durch die sie dargestellt werden. Ist also die Binärdarstellung einer solchen Zahl zu kurz, werden führende Nullen vorangestellt. Dies wird im folgenden Beispiel illustriert.

Beispiel 3.13 Sei $n = 4$ die Blocklänge der gewählten Blockchiffre. Die Identifikation bezieht sich dann auf alle ganzen Zahlen zwischen 0 und 15. So entspricht zum Beispiel 0 dem String 0000, die Zahl 7 entspricht dem String 0111 und 14 entspricht 1110.

Die Verschlüsselungsmodi lassen sich auch für Blockchiffren über jedem Alphabeth Σ definieren. Allerdings müssen wir dann die Behandlung von Zählern modifizieren, die in einigen Varianten verwendet werden.

3.10.1 ECB-Mode

Eine naheliegende Art aus einer Blockchiffre ein Verschlüsselungsverfahren für beliebig lange Texte zu machen, ist ihre Verwendung im *Electronic-Codebook-Mode (ECB-Mode)*. Ein beliebig langer Klartext wird in Blöcke der Länge n aufgeteilt. Gegebenenfalls wird der Klartext so ergänzt, dass seine Länge durch n teilbar ist. Diese Ergänzung erfolgt zum Beispiel durch Anhängen von zufälligen Zeichen. Bei Verwendung des Schlüssels $K \in \mathbf{K}$ wird dann jeder Block der Länge n mit Hilfe der Funktion

$$E(K, \cdot) : \Sigma^n \to \Sigma^n, \quad P \mapsto E(K, P)$$

verschlüsselt. Der ECB-Mode-Schlüsseltext ist die Folge der entstehenden Schlüsseltextblöcke. Die Entschlüsselung erfolgt durch Anwendung der Entschlüsselungsfunktion D auf die verschlüsselten Blöcke. Algorithmus 3.4 ist der ECB-Verschlüsselungsalgorithmus. Algorithmus 3.5 ist der ECB-Entschlüsselungsalgorithmus. Die Algorithmen werden in Abb. 3.1 gezeigt.

Algorithmus 3.4 (Enc(K, P))

(ECB-Verschlüsselungsalgorithmus)

Die Verschlüsselungsfunktion E und die Blocklänge n sind bekannt.
Ergänze P so, dass die Länge von P durch n teilbar ist.
Zerlege P in Blöcke der Länge n: $P = P_1 \dots P_t$.
for $j = 1, \dots t$ **do**
  $C_j \leftarrow E(K, P_j)$
end for
$C \leftarrow C_1 \dots C_t$
return C

Algorithmus 3.5 (Dec(K, C))

(ECB-Entschlüsselungsalgorithmus)

Die Entschlüsselungsfunktion D und die Blocklänge n sind bekannt.
Die Länge von C ist durch n teilbar.
Zerlege C in Blöcke der Länge n: $C = C_1 \dots C_t$.
for $i = 1, \dots, t$ **do**
  $P_j \leftarrow D(K, C_j)$
end for
$P \leftarrow P_1 \dots P_t$
return P

Beispiel 3.14 Um Dokumente im ECB-Mode zu verschlüsseln, verwenden wir die Permutationschiffre mit Blocklänge 4. Es ist also $\mathbf{K} = S_4$ und

$$E : S_4 \times \mathbb{Z}_2^4 \to \mathbb{Z}_2^4, \quad (\pi, (b_1 b_2 b_3 b_4)) \mapsto b_{\pi(1)} b_{\pi(2)} b_{\pi(3)} b_{\pi(4)}.$$

Der Klartext P sei

$$P = 101100010100101.$$

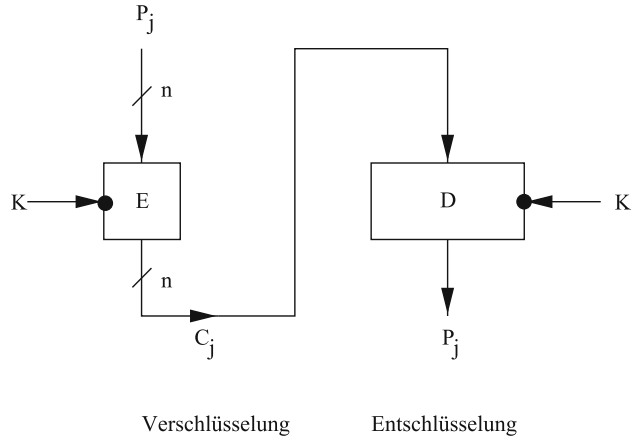

Abb. 3.1 ECB-Mode

Dieser Klartext wird in Blöcke der Länge vier aufgeteilt. Der letzte Block hat dann nur die Länge drei. Er wird auf die Länge vier ergänzt, indem eine Null angehängt wird. Man erhält

$$P = 1011\,0001\,0100\,1010,$$

also die Blöcke

$$P_1 = 1011, P_2 = 0001, P_3 = 0100, P_4 = 1010.$$

Wir verwenden den Schlüssel

$$K = (2, 3, 4, 1).$$

Die Blöcke werden nun einzeln verschlüsselt. Wir erhalten $C_1 = E(K, P_1) = 0111$, $C_2 = E(K, P_2) = 0010$, $C_3 = E(K, P_3) = 1000$, $C_4 = E(K, P_4) = 0101$. Der Chiffretext ist

$$C = 0111\,0010\,1000\,0101.$$

Der ECB-Mode kann auch bei Verschlüsselungsverfahren angewendet werden, die Blöcke der Länge n in Blöcke der größeren Länge m verschlüsseln. Ver- und Entschlüsselungsalgorithmen sind dann analog zu obigen Algorithmen definiert.

Bei der Verwendung des ECB-Mode werden gleiche Klartextblöcke in gleiche Chiffretextblöcke verschlüsselt. Diese Eigenschaft hat Nachteile für die Sicherheit, weil sie die Möglichkeit von Known-Plaintext-Angriffen eröffnet. Jedes Paar Klartext-Schlüsseltext, das der Angreifer kennt, ermöglicht es ihm in zukünftigen Chiffretexte alle Blöcke zu entschlüsseln, die in den Chiffretexten der bekannten Paare vorkommen.

Beispiel 3.15 Ein Klartext P wird für die ECB-Verschlüsselung in die Blöcke $P_1, \ldots, P_{10}$ aufgeteilt. Dann ist ist $C = C_1, \ldots, C_{10}$ der entsprechende Chiffretext. Angenommen, der Angreifer kennt P und C. Angenommen, der Angreifer sieht einen Schlüsseltext $C' = C'_1, \ldots, C'_{13}$ und es gilt $C'_5 = C_{10}$. Dann ist $P'_5 = P_{10}$.

3.10.2 CBC-Mode

Im *Cipherblock Chaining Mode* (CBC-Mode), der 1976 patentiert wurde, hängt die Verschlüsselung eines Klartextblocks nicht nur von diesem Block und dem Schlüssel, sondern auch von den vorhergehenden Blöcken ab. Die Verschlüsselung eines Blocks im CBC-Mode ist also im Gegensatz zur Verschlüsselung im ECB-Mode *kontextabhängig*. Das heißt, dass gleiche Blöcke in unterschiedlichem Kontext verschieden verschlüsselt werden. Außerdem verwendet der CBC-Mode einen Initialisierungsvektor. Er wird bei jeder Verwendung des CBC-Modes geändert. Wird dasselbe Dokument also mehrfach verschlüsselt, verändert der jeweils neue Initialisierungsvektor jedesmal den Chiffretext. Das erschwert die Anwendung eines Known-Plaintext-Angriffs. Der Initialisierungsvektor kann zufällig gewählt werden. Dann ist das Verschlüsselungsverfahren randomisiert. Es ist auch möglich, einen Zähler zu verwenden, der nach jeder Verschlüsselung hochgezählt wird. Dann ist das Verschlüsselungsverfahren zustandsbehaftet. Der Zustand ist der Zähler. Statt des Zählers kann man auch eine geeignete Kodierung der Zeit verwenden, zu der die Verschlüsselung stattfindet.

Der CBC-Mode wird nun im Detail beschrieben. Wir brauchen dazu noch eine Definition.

Definition 3.5 Die Verknüpfung

$$\oplus : \mathbb{Z}_2^2 \to \mathbb{Z}_2, (b, c) \mapsto b \oplus c$$

ist durch folgende Tabelle definiert:

b	c	$b \oplus c$
0	0	0
1	0	1
0	1	1
1	1	0

Diese Verknüpfung heißt *exklusives Oder* von zwei Bits.

Für $k \in \mathbb{N}$ und $b = (b_1, b_2, \ldots, b_k)$, $c = (c_1, c_2, \ldots, c_k) \in \mathbb{Z}_2^k$ setzt man $b \oplus c = (b_1 \oplus c_1, b_2 \oplus c_2, \ldots, b_k \oplus c_k)$.

Werden die Restklassen in $\mathbb{Z}/2\mathbb{Z}$ durch ihre kleinsten nicht negativen Vertreter 0 und 1 dargestellt, so entspricht das exklusive Oder zweier Elemente von $\mathbb{Z}/2\mathbb{Z}$ der Addition in $\mathbb{Z}/2\mathbb{Z}$.

Beispiel 3.16 Ist $b = 0100$ und $c = 1101$, dann ist $b \oplus c = 1001$.

Der im CBC-Mode verwendete *Initialisierungsvektor* ist ein Element von $\mathbb{Z}_2^n$. Beispiel 3.17 illustriert die Wahl dieses Initialisierungsvektors.

Beispiel 3.17 Wird eine Blockchiffre der Länge 6 verwendet, so ist ein zufälliger Initialisierungsvektor zum Beispiel 110001. Wird ein Zähler verwendet, der mit 0 initialisiert wird und dann jeweils um eins hochgezählt wird, so ist der erste Initialisierungsvektor 000000, der zweite 000001 und der zehnte 001001.

Wie im ECB-Mode wird auch im CBC-Mode der Klartext in Blöcke der Länge n aufgeteilt. Wollen wir eine Folge $P_1, \ldots, P_t$ von Klartextblöcken der Länge n mit dem Schlüssel K verschlüsseln, so setzen wir

$$C_0 = IV, \quad C_j = E(K, C_{j-1} \oplus P_j), \quad 1 \leq j \leq t.$$

Wir erhalten die Schlüsseltextblöcke

$$C_1, \ldots, C_t.$$

Um sie zu entschlüsseln, berechnen wir

$$C_0 = \text{IV}, \quad P_j = C_{j-1} \oplus D(K, C_j), \quad 1 \leq j \leq t. \tag{3.3}$$

Tatsächlich ist $C_0 \oplus D(K, C_1) = C_0 \oplus C_0 \oplus P_1 = P_1$. Entsprechend verifiziert man das ganze Verfahren. Algorithmus 3.6 ist die randomisierte Version der CBC-Verschlüsselung. Sie wird mit *CBC-$* bezeichnet und gibt als ersten Block des Chiffretextes den Initialisierungsvektor IV zurück, weil er für die Entschlüsselung benötigt wird, die in Algorithmus 3.7 gezeigt wird. Der CBC-Mode wird in Abb. 3.2 illustriert.

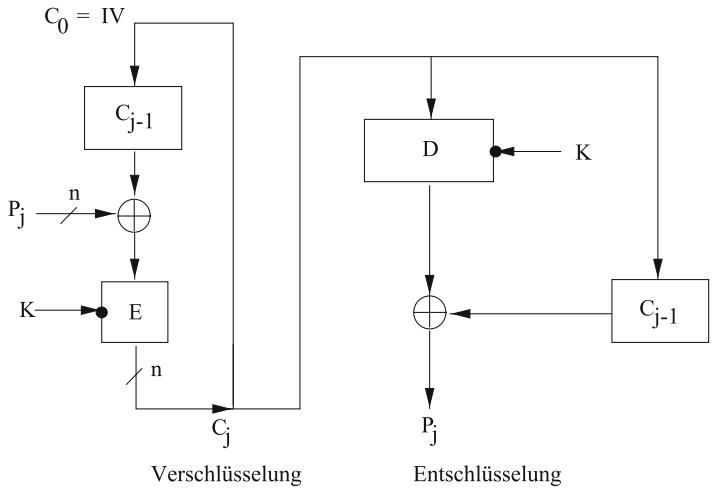

Abb. 3.2 CBC-Mode

Algorithmus 3.6 (Enc(K, P))

(CBC-$-Verschlüsselungsalgorithmus)

Die Verschlüsselungsfunktion E und die Blocklänge n sind bekannt.

IV $\xleftarrow{\$} \mathbb{Z}_2^n$

Ergänze P so, dass die Länge von P durch n teilbar ist.

Zerlege P in Blöcke der Länge n: $P = P_1 \dots P_t$.

$C_0 \leftarrow$ IV

for $j = 1, \dots, t$ **do**

  $C_j \leftarrow E(K, C_{j-1} \oplus P_j)$

end for

$C \leftarrow C_0 \dots C_t$

return C

Algorithmus 3.7 (Dec(K, C))

(CBC-Entschlüsselungsalgorithmus)

Die Entschlüsselungsfunktion D und die Blocklänge n sind bekannt.

Die Länge von C ist durch n teilbar.

Zerlege C in Blöcke der Länge n: $C = C_0 \dots C_t$.

for $j = 1, \dots, t$ **do**

  $P_j \leftarrow C_{j-1} \oplus D(K, C_j)$

end for

$P \leftarrow P_1 \dots P_t$

return P

Die Variante der CBC-Verschlüsselung, in der der Initialisierungsvektor hochgezählt wird, ist in Algorithmus 3.8 dargestellt. Sie wird mit CBC-CTR bezeichnet. Der Entschlüsselungsalgorithmus ist derselbe wie in der randomisierten Variante (siehe Algorithmus 3.7). Der Initialisierungsvektor IV ist die n-Bit-Darstellung eines Zählers. Er wird mit 0 initialisiert und ist der Zustand des Verschlüsselungsalgorithmus.

Algorithmus 3.8 (Enc(K, P))

(CBC-CTR-Verschlüsselungsalgorithmus)

Die Verschlüsselungsfunktion E und die Blocklänge n sind bekannt.

Der Initialisierungsvektor IV $\in \mathbb{Z}_2^n$ ist bekannt und kleiner als $2^n - 1$.

Ergänze P so, dass die Länge von P durch n teilbar ist.
Zerlege P in Blöcke der Länge n: $P = P_1 \dots P_t$.
$C_0 \leftarrow \text{IV}$
for $j = 1, \dots, t$ **do**
  $C_j \leftarrow E(K, C_{j-1} \oplus P_j)$
end for
$\text{IV} \leftarrow \text{IV} + 1$
$C \leftarrow C_0 \dots C_t$
return C

Beispiel 3.18 Wir verwenden dieselbe Blockchiffre, denselben Klartext und denselben Schlüssel wie in Beispiel 3.14. Die Klartextblöcke sind

$$P_1 = 1011, P_2 = 0001, P_3 = 0100, P_4 = 1010.$$

Der Schlüssel ist $K = (2, 3, 4, 1)$. Als Initialisierungsvektor verwenden wir

$$\text{IV} = 1010.$$

Damit ist also $C_0 = 1010, C_1 = E(K, C_0 \oplus P_1) = E(K, 0001) = 0010, C_2 = E(K, C_1 \oplus P_2) = E(K, 0011) = 0110, C_3 = E(K, C_2 \oplus P_3) = E(K, 0010) = 0100, C_4 = E(K, C_3 \oplus P_4) = E(K, 1110) = 1101$. Also ist der Schlüsseltext

$$C = 1010\,0010\,0110\,0100\,1101.$$

Wir entschlüsseln diesen Schlüsseltext wieder und erhalten $P_1 = C_0 \oplus D(K, C_1) = 1010 \oplus 0001 = 1011$, $P_2 = C_1 \oplus D(K, C_2) = 0010 \oplus 0011 = 0001$, $P_3 = C_2 \oplus D(K, C_3) = 0110 \oplus 0010 = 0100$, $P_4 = C_3 \oplus D(K, C_4) = 0100 \oplus 1110 = 1010$. Verwenden wir dagegen den Initialisierungsvektor

$$\text{IV} = 1110,$$

so ist

$$C = 1110\,1010\,0111\,0110\,1001.$$

Zuletzt ändern wir den vorletzten Klartextblock zu $P_3 = 0110$ und behalten den letzten Initialisierungsvektor bei. Dann ergibt sich der Chiffretext $C = 1110\,1010\,0111\,0010\,0100$. Die beiden letzten Blöcke haben sich verändert.

Man sieht an der Konstruktion und an Beispiel 3.18, dass gleiche Texte im CBC-Mode tatsächlich verschieden verschlüsselt werden, wenn man den Initialisierungsvektor ändert. Außerdem hängt die Verschlüsselung eines Blocks vom vorhergehenden Block ab.

Wir untersuchen, welche Auswirkungen Übertragungsfehler haben können. Bei solchen Übertragungsfehlern enthalten Blöcke des Schlüsseltextes, den der Adressat emp-

fängt, Fehler. Man sieht aber an (3.3), dass ein Übertragungsfehler im Schlüsseltextwort C_j nur bewirken kann, dass P_j und P_{j+1} falsch berechnet werden. Die Berechnung von $P_{j+2}, P_{j+3}, \ldots$ ist dann wieder korrekt, weil sie nicht von C_j abhängt. Das bedeutet auch, dass Sender und Empfänger nicht einmal denselben Initialisierungsvektor brauchen. Haben sie verschiedene Initialisierungsvektoren gewählt, so kann der Empfänger zwar vielleicht nicht den ersten Block, aber dann alle weiteren Blöcke korrekt entschlüsseln. Die Verschlüsselung ist aber nicht unabhängig vom Initialisierungsvektor. Die Information über den Initialisierungsvektor ist in allen Schlüsseltextblöcken enthalten.

3.10.3 CFB-Mode

Im *Cipher-Feedback-Mode (CFB-Mode)* werden Blöcke, die eine kürzere Länge als n haben können, nicht direkt durch die Blockverschlüsselungsfunktion E, sondern durch Addition mod 2 entsprechender Schlüsselblöcke verschlüsselt. Diese Schlüsselblöcke können mit Hilfe der Blockchiffre auf Sender- und Empfängerseite fast simultan berechnet werden. CFB-$-Mode-Verschlüsselung ist in Algorithmus 3.9 gezeigt. Der zustandsbehaftete CFB-CTR-Mode funktioniert analog. Wie bei der CBC-CTR-Verschlüsselung in Algorithmus 3.8 wird der Initialisierungsvektor als Zähler gewählt, der mit Null initialisiert und dann hochgezählt wird. Die entsprechende Illustration findet sich in Abb. 3.3

Algorithmus 3.9 (Enc(K, P))

(CFB-$-Verschlüsselungsalgorithmus)

Die Verschlüsselungsfunktion E, die Blocklänge n und r sind bekannt.

$\text{IV} \xleftarrow{\$} \mathbb{Z}_2^n$

Ergänze P so, dass die Länge von P durch r teilbar ist.

Zerlege P in Blöcke der Länge r: $P = P_1 \ldots P_t$.

$I_1 \leftarrow \text{IV}$

for $j = 1, \ldots, t$ **do**

$O_j \leftarrow E(K, I_j)$

$t_j \leftarrow$ die ersten r Bits von O_j

$C_j \leftarrow P_j \oplus t_j$

$I_{j+1} \leftarrow 2^r I_j + C_j \bmod 2^n$; I_{j+1} entsteht also

aus I_j durch Löschen der ersten r Bits und Anhängen von C_j

end for

$C \leftarrow C_1 \ldots C_t$

return (IV, C)

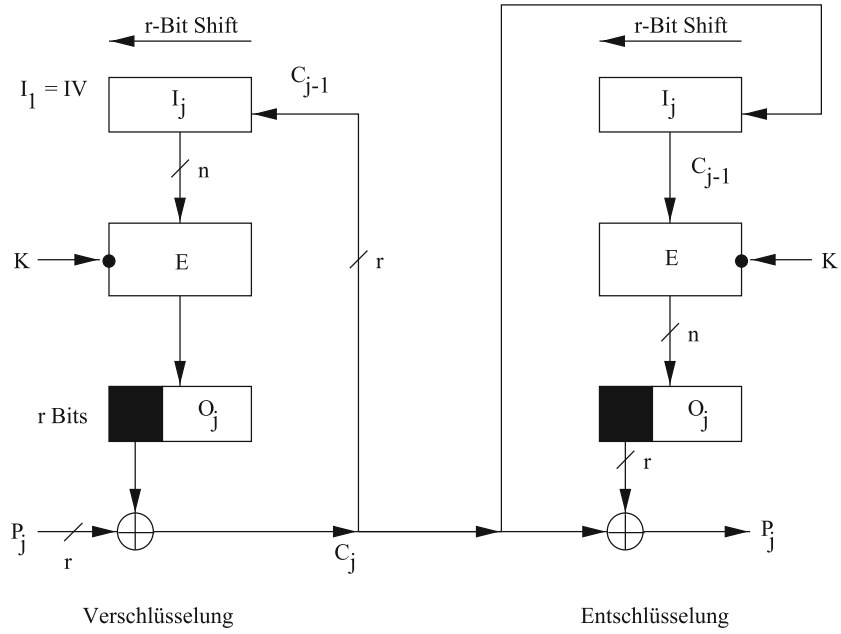

Abb. 3.3 CFB-Mode

Algorithmus 3.10 (Dec(K, C))

(CFB-Entschlüsselungsalgorithmus)

Die Verschlüsselungsfunktion E, die Blocklänge n und r sind bekannt.
Die Länge von C ist durch r teilbar.
Zerlege C in Blöcke der Länge r: $C = \text{IV}, C_1 \ldots C_t$.
$I_1 \leftarrow \text{IV}$
for $j = 1, \ldots, t$ **do**
  $O_j \leftarrow E(K, I_j)$
  $t_j \leftarrow$ die ersten r Bits von O_j
  $P_j \leftarrow C_j \oplus t_j$
  $I_{j+1} \leftarrow 2^r I_j + C_j \bmod 2^n$
end for
$P \leftarrow P_1 \ldots P_t$
return P

Wir erkennen, dass Sender und Empfänger den String t_1 simultan berechnen können. Außerdem kann der Empfänger t_{j+1} für $j \geq 1$ bestimmen, sobald er C_j kennt.

Beispiel 3.19 Wir verwenden Blockchiffre, Klartext und Schlüssel aus Beispiel 3.14. Außerdem verwenden wir als verkürzte Blocklänge $r = 3$. Die Klartextblöcke sind dann

$$P_1 = 101, P_2 = 100, P_3 = 010, P_4 = 100, P_5 = 101.$$

Der Schlüssel ist $K = (2, 3, 4, 1)$. Als Initialisierungsvektor verwenden wir

$$\text{IV} = 1010.$$

Die Verschlüsselung erfolgt dann gemäß folgender Tabelle.

j	I_j	O_j	t_j	P_j	C_j
1	1010	0101	010	101	111
2	0111	1110	111	100	011
3	1011	0111	011	010	001
4	1001	0011	001	100	101
5	1101	1011	101	101	000

Im CFB-Mode beeinflussen Übertragungsfehler das Ergebnis der Entschlüsselung solange, bis der fehlerhafte Ciphertextblock aus dem Vektor I_j herausgeschoben wurde. Wie lange das dauert, hängt von der Größe von r ab.

3.10.4 OFB-Mode

Der *Output-Feedback Mode* (OFB-Mode) ist dem CFB-Mode ähnlich. Der Unterschied besteht darin, dass

$$I_{j+1} \leftarrow O_j$$

gesetzt wird. Algorithmen 3.11 und 3.12 implementieren die OFB-$-Verschlüsselung und -Entschlüsselung, die einen Initialisierungsvektor zufällig wählt. Der zustandsbehaftete OFB-CTR-Mode funktioniert analog. Wie bei der CBC-CTR-Verschlüsselung in Algorithmus 3.8 wird der Initialisierungsvektor mit Null initialisiert und dann hochgezählt. Der OFB-Modus ist auch in Abb. 3.4 dargestellt.

Algorithmus 3.11 (Enc(K, P))

(OFB-$-Verschlüsselungsalgorithmus)

Die Verschlüsselungsfunktion E, die Blocklänge n und r sind bekannt.

$\text{IV} \xleftarrow{\$} \mathbb{Z}_2^n$

Ergänze P so, dass die Länge von P durch r teilbar ist.

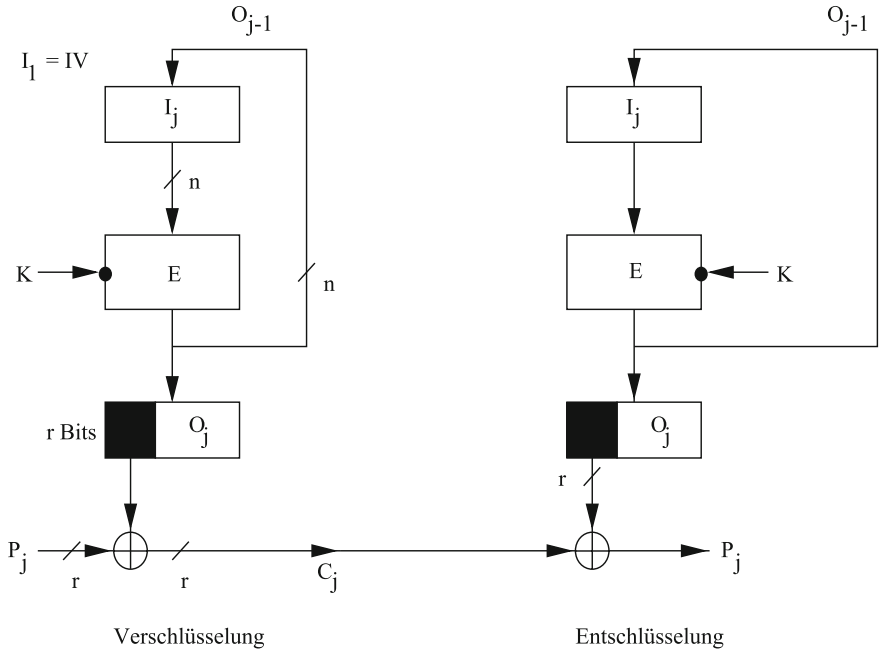

Abb. 3.4 OFB-Mode

Zerlege P in Blöcke der Länge r: $P = P_1 \dots P_t$.
$I_1 \leftarrow \text{IV}$
for $j = 1, \dots, t$ **do**
  $O_j \leftarrow E(K, I_j)$
  $t_j \leftarrow$ die ersten r Bits von O_j
  $C_j \leftarrow P_j \oplus t_j$
  $I_{j+1} \leftarrow O_j$
end for
$C \leftarrow C_1 \dots C_t$
return (IV, C)

Algorithmus 3.12 (Dec(K, C))

(OFB-Entschlüsselungsalgorithmus)

Die Verschlüsselungsfunktion E, die Blocklänge n und r sind bekannt.
Die Länge von C ist durch r teilbar.

Zerlege C in Blöcke der Länge r: $C = \mathrm{IV}, C_1 \ldots C_t$.
$I_1 \leftarrow \mathrm{IV}$
for $j = 1, \ldots, t$ **do**
  $O_j \leftarrow E(K, I_j)$
  $t_j \leftarrow$ die ersten r Bits von O_j
  $P_j \leftarrow C_j \oplus t_j$
  $I_{j+1} \leftarrow O_j$
end for
$P \leftarrow P_1 \ldots P_t$
return P

Werden im OFB-Mode bei der Übertragung eines Schlüsseltextwortes Bits verändert, so ensteht bei der Entschlüsselung nur ein Fehler an genau derselben Position, aber sonst nirgends.

Die Schlüsselstrings t_j hängen nur vom Initialisierungsvektor I_1 und vom Schlüssel K ab. Sie können also von Sender und Empfänger parallel berechnet werden. Die Verschlüsselung der Klartextblöcke hängt aber nicht von den vorherigen Klartextblöcken ab.

Beispiel 3.20 Wir verwenden Blockchiffre, Klartext und Schlüssel aus Beispiel 3.14. Außerdem verwenden wir als verkürzte Blocklänge $r = 3$. Die Klartextblöcke sind dann

$$P_1 = 101, P_2 = 100, P_3 = 010, P_4 = 100, P_5 = 101.$$

Der Schlüssel ist $K = (2, 3, 4, 1)$. Als Initialisierungsvektor verwenden wir

$$\mathrm{IV} = 1010.$$

Die Verschlüsselung erfolgt dann gemäß folgender Tabelle.

j	I_j	O_j	t_j	P_j	C_j
1	1010	0101	010	101	111
2	0101	1010	101	100	001
3	1010	0101	010	010	000
4	0101	1010	101	100	001
5	1010	0101	010	101	111

Wenn im OFB-Mode derselbe Schlüssel K mehrmals verwendet werden soll, muss der Initialisierungsvektor IV verändert werden. Sonst erhält man nämlich dieselbe Folge von Schlüsselstrings t_j. Hat man dann zwei Schlüsseltextblöcke $C_j = P_j \oplus t_j$ und $C'_j = P'_j \oplus t_j$, dann erhält man $C_j \oplus C'_j = P_j \oplus P'_j$. Hieraus kann man P'_j ermitteln, wenn P_j bekannt ist.

3.11 CTR-Mode

Der *Counter-Mode (CTR-Mode)* funktioniert folgendermaßen. Der Klartext P wird in Blöcke $P_1, \ldots, P_t$ der Länge n aufgeteilt. Der letzte Block darf kürzer sein. Ein Initialisierungsvektor IV wird entweder zufällig oder als Zähler gewählt. Entsprechend ist das entstehende Verschlüsselungsverfahren randomisiert oder zustandsbehaftet. Die Schlüsseltextblöcke sind

$$C_j = P_j \oplus E(K, \mathrm{IV} + j), \quad 1 \leq j \leq t. \tag{3.4}$$

Die Summe $\mathrm{IV} + j$ wird so berechnet: Zuerst wird die positive ganze Zahl I bestimmt, die durch den Bitstring IV dargestellt wird. Dann wird die Summe $I + j \bmod 2^n$ berechnet und als Bitstring der Länge n dargestellt. Die Entschlüsselung erfolgt so:

$$P_j = C_j \oplus E(K, \mathrm{IV} + j), \quad 1 \leq j \leq t. \tag{3.5}$$

Die Darstellung des Verfahrens in Form von Algorithmen überlassen wir dem Leser.

Beispiel 3.21 Wir verwenden Blockchiffre, Klartext und Schlüssel aus Beispiel 3.14. Die Klartextblöcke sind dann

$$P_1 = 1011, P_2 = 0001, P_3 = 0100, P_4 = 101.$$

Der Schlüssel ist $K = (2, 3, 4, 1)$ Als Initialisierungsvektor verwenden wir

$$\mathrm{IV} = 1010.$$

Dieser Initialisierungsvektor ist die Binärdarstellung der natürlichen Zahl 10. Um den Klartext zu verschlüsseln, benötigen wir die Binärdarstellungen der Zahlen 11, 12, 13 und 14. Sie sind 1011, 1100, 1101 und 1110. Damit erhalten wir die Schlüsseltextblöcke

$$\begin{aligned}
C_1 &= P_1 \oplus E(K, \mathrm{IV} + 1) = 1011 \oplus 0111 = 1100,\\
C_2 &= P_2 \oplus E(K, \mathrm{IV} + 2) = 0001 \oplus 1001 = 1000,\\
C_3 &= P_3 \oplus E(K, \mathrm{IV} + 3) = 0100 \oplus 1110 = 1010,\\
C_4 &= P_4 \oplus E(K, \mathrm{IV} + 4) = 101 \oplus 111 = 010.
\end{aligned}$$

Im letzen Schritt wurde $\mathrm{IV} + 4$ verkürzt anstatt P_4 um ein Bit zu ergänzen.

Im CTR-Mode wirken sich Übertragungsfehler in einem Chiffretext-Block nur auf den entsprechenden Klaretextblock aus. Übertragungsfehler beim Initialisierungsvektor wirken sich dagegen auf alle Klartextblöcke aus.

3.12 Stromchiffren

Im Gegensatz zu Blockchiffren verschlüsseln *Stromchiffren* Klartexte beliebiger Größe. Wir nehmen an, dass der Klartext ein Bitstring $P \in \mathbb{Z}_2^*$ ist. Andere Alphabete sind aber auch möglich. Die Länge von P sei $|P| = l$. Wir schreiben $P = P_1, \ldots, P_l$, wobei P_i die Bits in P sind für $1 \leq i \leq l$. Die Stromchiffre erzeugt aus einem Schlüssel und einem initialen Zustand der Stromchiffre, zum Beispiel einem Initialisierungsvektor, einen Schlüsselstrom $S = S_1, S_2, \ldots, S_l \in \mathbb{Z}_2^l$, der genauso lang ist wie P. Der Schlüsseltext $C = C_1, \ldots, C_l \in \mathbb{Z}_2^l$ wird so berechnet:

$$C_i = P_i \oplus S_i \quad 1 \leq i \leq l. \tag{3.6}$$

Entsprechend erfolgt die Entschlüsselung gemäß

$$P_i = C_i \oplus S_i, \quad 1 \leq i \leq l. \tag{3.7}$$

Es ist auch möglich, andere Berechnungsmethoden für Ver- und Entschlüsselung zu verwenden als in (3.6) dargestellt. Typischerweise wird aber diese Verschlüsselungsmethode verwendet. Bei Verwendung des Alphabets $\mathbb{Z}_2$ und der Verschlüsselung gemäß (3.6) spricht man von einer *binär additiven Stromchiffre*.

Beispiel 3.22 Wird eine Blockchiffre im CFB-Mode, im OFB-Mode oder im CTR-Mode verwendet, entsteht eine binär additive Stromchiffre. Der Startwert, aus dem der Schlüsselstrom erzeugt wird, ist der für die Blockchiffre verwendete Schlüssel K. Aus diesem Startwert wird der Schlüsselstrom $t_1, t_2, \ldots, t_u$ berechnet.

Für eine ausführlichere Behandlung von Stromchiffren verweisen wir auf [63].

3.13 Typen von Stromchiffren

In Abschn. 3.12 wurde bereits der Begriff der binär additiven Stromchiffre eingeführt. Hier werden weitere Typen vorgestellt.

In *synchronen Stromchiffren* wird der Schlüsselstrom unabhängig von Klartext und Chiffretext berechnet. Daher können sowohl der Verschlüsseler als auch der Entschlüsseler den Schlüsselstrom synchron berechnen, ohne dass der Entschlüsseler auf Teile des Chiffretextes warten muss. Der Schlüsseltext wird bitweise übertragen, damit Verschlüsseler und Entschlüsseler den Chiffretext bitweise synchron erzeugen bzw. entschlüsseln können. Dies ist besonders bei Realzeitanwendungen nützlich.

Beispiel 3.23 CFB-Mode und OFB-Mode sind synchrone Stromchiffren. In beiden Modi hängt der Schlüsselstrom nur vom Initialisierungsvektor, vom Verschlüsselungsverfahren und vom verwendeten Schlüssel ab aber nicht von Klar- der Chiffretexten.

Wenn in einer Stromchiffre bei der Übertragung des Chiffretextes ein Bit verloren geht, ohne dass der Empfänger dies bemerkt, kann das dazu führen, dass die gesamte folgende Entschlüsselung fehlschlägt. Dieses Problem beheben *selbstsynchronisierende Stromchiffren*. In solchen Stromchiffren werden die Schlüsselstrombits aus einer Anzahl vorhergehender Chiffretextbits berechnet. Das führt dazu, dass die Entschlüsselung auch bei Verlust einzelner Bits aus dem Schlüsseltext nach einer gewissen Zeit wieder funktioniert.

Beispiel 3.24 Die Verschlüsselungsverfahren, die bei Verwendung des CFB-Modes aus Abschn. 3.10.3 entstehen, sind selbstsynchronisierend. Mit den in Abschn. 3.10.3 verwendeten Bezeichnungen gilt nämlich folgendes: In der j-ten Iteration werden die nächsten r Bits des Schlüsselstroms als die ersten r Bits von $E(K, I_j)$ berechnet, wobei I_0 der Initialisierungsvektor und $I_{j+1} = 2^r I_j + C_j \bmod 2^n$ für $j \geq 0$ ist. Diese r Bits hängen also entweder vom Intialisierungsvektor oder vom vorhergehenden Schlüsseltextblock ab. Wenn bei der Übertragung des Initialisierungsvektors oder dieses Schlüsseltextblocks ein Fehler auftritt, hat das nur eine Folge für den gerade berechneten Block im Schlüsselstrom. Die folgenden Blöcke im Schlüsselstrom sind aber wieder korrekt.

Man erkennt am vorhergehenden Beispiel, dass bei selbstsynchronisierenden Stromchiffren der Schlüsseltext blockweise übertragen werden muss. Die Blöcke müssen so gewählt sein, dass daraus der Schlüsselstrom berechnet werden kann. Die blockweise Übertragung sorgt dafür, dass selbst bei fehlerhafter Übertragung eines Blocks der folgende Block wieder korrekt übertragen werden kann. Würde die Übertragung bitweise stattfinden, könnte der Entschlüsseler den Anfang der Blöcke möglicherweise nicht bestimmen. Dann würde sich ein Übertragungsfehler im Chiffretext auf alle folgenden Blöcke auswirken. Selbstsynchronisierende Stromchiffren sind deshalb *asynchron*. Das bedeutet, dass Verschlüsseler und Entschlüsseler die Bits nicht synchron ver- und entschlüsseln können.

3.14 Rückgekoppelte Schieberegister

Bekannte Stromchiffren sind *rückgekoppelte Schieberegister*. Der Klartext- und Schlüsseltextraum ist $\mathbb{Z}_2^*$. Die Schlüsselmenge ist $\mathbb{Z}_2^n$ für eine natürliche Zahl n. Wörter in $\mathbb{Z}_2^*$ werden Zeichen für Zeichen verschlüsselt. Das funktioniert so: Sei $K = (K_1, \ldots, K_n) \in \mathbb{Z}_2^n$ ein Schlüssel, $\mathrm{IV} = \mathrm{IV}_0, \ldots, \mathrm{IV}_{n-1}$ ein Initialisierungsvektor und $P = P_1 \ldots P_m$ ein Klartext der Länge m in $\mathbb{Z}_2^*$. Der Schlüsselstrom $S = S_1, S_2, \ldots$ wird gemäß

$$S_i = \mathrm{IV}_i, \quad 0 \leq i < n \tag{3.8}$$

berechnet und für $i \geq 0$ ist

$$S_{i+n} = f(S_i, S_{i+1}, \ldots, S_{i+n-1}). \tag{3.9}$$

Hierbei ist f die *Rückkopplungsfunktion*, die gleichzeitig den Schlüssel darstellt, also geheim gehalten werden muss. Die Verschlüsselungsfunktion E und die Entschlüsselungsfunktion D sind dann, wie bei Stromchiffren üblich, gegeben durch die Vorschriften

$$E(f, P) = P_1 \oplus S_1, \ldots, P_m \oplus S_m$$
$$D(f, C) = C_1 \oplus S_1, \ldots, C_m \oplus S_m.$$

Hierbei ist $P = P_1, \ldots, P_m \in \mathbb{Z}_2^*$ ein Klartext der Länge m und $C = C_1, \ldots, C_m \in \mathbb{Z}_2^*$ ein Schlüsseltext der Länge m. Der Initialsierungsvektor sorgt dafür, dass derselbe Klartext bei mehrfacher Verschlüsselung unterschiedlich verschlüsselt wird.

Man kann zum Beispiel *lineare Rückkopplungsfunktionen* wählen. Dann ist

$$f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} K_i x_i \bmod 2, \tag{3.10}$$

mit $K_0, \ldots K_{n-1} \in Z_2^n$. Das Schieberegister heißt dann *linear rückgekoppelt*. Die Gleichung (3.9) nennt man *lineare Rekursion* vom Grad n.

Beispiel 3.25 Sei $n = 4$ und sei $K = 0011$. Das entsprechende lineare Schieberegister berechnet den Schlüsselstrom gemäß der Formel

$$S_{i+4} = S_i + S_{i+1} \bmod 2, \quad i \geq 1.$$

Wir wählen den Initialisierungsvektor IV $= 1000$. Dann erhält man den Schlüsselstrom

$$100010011010111000\cdots$$

Der Schlüsselstrom ist periodisch und hat die Periodenlänge 15.

Lineare Rekursionen vom Grad n lassen sich durch ein sogenanntes lineares Schieberegister effizient als Hardwarebaustein realisieren. In Abb. 3.5 ist ein solches Schieberegister dargestellt. In den Registern befinden sich die letzten vier Werte des Schlüsselstroms. In jedem Schritt wird der Inhalt des ersten Registers zur Verschlüsselung verwendet. Dann wird der Inhalt des zweiten, dritten und vierten Registers um eins nach links geschoben und der Inhalt des vierten Registers entsteht durch Addition der Inhalte der Register mod 2, für die das entsprechende Bit K_i gleich 1 ist.

Wie wir im Abschn. 3.17 sehen werden, sind linear rückgekoppellte Stromchiffren leider unsicher, weil sie einen Known-Plaintext-Angriff zulassen.

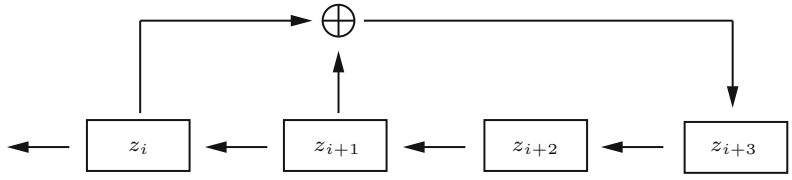

Abb. 3.5 Lineares Schieberegister

Rückgekoppelte Schieberegister sind periodisch. Bezeichne den aktuellen Zustand eines Schieberisters mit

$$Z_i = (S_i, S_{i+1}, \ldots, S_{i+n-1}). \tag{3.11}$$

Die Anzahl der verschiedenen Zustände ist höchstens 2^n. Nach spätestens 2^n Iterationen wiederholt sich also ein Zustand. Es gilt dann

$$Z_{i+k} = Z_i, \quad i, k \leq 2^n. \tag{3.12}$$

Wegen der rekursiven Konstruktion in (3.9) wird die Folge der Zustände dann periodisch und wir haben

$$S_{j+k} = S_j, \quad j \geq k. \tag{3.13}$$

Wählt man in (3.12) die Zahlen i und k minimal, so heißt i die *Vorperiodenlänge* und k die *Periodenlänge* der Chiffre. Wie schon bemerkt, ist die maximale Peridodenlänge einer Blockchiffre 2^n, wenn ein Schlüsselstrombit durch die n vorhergehenden Bits im Schlüsselstrom festgelegt ist.

3.15 Die affine Chiffre

Sei m eine natürliche Zahl. Die *affine Chiffre* mit Klartextalphabet $\mathbb{Z}_m$ ist eine Blockchiffre der Blocklänge $n = 1$. Der Schlüsselraum $\mathbf{K}$ besteht aus allen Paaren $(a, b) \in \mathbb{Z}_m^2$ mit zu m teilerfremdem a. Die Verschlüsselungsfunktion zum Schlüssel $K = (a, b) \in \mathbb{Z}_m^2$ ist

$$\mathbf{K} \times \Sigma \to \Sigma, \quad ((a, b), x) \mapsto ax + b \bmod m.$$

Die Entschlüsselungsfunktion zum Schlüssel K ist

$$\mathbf{K} \times \Sigma \to \Sigma, \quad ((a, b), x) \mapsto a'(x - b) \bmod m.$$

Hierbei ist a' so gewählt, dass $aa' \equiv 1 \bmod m$ ist. Die Zahl a' kann mit dem erweiterten euklidischen Algorithmus berechnet werden.

Beispiel 3.26 Wir verwenden das Alphabet $\{\mathrm{A}, \mathrm{B}, \ldots, \mathrm{Z}\}$ und identifizieren es mit $\mathbb{Z}_{26}$. Außerdem benutzen wir den Schlüssel $K = (a, b) = (7, 3)$ und verschlüsseln das Wort BALD mit der affinen Chiffre im ECB-Mode. Dann ergibt sich:

B	A	L	D
1	0	11	3
10	3	2	24
K	D	C	Y

Zur Berechnung der entsprechenden Entschlüsselungsfunktion bestimmen wir a' mit $7a' \equiv 1 \bmod 26$. Der erweiterte euklidische Algorithmus liefert $a' = 15$. Die Entschlüsselungsfunktion bildet also einen Buchstaben x auf $15(x - 3) \bmod 26$ ab. Tatsächlich

erhalten wir

K	D	C	Y
10	3	2	24
1	0	11	3
B	A	L	D

Der Schlüsselraum der affinen Chiffre mit $m = 26$ hat $\varphi(26) * 26 = 312$ Elemente. Also können Angreifer die affine Chiffre mit Hilfe vollständiger Durchsuchung des Schlüsselraums entschlüsseln. Ein Known-Plaintext-Angriff, bei dem zwei Zeichen und ihre Verschlüsselung bekannt sind, kann den Schlüssel mit Hilfe der linearen Algebra ermitteln. Dies wird im folgenden Beispiel vorgeführt.

Beispiel 3.27 Das Alphabet $\{A, B, \ldots, Z\}$ wird mit $\mathbb{Z}_{26}$ identifiziert. Wenn ein Angreifer weiß, dass bei Anwendung der affinen Chiffre mit Schlüssel (a, b) das Zeichen E zu R und das Zeichen S zu H verschlüsselt wird, dann gelten die Kongruenzen

$$4a + b \equiv 17 \bmod 26 \quad 18a + b \equiv 7 \bmod 26.$$

Aus der ersten Kongruenz ergibt sich $b \equiv 17 - 4a \bmod 26$. Setzt der Angreifer dies in die zweite Kongruenz ein, so erhält er $18a + 17 - 4a \equiv 7 \bmod 26$ und damit $14a \equiv 16 \bmod 26$. Daraus folgt $7a \equiv 8 \bmod 13$. Multipliziert der Angreifer diese Kongruenz mit dem Inversen 2 von 7 modulo 13, so erhält er $a \equiv 3 \bmod 13$ und schließt daraus $a = 3$ und $b = 5$.

3.16 Matrizen und lineare Abbildungen

Wir wollen affine Chiffren verallgemeinern. Dazu führen wir einige grundlegende Ergebnisse der linearen Algebra über Ringen auf, ohne sie zu beweisen. Details findet man in jedem Buch über lineare Algebra, zum Beispiel in [54]. Es sei R ein kommutativer Ring mit Einselement 1. Zum Beispiel kann $R = \mathbb{Z}/m\mathbb{Z}$ sein mit einer natürlichen Zahl m.

3.16.1 Matrizen über Ringen

Eine $k \times n$-*Matrix* über R ist ein rechteckiges Schema

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,n} \end{pmatrix}$$

mit k Zeilen und n Spalten. Wir schreiben auch

$$A = (a_{i,j}).$$

Ist $n = k$, so heißt die Matrix *quadratisch*. Die *i-te Zeile* von A ist der Vektor $(a_{i,1}, \ldots, a_{i,n})$, $1 \leq i \leq k$. Die *j-te Spalte* von A ist der Vektor $(a_{1,j}, \ldots, a_{k,j})$, $1 \leq j \leq n$. Der *Eintrag* in der i-ten Zeile und j-ten Spalte von A ist $a_{i,j}$. Die Menge aller $k \times n$-Matrizen über R wird mit $R^{(k,n)}$ bezeichnet.

Beispiel 3.28 Sei $R = \mathbb{Z}$. Eine Matrix aus $\mathbb{Z}^{(2,3)}$ ist z. B.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

Sie hat zwei Zeilen, nämlich $(1, 2, 3)$ und $(4, 5, 6)$ und drei Spalten, nämlich $(1, 4)$, $(2, 5)$ und $(3, 6)$.

3.16.2 Produkt von Matrizen mit Vektoren

Ist $A = (a_{i,j}) \in R^{(k,n)}$ und $\vec{v} = (v_1, \ldots, v_n) \in R^n$, dann ist das Produkt $A\vec{v}$ definiert als der Vektor $\vec{w} = (w_1, w_2, \ldots, w_k)$ mit

$$w_i = \sum_{j=1}^{n} a_{i,j} v_j, \quad 1 \leq i \leq k.$$

Beispiel 3.29 Sei $A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$, $\vec{v} = (1, 2)$. Dann ist $A\vec{v} = (5, 8)$.

3.16.3 Summe und Produkt von Matrizen

Sei $n \in \mathbb{N}$ und seien $A, B \in R^{(n,n)}$, $A = (a_{i,j})$, $B = (b_{i,j})$. Die *Summe* von A und B ist

$$A + B = (a_{i,j} + b_{i,j}).$$

Das *Produkt* von A und B ist $A \cdot B = AB = (C_{i,j})$ mit

$$C_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Beispiel 3.30 Sei $A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$, $B = \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix}$. Dann ist $A + B = \begin{pmatrix} 5 & 7 \\ 8 & 10 \end{pmatrix}$, $AB = \begin{pmatrix} 16 & 19 \\ 26 & 31 \end{pmatrix}$, $BA = \begin{pmatrix} 14 & 23 \\ 20 & 33 \end{pmatrix}$. Man sieht daran, dass die Multiplikation von Matrizen im allgemeinen nicht kommutativ ist.

3.16.4 Der Matrizenring

Die $n \times n$-Einheitsmatrix (über R) ist $E_n = (e_{i,j})$ mit

$$e_{i,j} = \begin{cases} 1 & \text{für } i = j, \\ 0 & \text{für } i \neq j. \end{cases}$$

Die $n \times n$-Nullmatrix (über R) ist die $n \times n$-Matrix, deren sämtliche Einträge Null sind. Wir schreiben dafür (0).

Beispiel 3.31 Die 2×2-Einheitsmatrix über $\mathbb{Z}$ ist $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Die 2×2-Nullmatrix über $\mathbb{Z}$ ist $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$.

Zusammen mit Addition und Multiplikation ist $R^{(n,n)}$ ein Ring mit Einselement E_n, der aber im allgemeinen nicht kommutativ ist. Das neutrale Element bezüglich der Addition ist die Nullmatrix.

3.16.5 Determinante

Die *Determinante* $\det A$ einer Matrix $A \in R^{(n,n)}$ kann rekursiv definiert werden. Ist $n = 1$, $A = (a)$, dann ist $\det A = a$. Sei $n > 1$. Für $i, j \in \{1, 2, \ldots n\}$ bezeichne mit $A_{i,j}$ die Matrix, die man aus A erhält, wenn man in A die i-te Zeile und die j-te Spalte streicht. Fixiere $i \in \{1, 2, \ldots, n\}$. Dann ist die Determinante von A

$$\det A = \sum_{j=1}^{n} (-1)^{i+j} a_{i,j} \det A_{i,j}$$

Dieser Wert ist unabhängig von der Auswahl von i und es gilt für alle $j \in \{1, 2, \ldots, n\}$

$$\det A = \sum_{i=1}^{n} (-1)^{i+j} a_{i,j} \det A_{i,j}.$$

Beispiel 3.32 Sei $A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$. Dann ist $A_{1,1} = (a_{2,2})$, $A_{1,2} = (a_{2,1})$, $A_{2,1} = (a_{1,2})$, $A_{2,2} = (a_{1,1})$. Daher ist $\det A = a_{1,1}a_{2,2} - a_{1,2}a_{2,1}$.

3.16.6 Inverse von Matrizen

Eine Matrix A aus $R^{(n,n)}$ besitzt genau dann ein multiplikatives Inverses, wenn $\det A$ eine Einheit in R ist. Wir geben eine Formel für das Inverse an. Falls $n = 1$ ist, so ist $(a_{1,1}^{-1})$ das Inverse von A. Sei $n > 1$ und $A_{i,j}$ wie oben definiert. Die *Adjunkte* von A ist eine $n \times n$-Matrix. Sie ist definiert als

$$\operatorname{adj} A = ((-1)^{i+j} \det A_{j,i}).$$

Die Inverse von A ist

$$A^{-1} = (\det A)^{-1} \operatorname{adj} A.$$

Beispiel 3.33 Sei $A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$. Dann ist $\operatorname{adj} A = \begin{pmatrix} a_{2,2} & -a_{1,2} \\ -a_{2,1} & a_{1,1} \end{pmatrix}$.

Sind $A = (a_{i,j})$, $B = (b_{i,j}) \in \mathbb{Z}^{(n,n)}$ und ist $m \in \mathbb{N}$, so schreiben wir

$$A \equiv B \bmod m$$

wenn $a_{i,j} \equiv b_{i,j} \bmod m$ ist für $1 \leq i, j \leq n$.

Als Anwendung der Ergebnisse dieses Abschnitts beschreiben wir, wann die Kongruenz

$$AA' \equiv E_n \bmod m \tag{3.14}$$

eine Lösung $A' \in \mathbb{Z}^{(n,n)}$ hat und wie man sie findet. Wir geben zuerst ein Beispiel.

Beispiel 3.34 Sei $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Wir lösen die Kongruenz

$$AA' \equiv E_2 \bmod 11 \tag{3.15}$$

mit $A' \in \mathbb{Z}^{(2,2)}$. Bezeichnet man mit $\bar{A}$ die Matrix, die man erhält, indem man die Einträge von A durch ihre Restklassen mod m ersetzt, dann ist also das Inverse dieser Matrix gesucht. Es existiert, wenn die Determinante von $\bar{A}$ eine Einheit in $\mathbb{Z}/11\mathbb{Z}$ ist. Das ist genau dann der Fall, wenn $\det A$ zu 11 teilerfremd ist. Nun ist $\det A = -2$, also teilerfremd zu 11. Außerdem ist $(-2)(-6) \equiv 1 \bmod 11$. Setzt man also

$$A' = (-6) * \operatorname{adj} A \bmod 11 = 5 * \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix} \bmod 11 = \begin{pmatrix} 9 & 1 \\ 7 & 5 \end{pmatrix},$$

so hat man eine Lösung der Kongruenz (3.15) gefunden.

Wir verallgemeinern das Ergebnis des vorigen Beispiels. Sei $A \in \mathbb{Z}^{n,n}$ und $m > 1$. Dann hat die Kongruenz (3.14) genau dann eine Lösung, wenn $\det A$ teilerfremd zu m ist. Ist dies der Fall und ist a eine ganze Zahl mit $a \det A \equiv 1 \bmod m$, dann ist

$$A' = a \operatorname{adj} A \bmod m$$

eine Lösung der Kongruenz (3.14). Diese Lösung ist eindeutig mod m. Man erkennt, dass die Matrix A' in Polynomzeit berechnet werden kann.

3.16.7 Affin lineare Funktionen

Wir definieren *affin lineare Funktionen*. Man kann sie verwenden, um einfache Blockchiffren zu konstruieren.

Definition 3.6 Eine Funktion $f : R^n \to R^l$ heißt *affin linear*, wenn es eine Matrix $A \in R^{(l,n)}$ und einen Vektor $\vec{b} \in R^l$ gibt, so dass für alle $\vec{v} \in R^n$

$$f(\vec{v}) = A\vec{v} + \vec{b}$$

gilt. Ist $\vec{b} = 0$, so heißt die Abbildung *linear*.

Affin lineare Abbildungen $\mathbb{Z}_m^n \to \mathbb{Z}_m^l$ sind analog definiert.

Definition 3.7 Eine Funktion $f : \mathbb{Z}_m^n \to \mathbb{Z}_m^l$ heißt *affin linear*, wenn es eine Matrix $A \in \mathbb{Z}_m^{(l,n)}$ und einen Vektor $\vec{b} \in \mathbb{Z}_m^l$ gibt, so dass für alle $\vec{v} \in \mathbb{Z}_m^n$

$$f(\vec{v}) = (A\vec{v} + \vec{b}) \bmod m$$

gilt. Ist $\vec{b} \equiv 0 \bmod m$, so heißt die Abbildung *linear*.

Theorem 3.3 *Die affin lineare Abbildung aus Definition 3.6 ist genau dann bijektiv, wenn $l = n$ und $\det A$ eine Einheit aus R ist.*

Aus Theorem 3.3 folgt, dass die Abbildung aus Definition 3.6 genau dann bijektiv ist, wenn $l = n$ und $\det A$ teilerfremd zu m ist.

Beispiel 3.35 Betrachte die Abbildung $f : \mathbb{Z}_2^2 \to \mathbb{Z}_2^2$, die definiert ist durch

$$f(0,0) = (0,0), f(1,0) = (1,1), f(0,1) = (1,0), f(1,1) = (0,1).$$

Diese Abbildung ist linear, weil $f(\vec{v}) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \vec{v}$ für alle $\vec{v} \in \mathbb{Z}_2^2$ gilt.

Wir charakterisieren lineare und affin lineare Abbildungen.

Theorem 3.4 *Eine Abbildung $f : R^n \to R^l$ ist genau dann linear, wenn für alle $\vec{v}, \vec{w} \in R^n$ und alle $a, b \in R$*

$$f(a\vec{v} + b\vec{w}) = af(\vec{v}) + bf(\vec{w})$$

gilt. Sie ist genau dann affin linear, wenn die Abbildung $R^n \to R^l, \vec{v} \mapsto f(\vec{v}) - f(\vec{0})$ linear ist.

3.17 Affin lineare Blockchiffren

Wir definieren *affin lineare Blockchiffren*. Sie sind Verallgemeinerungen der affinen Chiffren. Wir beschreiben diese Chiffren hier einerseits aus historischen Gründen. Andererseits zeigen wir, wie leicht Known-Plaintext-Angriffe auf diese Chiffren sind. Das zeigt, dass man beim Design von Blockchiffren vermeiden muss, dass sie affin linear sind.

Um affin lineare Blockchiffren zu definieren, brauchen wir eine natürliche Zahl n, die Blocklänge, und eine natürliche Zahl m, $m > 1$.

Wir beschreiben die allgemeinste affin lineare Blockchiffre mit Blocklänge n über dem Alphabet $\mathbb{Z}_m$ explizit. Klar- und Schlüsseltextraum sind $\mathbb{Z}_m^n$. Der Schlüsselraum $\mathbf{K}$ besteht aus allen Paaren $(A, \vec{b}) \in \mathbb{Z}_m^{(n,n)} \times \mathbb{Z}_m^n$ mit der Eigenschaft, dass $\det A$ teilerfremd zu m ist. Für einen Schlüssel $K = (A, \vec{b})$ ist die entsprechende Verschlüsselungsfunktion

$$\mathbf{Enc} : \mathbf{K} \times \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad ((A, \vec{b}), \vec{v}) \mapsto A\vec{v} + \vec{b} \bmod m$$

Die Bedingung, dass $\gcd(\det A, m) = 1$ ist zwingend, weil nach Theorem 3.2 Verschlüsselungsfunktionen von Blockchiffren bijektiv sind, woraus mit 3.3 folgt, dass $\det A$ teilerfremd zu m ist. Die entsprechende Entschlüsselungsfunktion ist nach den Ergebnissen von Abschn. 3.16.6

$$\mathbf{Dec} : \mathbf{K} \times \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad ((A, \vec{b}, \vec{v})) \mapsto A'(\vec{v} - \vec{b}) \bmod m,$$

wobei $A' = (a' \operatorname{adj} A) \bmod m$ und a' das Inverse von $\det A \bmod m$ ist.

Der Name „affin lineare Chiffre“ kommt daher, dass Ver- und Entschlüsselungsfunktion affin linear sind, wenn der Schlüssel fixiert wird. Bei *linearen Chiffren* sind die Schlüssel von der Form $(A, \vec{0})$. Also sind Ver- und Entschlüsslungsfunktion für jeden festen Schlüssel linear.

3.18 Vigenère, Hill- und Permutationschiffre

Wir geben zwei Beispiele für affin lineare Chiffren.

Die Vigenère Chiffre ist nach Blaise de Vigenère benannt, der im 16. Jahrhundert lebte. Der Schlüsselraum ist $\mathbb{Z}_m^n$. Ist $\vec{k} \in \mathbb{Z}_m^n$, dann ist

$$\mathbf{Enc} : (\mathbb{Z}_m^n)^2 \to \mathbb{Z}_m^n, \quad (\vec{k}, \vec{v}) \mapsto \vec{v} + \vec{k} \bmod m$$

und

$$\mathbf{Dec} : (\mathbb{Z}_m^n)^2 \to \mathbb{Z}_m^n, \quad (\vec{k}, \vec{v}) \mapsto \vec{v} - \vec{k} \bmod m.$$

Die Abbildungen sind offensichtlich affin linear. Der Schlüsselraum hat m^n Elemente.

Eine anderes klassisches Verschlüsselungsverfahren ist die *Hill-Chiffre*, die 1929 von Lester S. Hill erfunden wurde. Der Schlüsselraum $\mathbf{K}$ ist also die Menge aller Matrizen $A \in \mathbb{Z}_m^{(n,n)}$ mit $\gcd(\det A, m) = 1$. Für $A \in \mathcal{K}$ ist

$$\mathbf{Enc} : \mathbf{K} \times \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad (A, \vec{v}) \mapsto A\vec{v} \bmod m. \tag{3.16}$$

Die Hill-Chiffre ist also die allgemeinste lineare Blockchiffre. Die Anzahl der Schlüssel der Hill-Chiffre ist ungefähr gleich der Anzahl der $n \times n$ Matrizen mit Einträgen aus $\mathbb{Z}_m$, also ungefähr gleich m^{n^2}. Eine genauere Berechnung der Schlüsselanzahl findet sich in [55].

Zuletzt zeigen wir noch, dass die Permutationschiffre linear ist. Sei $\pi \in S_n$ und seien $\vec{e}_i$, $1 \le i \le n$, die Einheitsvektoren der Länge n, also die Zeilenvektoren der Einheitsmatrix. Sei ferner A_π die $n \times n$-Matrix, deren j-te Spalte $\vec{e}_{\pi(i)}$ ist, $1 \le j \le n$. Diese Matrix erhält man aus der Einheitsmatrix, indem man ihre Spalten gemäß der Permutation π vertauscht. Dann gilt für jeden Vektor $\vec{v} = (v_1, \ldots, v_n) \in \Sigma^n$

$$(v_{\pi(1)}, \ldots, v_{\pi(n)}) = E_\pi \vec{v}.$$

Die Permutationschiffre ist damit eine lineare Chiffre, also ein Spezialfall der Hill-Chiffre.

3.19 Kryptoanalyse affin linearer Blockchiffren

Wir zeigen, wie eine affin lineare Blockchiffre mit Alphabet $\mathbb{Z}_m$ und Blocklänge n mittels einer Known-Plaintext-Attacke gebrochen werden kann.

Die Ausgangssituation: Ein Modul $m \in \mathbb{N}_{\ge 2}$ und ein Schlüssel $(A, \vec{b}) \in \mathbb{Z}_m^{(m,n)} \times \mathbb{Z}_m^n$ für die affin lineare Chiffre wurden gewählt. Die zugehörige Verschlüsselungsfunktion ist von der Form

$$\mathbf{Enc} : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad \vec{v} \mapsto A\vec{v} + \vec{b} \bmod m$$

Wir zeigen wie ein Known-Plaintext-Angriff den Schlüssel $(A, \vec{b})$ findet. Dazu verwendet der Angreifer $n + 1$ Klartexte $P_i \in \mathbb{Z}_m^n$, $0 \le i \le n$, und die zugehörigen Schlüsseltexte $C_i = AP_i + \vec{b} \bmod m$, $0 \le i \le n$. Dann ist

$$C_i - C_0 \equiv A(P_i - P_0) \bmod m$$

Ist P die Matrix

$$P = (P_1 - P_0, \ldots, P_n - P_0) \bmod m$$

deren Spalten die Differenzen $(P_i - P_0) \bmod m$, $1 \leq i \leq n$, sind, und ist C die Matrix

$$C = (C_1 - C_0, \ldots, C_n - C_0) \bmod m,$$

deren Spalten die Differenzen $(C_i - C_0) \bmod m$, $1 \leq i \leq n$, sind, dann gilt

$$AP \equiv C \bmod m.$$

Ist $\det P$ teilerfremd zu m, so ist

$$A \equiv C(p' \operatorname{adj} P) \bmod m,$$

wobei p' das Inverse von $\det(P) \bmod m$ ist. Weiter ist

$$\vec{b} = C_0 - AP_0.$$

Damit ist der Schlüssel aus $n + 1$ Paaren von Klar- und Schlüsseltexten bestimmt worden. Ist die Chiffre sogar linear, so kann man $P_0 = C_0 = \vec{0}$ setzen, und es ist $\vec{b} = 0$. Dann benötigt man nur n Klartext-Schlüsseltext-Paare.

Beispiel 3.36 Wir zeigen, wie eine Hill-Chiffre mit Blocklänge 2 gebrochen werden kann. Angenommen, man weiß, dass HAND in FOOT verschlüsselt wird. Damit wird $P_1 = (7, 0)$ zu $C_1 = (5, 14)$ und $P_2 = (13, 3)$ zu $C_2 = (14, 19)$ verschlüsselt. Wir erhalten also $P = \begin{pmatrix} 7 & 13 \\ 0 & 3 \end{pmatrix}$ und $C = \begin{pmatrix} 5 & 14 \\ 14 & 19 \end{pmatrix}$. Es ist $\det P = 21$ teilerfremd zu 26. Das Inverse von 21 mod 26 ist 5. Also ist

$$A = 5C(\operatorname{adj} P) \bmod 26 = 5 * \begin{pmatrix} 5 & 14 \\ 14 & 19 \end{pmatrix} \begin{pmatrix} 3 & 13 \\ 0 & 7 \end{pmatrix} \bmod 26 = \begin{pmatrix} 23 & 9 \\ 2 & 15 \end{pmatrix}.$$

Tatsächlich ist $AP = C$.

3.20 Sichere Blockchiffren

Da sichere Blockchiffren sehr wichtige Bausteine für sichere Verschlüsselungsverfahren sind, behandeln wir zuletzt in diesem Kapitel Konstruktionsprinzipien für sichere Blockchiffren.

3.20.1 Konfusion und Diffusion

Wichtige Konstruktionsprinzipien sind große *Konfusion* und *Diffusion*. Sie wurden von Shannon vorgeschlagen, der die kryptographische Sicherheit im Rahmen seiner *Informationstheorie* untersuchte (siehe [68]). In Kap. 4 geben wir eine Einführung in die Shannonsche Theorie.

Die Konfusion einer Blockchiffre ist groß, wenn die statistische Verteilung der Chiffretexte in einer so komplizierten Weise von der Verteilung der Klartexte abhängt, dass ein Angreifer diese Abhängigkeit nicht ausnutzen kann. Die Verschiebungschiffre hat zum Beispiel viel zu geringe Konfusion. Die Wahrscheinlichkeitsverteilung der Klartextzeichen überträgt sich unmittelbar auf die Chiffretextzeichen.

Die Diffusion einer Blockchiffre ist groß, wenn jedes einzelne Bit des Klartextes und jedes einzelne Bit des Schlüssel möglichst viele Bits des Chiffretextes beeinflußt.

Man sieht, dass Konfusion und Diffusion intuitive aber nicht mathematisch formalisierte Begriffe sind. Sie zeigen aber, wie sichere Blockchiffren konstruiert werden sollen.

Neben den Prinzipien Konfusion und Diffusion postuliert Shannon, dass eine sichere Blockchiffre gegen alle bekannten Angriffe resistent sein muss. Wie wir im Abschn. 3.19 gesehen haben, dürfen sichere Blockchiffren zum Beispiel nicht affin linear sein. Aber das genügt nicht. Angriff durch vollständige Suche im Schlüsselraum wurde schon in Abschn. 3.4.2 behandelt. Wir geben einen Überblick über weitere bekannte Angriffe. Mehr Details finden sich in [38].

3.20.2 Time-Memory Trade-Off

Ist ein Klartext P mit dem zugehörigen Chiffretext C bekannt, dann kann die Rechenzeit für die vollständige Suche beschleunigt werden, wenn entsprechend mehr Speicherplatz verwendet wird. Hat der Schlüsselraum N Elemente dann benötigt der Time-Memory-Trade-Off-Algorithmus von M. Hellman [34] Zeit und Platz $O(N^{2/3})$ um den geheimen Schlüssel, der bei der Verschlüsselung von P verwendet wurde, zu finden. Der Algorithmus erfordert eine Vorberechnung, die Zeit $O(N)$ braucht. Eine Verallgemeinerung dieser Strategie findet man in [28].

Wir beschreiben den Algorithmus. Die Verschlüsselungsfunktion der Blockchiffre wird mit E bezeichnet. Der Schlüsselraum $\mathcal{K}$ der Blockchiffre, die angegriffen wird, hat N Elemente. Die Blockchiffre hat Blocklänge n. Das verwendete Alphabet ist Σ. Wir nehmen an, dass ein Klartext P und der zugehörige Schlüsseltext C bekannt sind. Gesucht ist der verwendete Schlüssel. Sei

$$m = \lceil N^{1/3} \rceil.$$

Wähle m Funktionen

$$g_k : \Sigma^n \to \mathcal{K}, \quad 1 \leq k \leq m$$

zufällig. Die Funktionen g_k machen aus Klartexten oder Chiffretexten Schlüssel.

Wähle m Schlüssel K_i, $1 \leq i \leq m$, zufällig und setze

$$K_k(i, 0) = K_i, 1 \leq i, k \leq m.$$

Damit können jetzt folgende Schlüsseltabellen berechnet werden:

$$K_k(i, j) = g_k(E(K_k(i, j-1), P)), \quad 1 \leq i, j, k \leq m. \tag{3.17}$$

Die k-te Schlüsseltabelle hat also die in Tab. 3.4 dargestellte Form.

Tab. 3.4 Die k-te Schlüsseltabelle im Time-Memory-Trade-Off

$K_k(1,0)$	$\xrightarrow{g_k}$	$K_k(1,1)$	$\xrightarrow{g_k}$	$\cdots$	$\xrightarrow{g_k}$	$K_k(1,m)$
$K_k(2,0)$	$\xrightarrow{g_k}$	$K_k(2,1)$	$\xrightarrow{g_k}$	$\cdots$	$\xrightarrow{g_k}$	$K_k(2,m)$
$\ldots$						$\ldots$
$K_k(m,0)$	$\xrightarrow{g_k}$	$K_k(m,1)$	$\xrightarrow{g_k}$	$\cdots$	$\xrightarrow{g_k}$	$K_k(m,m)$

Die Anzahl der Tabelleneinträge ist ungefähr N. Um den geheimen Schlüssel zu finden berechnen wir

$$g_k(c), 1 \leq k \leq m.$$

Wir prüfen, ob

$$K_k(i,j) = g_k(C)$$

gilt für Indizes $i, j, k \in \{1, \ldots, m\}$. Da

$$K_k(i,j) = g_k(E(K_k(i,j-1), P))$$

gilt, ist es gut möglich, dass $K_k(i,j-1)$ der gesuchte Schlüssel ist. Wir überprüfen also, ob

$$c = E(K_k(i,j-1), (x)$$

gilt. Wenn ja, ist der gesuchte Schlüssel $K(i,j-1)$.

So wie das Verfahren bis jetzt beschrieben worden ist, erfordert es Berechnung von ungefähr $N^{1/3}$ Schlüsseln aber die Speicherung von N Schlüsseln. Damit ist der Speicherplatzbedarf zu groß. Statt dessen werden tatsächlich nur die letzten Spalten der Tabellen, also die Werte $K_k(i,m)$, $1 \leq i \leq m$ gespeichert.

Um den Schlüssel zu finden, der aus einem Klartext P einen Schlüsseltext C macht, berechnen wir

$$K(1,k) = g_k(C), 1 \leq k \leq m$$

und dann

$$K(j,k) = g_k(E(K(j-1,k), P), \quad 1 \leq k \leq m, 1 < j \leq m.$$

Wir versuchen, einen dieser Schlüssel in unserer Tabelle zu finden. Wir suchen also Indizes $i, j, k \in \{1, \ldots, m\}$ mit der Eigenschaft

$$K(j,k) = K_k(i,m).$$

Sobald diese gefunden sind, liegt es nahe zu vermuten, dass $K(1,k) = K_k(i, m-j+1)$ und der verwendete Schlüssel $K_k(i, m-j)$ ist. Wir konstruieren diesen Schlüssel und prüften, ob $C = E(K_k(i, m-j), P)$ ist. Wenn ja, ist der gesuchte Schlüssel gefunden. Unter geeigneten Voraussetzungen kann man zeigen, dass auf diese Weise der gesuchte Schlüssel mit hoher Wahrscheinlichkeit gefunden wird. Die Anzahl der untersuchten Schlüssel ist ungefähr $N^{2/3}$. Speichert man die letzten Spalten der Tabellen als Hashtabellen, ist die gesamte Laufzeit $O(N^{2/3})$, wenn man davon ausgeht, dass die Berechnung jedes einzelnen Schlüssels und jeder Tabellenzugriff Zeit $O(1)$ benötigt.

3.20.3 Differentielle Kryptoanalyse

Die differentielle Kryptoanalyse [13] wurde 1990 von Biham und Shamir erfunden, um den DES anzugreifen. Diese Technik kann aber gegen Blockchiffren im allgemeinen angewendet werden. Differentielle Angriffe wurden zum Beispiel auch auf IDEA, SAFER K und Skipjack angewendet. Die Referenzen finden sich in [38].

Die differentielle Kryptoanalyse ist ein Chosen-Plaintext-Angriff. Aus vielen Paaren Klartext-Schlüsseltext versucht der Angreifer den verwendeten Schlüssel zu bestimmen. Dabei verwendet er die „Differenzen“ der Klar- und Schlüsseltexte, d. h. sind P und P' Klartexte und sind C und C' die zugehörigen Schlüsseltexte, dann berechnet der Angreifer $P \oplus P'$ und $C \oplus C'$. Er nutzt aus, dass in vielen Verschlüsselungsverfahren aus dem Paar $(P \oplus P', C \oplus C')$ Rückschlüsse auf den verwendeten Schlüssel gezogen werden können.

3.20.4 Algebraische Kryptoanalyse

In diesem Abschnitt beschreiben wir, wie das Problem, eine Blockchiffre zu brechen, auf die Aufgabe reduziert werden kann, ein System multivariater Gleichungssysteme zu lösen. Dies ist eine neue und Erfolg versprechende Technik der Kryptoanalyse.

Betrachte eine Blockchiffre mit Alphabet $\mathbb{Z}_2$, Blocklänge n und Schlüsselraum $\mathbb{Z}_2^m$ für $n, m \in \mathbb{N}$. Wir zeigen, dass die Verschlüsselungsfunktion $E(K, P)$, $K \in \mathbb{Z}_2^m$, $P \in \mathbb{Z}_2^m$ als Tupel von Polynomfunktionen mit Koeffizienten im Körper GF(2) in n Klartext-Variablen $p_1, \ldots, p_n$, und m Schlüssel-Variablen $k_1, \ldots, k_m$ geschrieben werden kann. Schreibe

$$p = (p_1, \ldots, p_n), k = (k_1, \ldots, k_m).$$

Für jeden Schlüssel $K = (K_1, \ldots, K_m) \in \mathbb{Z}_2^m$ und jeden Schlüsseltext $P = (P_1, \ldots, P_n) \in \mathbb{Z}_2^n$ definieren wir das Monom

$$M_{(P,K)} = \left(\prod_{i=1}^{n}(p_i + P_i + 1)\right)\left(\prod_{j=1}^{m}(k_j + K_j + 1)\right). \tag{3.18}$$

Beispiel 3.37 Sei $n = 2, m = 1$. Dann erhalten wir die folgenden Monome

$$\begin{aligned}
M_{(0,0),(0)} &= (p_1 + 1)(p_2 + 1)(k_1 + 1)\\
M_{(0,0),(1)} &= (p_1 + 1)(p_2 + 1)k_1\\
M_{(0,1),(0)} &= (p_1 + 1)p_2(k_1 + 1)\\
M_{(0,1),(1)} &= (p_1 + 1)p_2k_1\\
M_{(1,0),(0)} &= p_1(p_2 + 1)(k_1 + 1)\\
M_{(1,0),(1)} &= p_1(p_2 + 1)k_1
\end{aligned}$$

$$M_{(1,1),(0)} = p_1 p_2 (k_1 + 1)$$
$$M_{(1,1),(1)} = p_1 p_2 k_1$$

Wenn man in das Monom $M_{(P,K)}$ die Werte P und K für p und k einsetzt, dann erhält man

$$M_{(P,K)}(P, K) = 1, \tag{3.19}$$

weil in GF(2) die Regel $1 + 1 = 0$ gilt. Wenn man aber $P' \neq P$ und $K' \neq K$ für p und k einsetzt, dann erhält man

$$M_{(P,K)}(P', K') = 0, \tag{3.20}$$

weil wenigstens einer der Faktoren auf der rechten Seite in (3.18) den Wert Null hat. Setzt man also

$$P(p, k) = \sum_{P \in \mathrm{GF}(2)^n,\ K \in \mathrm{GF}(2)^m} E(K, P) M_{(P,K)}(p, k) \tag{3.21}$$

dann gilt tatsächlich

$$P(P, K) = E(K, P), \quad P \in \mathrm{GF}(2)^n, K \in \mathrm{GF}(2)^m. \tag{3.22}$$

Beispiel 3.38 Wir setzen Beispiel 3.37 fort. Sei $n = 2, m = 1$ und sei die Verschlüsselungsfunktion durch

P	00	01	10	11
$E(0, P)$	01	00	11	10
$E(1, P)$	11	10	01	00

definiert. Dann erhalten wir

$$P(p, k) = (P_1(p, k), P_2(p, k))$$

mit

$$\begin{aligned} P_1(p, k) = {} & (p_1 + 1)(p_2 + 1)k_1 + (p_1 + 1)p_2 k_1 \\ & + p_1(p_2 + 1)(k_1 + 1) + p_1 p_2 (k_1 + 1) \end{aligned}$$

und

$$\begin{aligned} P_2(p, k) = {} & (p_1 + 1)(p_2 + 1)(k_1 + 1) + (p_1 + 1)(p_2 + 1)k_1 \\ & + p_1(p_2 + 1)(k_1 + 1) + p_1(p_2 + 1)k_1 . \end{aligned}$$

Angenommen, ein Angreifer will einen Known-Plaintext-Angriff anwenden. Er weiß also Klartexte und zugehörige Schlüsseltexte. Aber die Schlüsselbits sind ihm unbekannt. Er kann dann (3.22) verwenden, um ein multivariates Gleichungssystem für diese Schlüsselbitvariablen aufzustellen. Solche multivariaten Systeme können mit Methoden aus der algorithmischen algebraischen Geometrie gelöst werden. Aber diese Lösung kann sehr aufwändig sein. Es gibt aber zahreiche Modifikationen dieser Methode, die die Gleichungssysteme deutlich einfacher machen.

Beispiel 3.39 Wir setzen Beispiel 3.38 fort. Angenommen, ein Angreifer weiß, dass der Klartext $(0, 0)$ den Schlüsseltext $(0, 1)$ liefert. Dann bekommt er folgende Polynomgleichungen

$$\begin{aligned} k_1 &= 0 \\ 1 &= 1. \end{aligned}$$

Daraus ergibt sich $k_1 = 0$.

3.21 Übungen

Übung 3.1 Der Schlüsseltext JIVSOMPMQUVQA wurde mit der Verschiebungschiffre erzeugt. Ermitteln Sie den Schlüssel und den Klartext.

Übung 3.2 Zeigen Sie, dass auf folgende Weise ein Kryptosystem definiert ist.

Sei P ein String über {A,B, . . . , Z}. Wähle zwei Schlüssel K_1 und K_2 für die Verschiebungschiffre. Verschlüssele Zeichen mit ungeradem Index unter Verwendung von K_1 und die mit geradem Index unter Verwendung von K_2. Dann kehre die Reihenfolge der Zeichen um.

Bestimmen Sie den Klartextraum, den Schlüsseltextraum und den Schlüsselraum.

Übung 3.3 Zeigen Sie, dass die Verschlüsselungsfunktionen eines Kryptosystems immer injektiv sind.

Übung 3.4 Bestimmen Sie die Anzahl der Wörter der Länge n über einem Alphabet Σ, die sich nicht ändern, wenn sie umgekehrt werden.

Übung 3.5 Sei Σ ein Alphabet. Zeigen Sie, dass die Menge Σ^* zusammen mit der Konkatenation eine Halbgruppe mit neutralem Element ist. Welches ist das neutrale Element? Ist diese Halbgruppe sogar eine Gruppe?

Übung 3.6 Wieviele verschiedene Verschlüsselungsfunktionen kann eine Blockchiffre mit Alphabet $\mathbb{Z}_2$ und Blocklänge n höchstens haben?

Übung 3.7 Welches der folgenden Systeme ist ein Verschlüsselungsverfahren? Geben Sie gegebenenfalls Klartextraum, Schlüsseltextraum, Schlüsselraum, Verschlüsselungs- und Entschlüsselungsfunktion an. In der Beschreibung werden Buchstaben aus Σ = {A,B,. . . ,Z} gemäß Tab. 3.1 durch Zahlen ersetzt.

1. Jeder Buchstabe σ aus Σ wird durch $k\sigma$ mod 26 ersetzt, $k \in \{1, 2, \ldots, 26\}$.
2. Jeder Buchstabe σ aus Σ wird durch $k\sigma$ mod 26 ersetzt, $k \in \{1, 2, \ldots, 26\}$, $\gcd(k, 26) = 1$.

Übung 3.8 Geben Sie ein Beispiel für ein Kryptosystem, das Verschlüsselungsfunktionen besitzt, die zwar injektiv aber nicht surjektiv sind.

Übung 3.9 Bestimmen Sie die Anzahl der Bitpermutationen der Menge $\mathbb{Z}_2^n$, $n \in \mathbb{N}$. Bestimmen Sie auch die Anzahl der zirkulären Links- und Rechtsshifts von $\mathbb{Z}_2^n$.

Übung 3.10 Eine *Transposition* ist eine Permutation, die zwei Elemente vertauscht und die anderen unverändert lässt. Zeigen Sie, dass jede Permutation als Komposition von Transpositionen dargestellt werden kann.

Übung 3.11 Geben Sie eine Permutation von $\mathbb{Z}_2^n$ an, die keine Bitpermutation ist.

Übung 3.12 Geben Sie eine Permutation von $\mathbb{Z}_2^n$ an, die nicht affin linear ist.

Übung 3.13 Sei X eine Menge. Man zeige, dass die Menge $S(X)$ der Permutationen von X eine Gruppe bezüglich der Hintereinanderausführung ist. Man zeige auch, dass diese Gruppe im allgemeinen nicht kommutativ ist.

Übung 3.14 Entschlüsseln Sie 111111111111 im ECB-Mode, im CBC-Mode, im CFB-Mode und im OFB-Mode. Verwenden Sie die Permutationschiffre mit Blocklänge 3 und Schlüssel

$$k = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}.$$

Der Initialisierungsvektor ist 000. Im OFB- und CFB-Mode verwenden Sie $r = 2$.

Übung 3.15 Verschlüsseln Sie den String 101010101010 im ECB-Mode, im CBC-Mode, im CFB-Mode und im OFB-Mode. Verwenden Sie die Permutationschiffre mit Blocklänge 3 und Schlüssel

$$k = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}.$$

Der Initialisierungsvektor ist 000. Im OFB- und CFB-Mode verwenden Sie $r = 2$.

Übung 3.16 Sei $k = 1010101$, $c = 1110011$ und $w = 1110001\ 1110001\ 1110001$. Verschlüsseln Sie w unter Verwendung der Stromchiffre aus Abschn. 3.12.

Übung 3.17 Man zeige, dass man die Stromchiffre, die mit Hilfe eines linearen Schieberegisters erzeugt wird, auch mit einer Blockchiffre im OFB-Mode erzeugen kann, sofern die Länge der verschlüsselten Wörter ein Vielfaches der Blocklänge und $C_1 = 1$ ist.

Übung 3.18 Bestimmen Sie die Determinante der Matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix}.$$

Übung 3.19 Geben Sie eine geschlossene Formel für die Determinante einer 3×3-Matrix an.

Übung 3.20 Finden Sie eine injektive affin lineare Abbildung $(\mathbb{Z}/2\mathbb{Z})^3 \to (\mathbb{Z}/2\mathbb{Z})^3$ die $(1, 1, 1)$ auf $(0, 0, 0)$ abbildet.

Übung 3.21 Bestimmen Sie die Inverse der Matrix

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

mod 2.

Übung 3.22 Geben Sie einen Schlüssel für eine affin lineare Chiffre mit Alphabet {A,B,C,...,Z} und Blocklänge 3 an, mit dem „ROT“ in „GUT“ verschlüsselt wird.