WŁADYSŁAW MOCHNACKI

KODY KOREKCYJNE
I
KRYPTOGRAFIA

Władysław Mochnacki

KODY KOREKCYJNE I KRYPTOGRAFIA

Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2000

Wydział Elektroniki Politechniki Wrocławskiej

Opiniodawcy:
Józef DUDEK
Czesław KOŚCIELNY

Opracowanie redakcyjne i korekta
Ewa SOBESTO

Wydanie drugie poprawione

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISBN

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam.

Spis treści

Wstęp

W okresie dynamicznego rozwoju techniki komputerowej i systemów teleinformatycznych duże znaczenie mają metody ochrony informacji przed błędami i nielegalnym dostępem. Zagadnienia te są rozpatrywane w teorii kodowania i kryptografii. Pierwsze prace z tego zakresu zostały opublikowane jeszcze w końcu lat czterdziestych przez Shannona, który zainicjował powstanie teorii informacji. Dzięki ogólnemu charakterowi prace Shannona miały pobudzający wpływ na badania dotyczące przekazywania i przechowywania informacji.

Teoria kodów korekcyjnych opiera się na algebrze ciał skończonych. Rozwój teorii kodowania doprowadził do skonstruowania efektywnych kodów korekcyjnych, ale metody te nie są jeszcze w pełni wykorzystywane w praktyce. Kody korekcyjne umożliwiają zwiększenie niezawodności informatycznego systemu cyfrowego, lecz ich zastosowanie powoduje wzrost objętości danych, co wiąże się z koniecznością zwiększenia przepustowości kanałów transmisyjnych i wielkości pamięci.

W kryptografii, która jest częścią kryptologii, wykorzystuje się teorię liczb. Przyspieszony rozwój nowoczesnych metod kryptograficznych nastąpił po wprowadzeniu do powszechnego użytku komputerów. W roku 1977 zaczęto stosować pierwszą normę kryptograficzną. Obecnie systemy kryptograficzne są powszechnie używane nie tylko w wojsku i dyplomacji, ale również we wszystkich sieciach teleinformatycznych.

Niniejszy podręcznik ma na celu przedstawienie w prosty sposób podstaw matematycznych teorii kodowania i kryptografii oraz pokazanie możliwości wykorzystania tych metod matematycznych we współczesnej technice. Zakres materiału zawarty w książce został dostosowany do wykładu z teorii kodów i kryptografii prowadzonego na Wydziale Elektroniki Politechniki Wrocławskiej. Można się spodziewać, że książka będzie ciekawą lekturą również dla osób interesujących się zastosowaniami matematyki i tych, którzy zechcą samodzielnie zapoznać się z zagadnieniami ochrony informacji przed błędami i nielegalnym dostępem w nowoczesnych systemach komputerowych i teleinformatycznych.

Podręcznik zawiera trzy części. Część pierwszą poświęcono wybranym elementom algebry ciał skończonych i teorii liczb, które są potrzebne do opisu kodów korekcyjnych i algorytmów kryptograficznych. Zapoznanie się z tym rozdziałem umożliwi zrozumienie dalszych części książki. Twierdzenia matematyczne podano bez dowodów, ale zaopatrzono w komentarze i przykłady wskazujące na ich zastosowania.

W części drugiej przedstawiono kody korekcyjne służące do wykrywania i korekcji błędów. Opisano kody blokowe umożliwiające korekcję zarówno błędów losowych, jak i grupowych ze szczególnym uwzględnieniem kodów cyklicznych, które najczęściej są stosowane w praktyce. Pokazano tu również realizację techniczną koderów i dekoderów z zastosowaniem cyfrowych układów logicznych.

Metodom ochrony informacji przed nielegalnym dostępem jest poświęcona część trzecia. Opisano w niej systemy kryptograficzne z kluczem tajnym i jawnym. Najwię-

cej uwagi zwrócono na algorytmy kryptograficzne stosowane w ochronie kryptograficznej kanałów transmisyjnych i pamięci komputerów. Wspomniano również o niektórych metodach łamania szyfrów.

Inicjatorem wykładów z teorii kodów i kryptografii na Politechnice Wrocławskiej, był Profesor Czesław Kościelny. Autor podręcznika wyraża mu głęboką wdzięczność za wprowadzenie do tej interesującej dziedziny wiedzy i za wieloletnią współpracę.

Część 1
Elementy algebry ciał skończonych

Pojęcie ciała skończonego wprowadzono do matematyki w XIX wieku. Do powstania tego pojęcia przyczynił się matematyk francuski Evariste Galois (1811-1832), który zajmował się teorią równań algebraicznych. Jego teoria, początkowo nie doceniana, wywarła ogromny wpływ na rozwój matematyki w XIX i XX wieku.

1.1. Wprowadzenie

1.1.1. Systemy algebraiczne

Zbiór z określonymi działaniami nazywamy systemem algebraicznym. Do najczęściej stosowanych klas systemów algebraicznych należą: grupy, pierścienie i ciała.

Ciałem nazywamy zbiór A, zawierający więcej niż jeden element, dla którego zdefiniowano operacje dodawania i mnożenia, oraz spełniający niżej podane aksjomaty:

A1. $\forall a,b \in A \quad \exists c \in A \quad a+b=c$ zamkniętość dodawania,

A2. $\forall a,b \in A \quad a+b=b+a$ przemienność dodawania,

A3. $\forall a,b,c \in A \quad a+(b+c)=(a+b)+c$ łączność dodawania,

A4. $\forall a \in A \quad a+0=0+a=a$ istnienie zera,

A5. $\forall a \in A \quad \exists b \in A \quad a+b=b+a=0$ istnienie elementu przeciwnego,

M1. $\forall a,b \in A \quad \exists c \in A \quad a \cdot b=c$ zamkniętość mnożenia,

M2. $\forall a,b \in A \quad a \cdot b=b \cdot a$ przemienność mnożenia,

M3. $\forall a,b,c \in A \quad a \cdot (b \cdot c)=(a \cdot b) \cdot c$ łączność mnożenia,

M4. $\forall a \in A \quad a \cdot 1=1 \cdot a=a$ istnienie jedynki,

M5. $\forall a \in A \quad \exists b \in A \quad a \cdot b=b \cdot a=1$ istnienie elementu odwrotnego,

D. $\forall a,b,c \in A \quad a \cdot (b+c)=(b+c) \cdot a=a \cdot b+a \cdot c$

rozdzielność mnożenia względem dodawania.

Aksjomaty A1÷A4 gwarantują, że zbiór A stanowi grupę przemienną względem dodawania zwaną grupą addytywną ciała A. Aksjomaty M1÷M5 orzekają, że zbiór elementów A różnych od zera stanowi grupę przemienną względem mnożenia zwaną grupą multyplikatywną ciała A.

Wybrane klasy systemów algebraicznych, stosowane w teorii kodów, pokazano w tabeli 1.1.1. W tabeli tej zaznaczono również aksjomaty określające daną klasę.

Tabela 1.1.1. Wybrane klasy systemów algebraicznych

System algebraiczny	A1	A2	A3	A4	A5	M1	M2	M3	M4	M5	D
Grupa addytywna przemienna	X	X	X	X	X						
Grupa multyplikatywna przemienna						X	X	X	X	X	
Pierścień	X	X	X	X	X	X	X	X	X		X
Ciało	X	X	X	X	X	X	X	X	X	X	X

Ciało może być skończone lub nieskończone w zależności od zbioru elementów ciała. Ciała skończone zwane są też ciałami Galois i oznacza się je przez $GF(q)$, co jest skrótem od Galois Field. Rozróżnia się *ciała skończone proste* (prime fields) i *rozszerzenia ciał skończonych* (extension fields). Rozszerzenia ciał skończonych w literaturze technicznej nazywane są *ciałami rozszerzonymi*. W niniejszym podręczniku obie te nazwy będą stosowane alternatywnie jako synonimy.

Ciała skończone proste mają liczbę elementów równą liczbom pierwszym p, a oznaczamy je przez $GF(p)$. Elementy takiego ciała są liczbami ze zbioru $\{0,1,2,\ldots,p-1\}$. Skończone ciało proste jest zatem skończonym zbiorem elementów z dwoma działaniami: dodawaniem i mnożeniem modulo p

$$\langle\{0,1,2,\ldots,p-1\},+,\cdot\rangle.$$

Zbiór wszystkich elementów $GF(p)$ wraz z dodawaniem modulo p stanowi przemienną skończoną grupę addytywną

$$\langle\{0,1,2,\ldots,p-1\},+\rangle.$$

Podobnie, zbiór wszystkich niezerowych elementów $GF(p)$ tworzy przemienną skończoną grupę multyplikatywną

$$\langle\{1,2,\ldots,p-1\},\cdot\rangle.$$

Liczba elementów rozszerzonych ciał skończonych jest równa potędze liczby pierwszej i elementy te oznaczamy przez $GF(q)$, gdzie $q=p^m$, a m jest liczbą naturalną. W ciałach rozszerzonych elementy ciała a, b i c nie są liczbami, a symbole działań + i · nie mają nic wspólnego z dodawaniem i mnożeniem liczb znanym z arytmetyki. Niezerowe elementy ciał rozszerzonych można wyrazić za pomocą potęg elementu pierwotnego α: $1,\alpha,\alpha^2,\ldots,\alpha^{q-2}$.

1.1.2. Kongruencje i arytmetyka modularna

Teoria kongruencji stanowi ważny dział teorii liczb i wiąże się z teorią ciał skończonych. Kongruencję lub przystawanie liczb a i b według modułu m (modulo m) zapisujemy następująco

$$a \equiv b \pmod m \quad \text{lub} \quad a \equiv_m b, \quad \text{gdy} \quad m \mid (a-b).$$

Liczba a przystaje do b wtedy, gdy m dzieli bez reszty $a-b$.

P r z y k ł a d 1.1.1.

$7 \equiv 1 \pmod 3, \quad 7 \equiv -2 \pmod 3, \quad 6 \equiv 0 \pmod 3,$

$x^3 \equiv x+1 \pmod{x^3+x+1}$ nad $GF(2)$.

Kongruencja jest:

- zwrotna: $a \equiv a \pmod m$,
- symetryczna: jeśli $a \equiv b \pmod m$, to $b \equiv a \pmod m$,
- przechodnia: jeśli $a \equiv c \pmod m$ i $b \equiv c \pmod m$, to $a \equiv b \pmod m$.

Relację dwuargumentową zwrotną, symetryczną i przechodnią nazywamy relacją równoważności.

Kongruencje można dodawać, odejmować i mnożyć stronami. Dla wszystkich liczb całkowitych a, b, c i każdej liczby naturalnej m

$$\text{jeśli} \quad a \equiv b \pmod m \quad \text{i} \quad c \equiv d \pmod m,$$
$$\text{to} \quad a+c \equiv b+d \pmod m \quad \text{i} \quad a \cdot c \equiv b \cdot d \pmod m.$$

Kongruencji nie można dzielić stronami. Ale elementy kongruencji można dzielić przez wspólny podzielnik. Dla wszystkich liczb całkowitych a, b, c i każdej liczby naturalnej m: jeśli $a \equiv b \pmod m$ i a, b, m mają wspólny podzielnik k, to

$$\frac{a}{k} \equiv \frac{b}{k} \left(\bmod \frac{m}{k} \right).$$

Do kongruencji odnosi się małe twierdzenie Fermata, które mówi, że jeśli p jest liczbą pierwszą, to dla a całkowitych niepodzielnych przez p

$$a^{p-1} \pmod p \equiv 1. \tag{1.1.1}$$

Z twierdzenia tego wynika, że

$$a^p \pmod p \equiv a.$$

Kongruencje mają nieskończenie wiele rozwiązań. Na przykład kongruencję $7 \equiv x \pmod 3$ spełniają wszystkie liczby $x = 1 + 3c$, gdzie c jest liczbą całkowitą dodatnią, ujemną lub zerem. W teorii kodów i kryptografii wykorzystujemy zwykle jedno z rozwiązań w przedziale $[0, m-1]$.

Jeżeli $a \equiv b \pmod m$, to b nazywamy resztą (residuum) a modulo m. W arytmetyce modularnej do oznaczenia reszty b z $a \bmod m$ w przedziale $[0, m-1]$ stosuje się zapis

$b = a \bmod m$. Jeśli $b \equiv a \pmod m$, to $b = a \bmod m$, ale nie odwrotnie. Ponadto $a \equiv b \pmod m$ wtedy i tylko wtedy, kiedy $a \bmod m = b \bmod m$. Liczby przystające mają te same reszty w przedziale $[0, m-1]$.

Podobnie jak liczby całkowite, również liczby całkowite mod m razem z operacjami dodawania i mnożenia tworzą pierścień przemienny. Prowadząc obliczenia w arytmetyce modularnej, możemy redukować pośrednie wyniki obliczeń mod m, a wynik będzie taki sam jak po redukcji wyniku końcowego. Jest to szczególnie przydatne w konstrukcji szyfrów z kluczem publicznym, kiedy używamy dużych liczb.

1.1.3. Funkcja Eulera

Funkcja Eulera jest często używana w algebrze ciał skończonych. Funkcja Eulera $\varphi(n)$ określa liczbę liczb naturalnych w zbiorze $\{1, 2, ..., n-1\}$ względnie pierwszych z n. Na przykład $\varphi(8) = 4$, gdyż w zbiorze liczb mniejszych od 8 tylko 1, 3, 5 i 7 są względnie pierwsze z 8. Liczby względnie pierwsze nie mają żadnego wspólnego podzielnika oprócz 1. Funkcja Eulera dla liczby pierwszej p jest równa $p-1$, gdyż wszystkie liczby mniejsze od p są względnie pierwsze z p.

Aby znaleźć wartość funkcji Eulera liczby złożonej n, rozkładamy ją na iloczyn potęg liczb pierwszych

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_m^{e_m}.$$

Wartość funkcji Eulera dla takiej liczby złożonej wylicza się ze wzoru

$$\varphi(n) = \prod_{i=1}^{m} p_i^{e_i - 1}(p_i - 1). \tag{1.1.2}$$

P r z y k ł a d 1.1.2.
Obliczenie funkcji Eulera liczby złożonej.
$n = 2646 = 2 \cdot 3^3 \cdot 7^2$, $\varphi(2646) = 1 \cdot 3^2 \cdot 2 \cdot 7 \cdot 6 = 756$.

Funkcję $\varphi(n)$ wykorzystuje się w uogólnieniu Eulera małego twierdzenia Fermata (1.1.1). Jeśli a i n są względnie pierwsze, to

$$a^{\varphi(n)} \pmod n \equiv 1. \tag{1.1.3}$$

Funkcja Eulera służy również do obliczania liczb odwrotnych modulo n. Liczbę odwrotną do a modulo n oznaczamy przez a^{-1}. Liczby te spełniają zależność $a\,a^{-1} \pmod n \equiv 1$. Jeśli a i n są względnie pierwsze, to liczba odwrotna a^{-1} wynosi

$$a^{-1} \equiv a^{\varphi(n)-1} \pmod n. \tag{1.1.4}$$

Na przykład, jeśli $a = 4$ i $n = 11$, to $4^{-1} = 4^9 \bmod 11 = 3$.

1.2. Ciała proste

1.2.1. Konstrukcja ciała prostego

Skończone ciała proste można skonstruować dla zbiorów liczbowych o liczbie elementów równej liczbie pierwszej p. Ciała takie oznaczamy symbolem $GF(p)$. Elementami ciała prostego są liczby: 0, 1, 2, ..., $p-1$. Działania w ciałach prostych są takie same jak działania arytmetyczne z operacją modulo p. Ciało proste jest więc ciałem reszt modulo p. Sumę S i iloczyn P dwóch elementów ciała prostego a i b określają zależności:

$$S \equiv a + b \pmod{p}, \tag{1.2.1}$$

$$P \equiv a \cdot b \pmod{p}. \tag{1.2.2}$$

Konstrukcja ciała polega na utworzeniu zbioru elementów ciała i wyznaczeniu tabliczek dodawania i mnożenia. W praktyce najczęściej nie korzystamy z tabliczek działań, lecz na bieżąco obliczamy sumy i iloczyny elementów ciała prostego. Ciała proste nie są stosowane bezpośrednio do konstrukcji kodów, ale służą do konstrukcji ciał rozszerzonych.

Przykłady ciał prostych podano w następnym punkcie. Tabliczka dodawania ciała skończonego jest *kwadratem łacińskim*. W kwadracie łacińskim we wszystkich kolumnach i wierszach każdy element ciała pojawiaja się tylko raz. Ta właściwość tabliczki dodawania wynika z aksjomatu zamkniętości dodawania A1, p. 1.1.1. Podobną właściwość ma tabliczka mnożenia w części zawierającej elementy grupy multyplikatywnej.

1.2.2. Przykłady ciał prostych

Ciało $GF(2)$

Najprostszym ciałem skończonym jest *ciało binarne* $GF(2)$. Działania w ciele $GF(2)$ i struktury tworzone nad tym ciałem są używane do opisu pracy komputerów i transmisji danych. Ciało $GF(2)$ jest ciałem prostym, a jego elementami są 0 i 1. Tabliczki działań ciała $GF(2)$ można wyznaczyć za pomocą wzorów (1.2.1) i (1.2.2). Tabliczki te pokazano w tabeli 1.2.1.

Tabela 1.2.1. Tabliczka dodawania i mnożenia ciała $GF(2)$

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

Dodawanie w ciele $GF(2)$ nazywa się dodawaniem modulo dwa. Zauważmy, że w ciele $GF(2)$ $1+1=0$, zatem $1=-1$. Oznacza to, że w ciele $GF(2)$ dodawanie jest

równoważne odejmowaniu, a w wielomianach utworzonych nad ciałem $GF(2)$ znak odejmowania można zastąpić znakiem dodawania. Operację dodawania modulo dwa realizuje się za pomocą bramki logicznej Ex-OR.

Nad ciałem $GF(2)$ można utworzyć wielomiany i skonstruować kody korekcyjne. Współczynnikami wielomianu utworzonego nad ciałem $GF(2)$ są elementy tego ciała, np. $x^3 + x + 1$.

Ciało $GF(7)$

Jako przykład wieloelementowego skończonego ciała prostego przyjmijmy ciało $GF(7)$. Elementami ciała $GF(7)$ są liczby: 0, 1, 2, 3, 4, 5, 6. Tabliczki działań ciała $GF(7)$ obliczamy z zależności (1.2.1) i (1.2.2). Pokazano je w tabeli 1.2.2.

Tabela 1.2.2. Tabliczki dodawania i mnożenia ciała $GF(7)$

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

·	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Element przeciwny ciała obliczamy za pomocą aksjomatu A5, p. 1.1.1; np. $2+5=0$, $2=-5$. Element odwrotny ciała obliczamy za pomocą aksjomatu M5, p. 1.1.1; np. $3 \cdot 5=1$, $3=1/5$.

1.2.3. Rząd multyplikatywny elementów ciała

Niezerowe elementy ciała charakteryzuje rząd multyplikatywny. *Rzędem multyplikatywnym* dowolnego elementu ciała *a* jest najmniejsza liczba naturalna *e* taka, że

$$a^e = 1. \tag{1.2.3}$$

Na przykład rzędem multyplikatywnym elementu 5 ciała $GF(7)$ jest 6, ponieważ $5^6=1$. Rząd mutyplikatywny elementu ciała $GF(p)$ jest dzielnikiem $p-1$.

Elementy ciała $GF(7)$ mają następujące rzędy multyplikatywne:

- element 1 – rząd multyplikatywny 1,
- elementy 2 i 4 – rząd multyplikatywny 3,
- elementy 3 i 5 – rząd multyplikatywny 6,
- element 6 – rząd multyplikatywny 2.

Elementy ciała *GF*(*p*) mające rząd multyplikatywny równy $p-1$ nazywamy *elementami pierwotnymi* ciała. Liczbę elementów pierwotnych *n* ciała *GF*(*p*) można określić z zależ-

ności

$$n = \varphi\left(p-1\right), \tag{1.2.4}$$

gdzie φ jest funkcją Eulera.

Każdy element niezerowy ciała generuje grupę cykliczną. Element pierwotny ciała generuje grupę multyplikatywną ciała. W tak utworzonej grupie będą wszystkie niezerowe elementy ciała. Elementy grupy multyplikatywnej o rzędzie multyplikatywnym większym od 1 i mniejszym od $p-1$ generują podgrupy multyplikatywne. Taka podgrupa zachowuje działania grupy.

Grupę cykliczną generowaną przez dowolny element ciała skończonego otrzymamy, biorąc kolejne potęgi tego elementu. Na przykład element 5 ciała $GF(7)$ generuje grupę multyplikatywną: 5, 4, 6, 2, 3, 1, gdyż kolejne potęgi elementu 5 wynoszą: 5, $5\cdot5=4$, $4\cdot5=6$, $6\cdot5=2$, $2\cdot5=3$, $3\cdot5=1$. Podobnie element 2 generuje podgrupę trzyelementową: 2, 4, 1.

1.2.4. Charakterystyka ciała

Charakterystyką ciała skończonego jest najmniejsza liczba naturalna n, taka że

$$\underbrace{x^i + x^i + \ldots + x^i}_{n} = 0, \tag{1.2.5}$$

gdzie x^i jest dowolnym niezerowym elementem ciała.

Ciało $GF(2)$ ma charakterystykę 2, ponieważ 1+1 = 0. Ciało $GF(7)$ ma charakterystykę 7, ponieważ np. $3+3+3+3+3+3+3=0$.

Jeśli liczba n nie istnieje, to charakterystyka ciała jest z definicji równa zero, np. ciała liczb wymiernych, rzeczywistych i zespolonych mają charakterystykę zero. W przypadku ciał skończonych charakterystyka ciała jest liczbą pierwszą, a ciało rozszerzone zachowują charakterystykę ciała prostego, nad którym zostało skonstruowane rozszerzenie.

1.3. Algebra wielomianów i przestrzeń wektorowa

1.3.1. Wielomiany nad ciałami skończonymi

W teorii kodowania są szeroko wykorzystywane wielomiany nad ciałami skończonymi. Wielomian stopnia m nad ciałem $GF(q)$ ma ogólną postać

$$p(x)=\sum_{i=0}^{m} a_i x^i = a_m x^m + a_{m-1}x^{m-1} + \ldots + a_1 x + a_0, \quad a_i \in GF(q). \tag{1.3.1}$$

Wielomian jest unormowany, gdy $a_m = 1$.

Wielomian (1.3.1) można zapisać również w postaci ciągu współczynników:

$$a_m, a_{m-1}, a_{m-2}, \ldots, a_1, a_0.$$

Wtedy potęga zmiennej określa miejsce współczynnika w ciągu. Pokazano to na poniższych przykładach.

Wielomian nad $GF(2)$: $x^3 + x + 1$, 1 0 1 1.

Wielomian nad $GF(7)$: $x^3 + 5x + 3$, 1 0 5 3.

Wielomian nad $GF(8)$: $x^4 + \alpha^6 x^2 + x + \alpha^2$, 1 0 α^6 1 α^2.

Na wielomianach nad ciałami skończonymi można wykonywać operacje algebraiczne, stosując wielomiany lub ciągi ich współczynników. Na przykład dzielenie wielomianów nad $GF(2)$ ma postać:

$$\begin{array}{rl|l}
 & x^3 + 0 + x + 1 & \\
x + 1 & x^4 + x^3 + x^2 + 0 + 1 & \\
 & x^4 + x^3 & \\
\hline
 & x^2 + 0 & \\
 & x^2 + x & \\
\hline
 & x + 1 & \\
 & x + 1 & \\
\hline
 & 0 &
\end{array}
\qquad\qquad
\begin{array}{rl|l}
 & 1\ 0\ 1\ 1 & \\
1\ 1 & 1\ 1\ 1\ 0\ 1 & \\
 & 1\ 1 & \\
\hline
 & 1\ 0 & \\
 & 1\ 1 & \\
\hline
 & 1\ 1 & \\
 & 1\ 1 & \\
\hline
 & 0 &
\end{array}$$

Każdy wielomian nad ciałem skończonym ma wielomian odwrotny. Dla wielomianu zapisanego w postaci ogólnej, podanej we wzorze (1.3.1), wielomian odwrotny $p^*(x)$ oblicza się następująco

$$p^*(x) = \frac{x^m}{a_0} \cdot p\left(\frac{1}{x}\right). \tag{1.3.2}$$

Dla wielomianów nad $GF(2)$, podanych w postaci ciągu współczynników, wielomian odwrotny można uzyskać, zapisując ciąg współczynników w odwrotnej kolej-

ności. Ponieważ wielomian $x^6 + x + 1$ możemy zapisać jako 1000011, więc wielomianem odwrotnym będzie 1100001.

P r z y k ł a d 1.3.1.
Obliczanie wielomianu odwrotnego.
Jako przykład wyznaczmy wielomian odwrotny do wielomianu

$$p(x) = x^6 + x + 1.$$

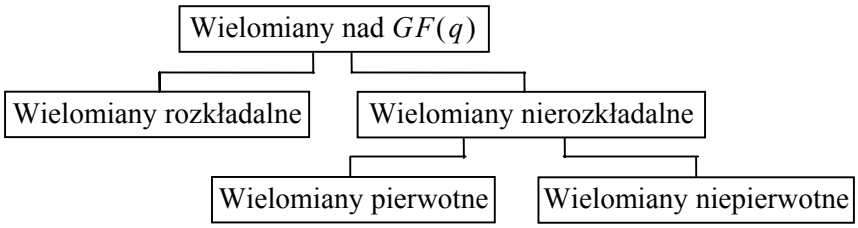
Wielomian odwrotny obliczamy z (1.3.2)

$$p^*(x) = x^6\left(\frac{1}{x^6} + \frac{1}{x} + 1\right) = x^6 + x^5 + 1.$$

Niektóre wielomiany odwrotne mają taką samą postać jak wielomiany oryginalne. Wielomiany takie nazywamy samoodwrotnymi. Na przykład wielomianami samoodwrotnymi są wielomiany: $x^2 + x + 1, \quad x^6 + x^3 + 1.$

1.3.2. Wielomiany nierozkładalne i pierwotne

Każdy wielomian stopnia dodatniego o współczynnikach z ciała skończonego $GF(q)$ jest albo nierozkładalny, albo jest iloczynem wielomianów nierozkładalnych nad ciałem $GF(q)$. Wielomian $p(x)$ stopnia m jest nierozkładalny nad $GF(q)$, jeśli nie dzieli się przez dowolny wielomian stopnia większego od zera i mniejszego od m. W przeciwnym przypadku wielomian jest rozkładalny. Wielomiany nad ciałem skończonym można zatem podzielić na typy pokazane na rys. 1.3.1.



Rys. 1.3.1. Typy wielomianów nad ciałem skończonym

P r z y k ł a d 1.3.2.
Wielomiany nierozkładalne nad $GF(2)$: $x + 1, \quad x^2 + x + 1, \quad x^4 + x + 1.$

Wielomian rozkładalny nad $GF(2)$: $x^4 + x^3 + x^2 + 1 = (x^3 + x + 1)(x + 1).$

Nie istnieją ogólne kryteria pozwalające w łatwy sposób odróżnić wielomiany rozkładalne od nierozkładalnych nad danym ciałem. Kryteria takie można podać tylko dla niektórych ciał, np. ciał liczb rzeczywistych, wymiernych i zespolonych.

Wielomiany nierozkładalne nad ciałami skończonymi odgrywają ważną rolę w teorii ciał skończonych i teorii kodów korekcyjnych. Służą one, między innymi, do konstruowania ciał rozszerzonych i niektórych kodów korekcyjnych. Nie ma ogólnego algorytmu wyznaczania wielomianów nierozkładalnych. Istnieją natomiast metody obliczania niektórych typów tych wielomianów. Metody te omówiono w [1.2]. Jedną z metod wyznaczania wielomianów nierozkładalnych jest metoda obliczania wielomianów minimalnych elementów ciała skończonego omówiona w p. 1.5.7. W tym celu należy skonstruować ciało skończone, a jeden z wielomianów pierwotnych, potrzebny do konstrukcji tego ciała, znaleźć metodą prób i błędów.

W praktyce nie zawsze musimy szukać wielomianów nierozkładalnych, gdyż możemy się posługiwać tablicami dołączonymi do podręczników dotyczących ciał skończonych lub kodów korekcyjnych. Niewielki zbiór wielomianów nierozkładalnych nad ciałami prostymi podano w p. 1.3.3.

Wielomiany nierozkładalne dzielą się na pierwotne i niepierwotne. Wielomian nierozkładalny stopnia m jest wielomianem pierwotnym, jeśli wszystkie jego pierwiastki są elementami pierwotnymi rozszerzenia stopnia m. Znajdowanie pierwiastków wielomianów jest jednak operacją złożoną. Zagadnienia te będą omówione w p. 1.5.7.

Można łatwo sprawdzić, czy wielomian nierozkładalny jest pierwotny, czy też nie, korzystając z metody polegającej na wygenerowaniu sekwencji okresowej i wyznaczenie okresu tej sekwencji. Metodę generowania sekwencji okresowej pokazano w p. 1.4.1. Jeśli okres sekwencji okresowej, wygenerowanej za pomocą danego wielomianu, osiąga wartość maksymalną podaną w (1.4.2), to możemy stwierdzić, że wielomian jest pierwotny.

Wielomiany pierwotne stopnia m istnieją dla każdej dodatniej liczby całkowitej m. Liczbę wielomianów pierwotnych stopnia m nad ciałem $GF(q)$ można obliczyć z zależności

$$N = \frac{\varphi\left(q^m - 1\right)}{m}, \tag{1.3.3}$$

gdzie $\varphi(x)$ jest funkcją Eulera.

Tabela 1.3.1. Liczba wielomianów pierwotnych nad $GF(2)$

m	1	2	3	4	5	6	10	15	20
2^m-1	1	3	7	15	31	63	1023	32767	1048575
N	1	1	2	2	6	6	60	1800	24000

Liczbę wielomianów pierwotnych N dla niektórych m nad ciałem $GF(2)$ podano w tabeli 1.3.1. W tabeli tej podano również okresy sekwencji okresowych generowanych przez wielomiany pierwotne. Zauważmy, że ze wzrostem m liczba wielomianów pierwotnych szybko rośnie.

1.3.3. Tablice wielomianów nierozkładalnych nad ciałami prostymi

W punkcie tym zebrano wybrane wielomiany nierozkładalne nad ciałami prostymi $GF(2)$, $GF(3)$, $GF(5)$ i $GF(7)$. Tablice zawierają ciągi współczynników wielomianów: $a_m, a_{m-1}, a_{m-2}, \ldots, a_1, a_0$ oraz okresy generowanych sekwencji. Wielomiany nierozkładalne, które generują sekwencje o okresie $p^m - 1$, są wielomianami pierwotnymi, a pozostałe – wielomianami niepierwotnymi. W tabelach pominięto wielomiany odwrotne. Tablice z większą liczbą wielomianów nierozkładalnych można znaleźć w [1.2, 2.4].

Tabela 1.3.2. Wielomiany nierozkładalne nad $GF(2)$

m	Wielomian – okres generowanej sekwencji			
1	11 – 1			
2	111 – 3			
3	1011 – 7			
4	10011 – 15	11111 – 5		
5	100101 – 31	101111 – 31	110111 – 31	
6	1000011 – 63	1001001 – 9	1010111 – 21	1011011 – 63
	1100111 – 63			
7	10000011 – 127	10001001 – 127	10001111 – 127	10011101 – 127
	10100111 – 127	10101011 – 127	10111111 – 127	11001011 – 127
	11101111 – 127			
8	100011011 – 51	100011101 – 255	100101011 – 225	100101101 – 255
	100111001 – 17	100111111 – 85	101001101 – 225	101011111 – 255
	101100011 – 255	101110111 – 85	101111011 – 85	110000111 – 255
	110001011 – 85	110011111 – 51	111001111 – 225	111010111 – 17

Tabela 1.3.3. Wielomiany nierozkładalne nad $GF(3)$

m	Wielomian – okres generowanej sekwencji					
1	11 – 1	12 – 2				
2	101 – 4	112 – 8				
3	1021 – 26	1022 – 13	1112 – 13	1121 – 26		
4	10012 – 80	10022 – 80	10102 – 16	10111 – 40	10121 – 40	11021 – 20
	11111 – 5	11122 – 80	12121 – 10	12212 – 80		

Tabela 1.3.4. Wielomiany nierozkładalne nad $GF(5)$

m	Wielomian – okres generowanej sekwencji					
1	11 – 1	12 – 4	14 – 2			
2	102 – 8	111 – 3	112 – 24	123 – 24	124 – 12	
	141 – 6					
3	1011 – 62	1014 – 31	1021 – 62	1024 – 31	1032 – 124	1033 – 124
	1042 – 124	1043 – 124	1113 – 124	1114 – 31	1131 – 62	1134 – 31
	1141 – 62	1143 – 124	1213 – 124	1214 – 31	1223 – 124	1312 – 124
	1323 – 124	1341 – 62				

Tabela 1.3.5. Wielomiany nierozkładalne nad $GF(7)$

m	Wielomian – okres generowanej sekwencji						
1	11 – 1	12 – 6	13 – 3	16 – 2			
2	101 – 4	102 – 12	113 – 48	114 – 24	116 – 16	123 – 48	125 – 48
	131 – 8	136 – 16	141 – 8	145 – 48	142 – 24		

1.3.4. Przestrzeń wektorowa

Uporządkowany zbiór n elementów v_i zapisanych w postaci wierszowej lub kolumnowej

$$\boldsymbol{v} = \left[v_1, v_2, \ldots, v_n\right], \quad \boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \ldots \\ v_n \end{bmatrix}$$

nazywamy wektorem n-wymiarowym. Jeśli współrzędnymi wektorów są elementy binarne, 1 lub 0, to możliwe jest wygenerowanie 2^n wektorów n-wymiarowych, które tworzą *przestrzeń wektorową* $\boldsymbol{V_n}$ nad $GF(2)$. Przestrzeń wektorowa odgrywa ważną rolę w teorii kodowania.

W teorii kodowania stosuje się najczęściej dodawanie wektorów i mnożenie przez skalar. Sumę dwóch wektorów $\boldsymbol{v}$ i $\boldsymbol{u}$ oblicza się w następujący sposób:

$$\begin{aligned} \boldsymbol{v} &= \left[v_1, v_2, \ldots, v_n\right], \\ \boldsymbol{u} &= \left[u_1, u_2, \ldots, u_n\right], \end{aligned}$$

$$\boldsymbol{v} + \boldsymbol{u} = \left[v_1 + u_1, v_2 + u_2, \ldots, v_n + u_n\right].$$

Iloczyn liczby a i wektora $\boldsymbol{v}$ jest wektorem o postaci

$$a\,\boldsymbol{v} = \left[a\,v_1, a\,v_2, \ldots, a\,v_n\right].$$

Podane działania na wektorach są działaniami przemiennymi i łącznymi.

Układ wektorów $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ należących do przestrzeni wektorowej $\boldsymbol{V_n}$ nazywamy *liniowo zależnym*, jeśli istnieją takie liczby $a_1, a_2, \ldots, a_k$ nie wszystkie jednocześnie równe zeru, że

$$a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \ldots + a_n\boldsymbol{v}_n = 0.$$

Na to, aby układ wektorów $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ był liniowo zależny, potrzeba i wystarczy, że jeden wektor tego układu był liniową kombinacją pozostałych wektorów.

Układ wektorów $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, gdzie $\boldsymbol{v}_i \in \boldsymbol{V}_n$, nazywamy *liniowo niezależnym*, jeśli powyższa równość zachodzi tylko wówczas, kiedy $a_1 = a_2 = \ldots = a_k = 0$. Zbiór n-wymiarowych wektorów $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ tworzy *bazę* n-wymiarowej przestrzeni, jeśli:

1. Liczba wektorów $\boldsymbol{v}_i$ w danym układzie wektorów jest identyczna z wymiarem przestrzeni, z której pochodzą te wektory.
2. Układ wektorów jest liniowo niezależny.

Wektory należące do przestrzeni wektorowej $\boldsymbol{V_n}$ można przedstawić jako kombinację liniową wektorów bazy.

P r z y k ł a d 1.3.3.
Przestrzeń wektorowa.
Niech $n = 3$. Przestrzeń wektorowa $\boldsymbol{V}_3$ zawiera następujące wektory:

[0 0 0], [0 0 1], [0 1 0], [0 1 1],

[1 0 0], [1 0 1], [1 1 0], [1 1 1].

Wektorami liniowo niezależnymi są np. wektory: [0 0 1], [0 1 0], [1 0 0]. Wektory te tworzą bazę przestrzeni wektorowej.

1.4. Sekwencje okresowe nad ciałami prostymi

1.4.1. Generowanie sekwencji okresowych

Każdy wielomian nad ciałem skończonym może być użyty do generowania nieskończonej sekwencji okresowej. Aby wygenerować sekwencję okresową, należy napisać zależność rekurencyjną stowarzyszoną z wybranym wielomianem. Rozważmy wielomian unormowany nad ciałem skończonym w postaci ogólnej (1.3.1)

$$p(x) = x^m + a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_1 x + a_0 \quad \text{nad } GF(q).$$

Przyrównując ten wielomian do zera, otrzymamy

$$x^m = -a_{m-1}x^{m-1} - a_{m-2}x^{m-2} - \ldots - a_1 x - a_0 .$$

Zależność rekurencyjna stowarzyszona z tym wielomianem będzie

$$s_{j+m} = -a_{m-1}s_{j+m-1} - a_{m-2}s_{j+m-2} - \ldots - a_1 s_{j+1} - a_0 s_j , \quad j = 0,1,2,3,\ldots \tag{1.4.1}$$

Działania należy tu wykonywać zgodnie z zasadami rachowania w ciele $GF(q)$. Gdy założymy ciąg początkowy o długości *m* elementów: $s_0, s_1, s_2, \ldots, s_{m-1}$, wówczas dla kolejnych wartości j można obliczyć z zależności (1.4.1) elementy sekwencji okresowej.

Okres wygenerowanej sekwencji okresowej zależy od typu wielomianu. W przypadku wielomianów pierwotnych sekwencja osiąga okres maksymalny. Okres maksymalny *M* dla wielomianu stopnia *m* nad ciałem $GF(q)$ wynosi

$$M = q^m - 1. \tag{1.4.2}$$

Wielomiany niepierwotne generują sekwencje o okresie mniejszym od *M*.

P r z y k ł a d 1.4.1.

Generowanie sekwencji okresowej nad ciałem binarnym.

Niech wielomianem generującym sekwencję okresową będzie wielomian stopnia trzeciego nad ciałem $GF(2)$

$$x^3 + x + 1 = 0.$$

Zależność rekurencyjna stowarzyszona z tym wielomianem ma postać

$$s_{j+3} = s_j + s_{j+1}, \qquad j = 0,1,2,3,\ldots$$

W zależności tej dodawanie jest zgodne z zasadami obowiązującymi w ciele $GF(2)$. Zakładając trzy elementy początkowe: s_0, s_1 i s_2, za pomocą powyższej zależności można wygenerować nieskończoną sekwencję okresową. Pierwszymi elementami takiej sekwencji będą elementy ciągu początkowego, a dalsze elementy sekwencji oblicza się z zależności rekurencyjnej, podstawiając kolejne wartości *j*. Proces ten ilustruje tabela 1.4.1. W pierwszym wierszu tabeli podano wartości parametru *j*.

Drugi wiersz zawiera symbole elementów ciągu s_i, a ostatni wiersz – wygenerowaną sekwencję. Okres wygenerowanej sekwencji, pokazanej w tabeli 1.4.1, wynosi siedem. Z zależności (1.4.2) możemy wnioskować, że zastosowany wielomian jest wielomianem pierwotnym.

Tabela 1.4.1. Generowanie sekwencji okresowej

Ciąg początkowy			j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	...
s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	...
1	0	0	1	0	1	1	1	0	0	1	...

Wielomiany niepierwotne nie generują sekwencji o maksymalnym okresie. Przykładem takiego wielomianu jest wielomian stopnia czwartego nad $GF(2)$

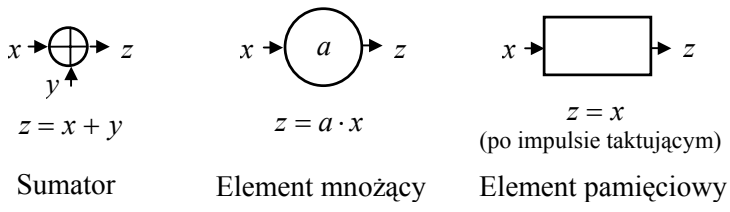
$$x^4 + x^3 + x^2 + x + 1.$$

Wielomian ten generuje sekwencję o okresie pięć: 1 0 0 0 1 1 0 0 0 ...

Sekwencje okresowe, generowane przez wielomiany pierwotne, mają maksymalny okres definiowany wzorem (1.4.2) i nazywają się *sekwencjami pseudolosowymi*. Sekwencje pseudolosowe spełniają ważną rolę w teorii kodów korekcyjnych i kryptografii i będą tu dokładnie omówione. Ciąg początkowy generowanej sekwencji pseudolosowej może być dowolny. Każda z $2^m - 1$ możliwych kombinacji początkowych, bez kombinacji złożonej z samych zer, umożliwia wygenerowanie sekwencji okresowej o tym samym okresie i kolejności elementów. Będą one tylko przesunięte względem siebie. Jeżeli ciąg początkowy będzie zawierał same zera, to zostanie wygenerowana sekwencja zawierająca same zera.

1.4.2. Generatory sekwencji okresowych

Generowanie sekwencji okresowych oraz realizacja różnych algorytmów z zakresu kodów korekcyjnych i kryptografii może się odbywać programowo lub sprzętowo. W realizacji sprzętowej stosujemy standardowe układy logiczne, jak: bramki, przerzutniki i rejestry.

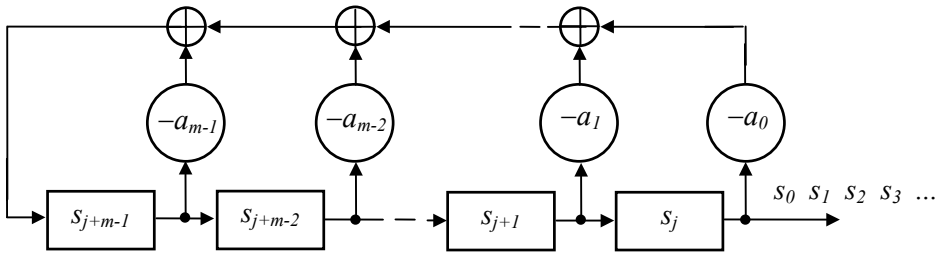


Rys. 1.4.1. Oznaczenia elementów na schematach blokowych

W niniejszym podręczniku ograniczono się do pokazania schematów blokowych realizowanych urządzeń bez opisu ich konstrukcji. Na schematach blokowych przyjęto oznaczenia elementów pokazane na rys. 1.4.1. Oznaczenia te są uniwersalne i mogą być stosowane w urządzeniach realizujących algorytmy zarówno nad ciałami binarnymi, jak i ciałami rozszerzonymi.

Jeśli urządzenie realizuje algorytm nad ciałem binarnym, to sumator odpowiada bramce Ex-OR, a element pamięciowy jest przerzutnikiem bistabilnym. Zasady konstrukcji elementów realizujących algorytmy nad ciałami rozszerzonymi omówionow p. 1.4.4.

Rekurencja (1.4.1), służąca do generowania sekwencji okresowej nad ciałem skończonym, może być zrealizowana za pomocą rejestru przesuwnego ze sprzężeniem zwrotnym (Linear Feedback Shift Register – LFSR) pokazanym na rys. 1.4.2.



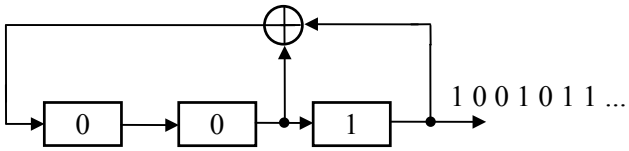
Rys. 1.4.2. Generator sekwencji okresowej

Konfiguracja sprzężenia zwrotnego będzie określona zależnością rekurencyjną. Ciąg początkowy jest wpisany do przerzutników rejestru w chwili startu układu. W takt impulsów zegarowych będzie generowana sekwencja okresowa, której kolejne elementy pojawią się na wyjściu generatora.

W przykładzie 1.4.1 pokazano sposób generowania sekwencji okresowej za pomocą zależności rekurencyjnej

$$s_{j+3} = s_j + s_{j+1}, \qquad j = 0,1,2,3,\ldots$$

Realizację tej zależności za pomocą układów logicznych pokazano na rys. 1.4.3.



Rys. 1.4.3. Generator binarnej sekwencji okresowej

1.4.3. Właściwości binarnych sekwencji pseudolosowych

W tym punkcie przestawimy właściwości binarnych sekwencji pseudolosowych. Sekwencje niebinarne będą omówione w p. 1.6.2. Właściwości sekwencji pseudolosowych i tablic zostały opisane w [1.3].

Zakładamy, że wielomian pierwotny $p(x)$ stopnia m generuje zbiór $\delta(m)$ sekwencji okresowych, zawierający $2^m - 1$ sekwencji o okresie $2^m - 1$ oraz sekwencję zerową. W punkcie tym będziemy się posługiwać przykładami sekwencji generowanymi przez wielomiany pierwotne czwartego i piątego stopnia:

1. $x^4 + x + 1$, 000100110101111,0001...
2. $x^5 + x^2 + 1$, 00001001011001111100011011101 01,00001...

Przecinek w ciągu oznacza okres sekwencji. Na przykład wielomian $x^4 + x + 1$ generuje zbiór sekwencji pokazany w tabeli 1.4.1.

Tabela 1.4.1. Zbiór sekwencji wygenerowanych przez wielomian $x^4 + x + 1$

000000000000000,	100010011010111,
000100110101111,	100110101111000,
001001101011110,	101011110001001,
001101011110001,	101111000100110,
010011010111100,	110001001101011,
010111100010011,	110101111000100,
011010111100010,	111000100110101,
011110001001101,	111100010011010.

Omawiane właściwości pseudolosowych sekwencji binarnych można sprawdzić, posługując się tabelą 1.4.1 lub podanymi wyżej przykładami. W koniecznych przypadkach zamieszczono dodatkowe wyjaśnienia.

1. Przesunięcie cykliczne

Jeśli $S = s_0\, s_1\, s_2 \ldots s_{2^m-2}$ jest sekwencją pseudolosową w zbiorze $\delta(m)$, to każde przesunięcie cykliczne $s_j\, s_{j+1} \ldots s_{2^m-2}\, s_0 \ldots s_{j-1}$ jest również w zbiorze $\delta(m)$. Po przesunięciu cyklicznym, np. w prawo, element ostatni przechodzi na pierwszą pozycję.

2. Rekurencja

Każda sekwencja $S \in \delta(m)$ spełnia zależność rekurencyjną

$$s_{j+m} = a_{m-1}s_{j+m-1} + a_{m-2}s_{j+m-2} + \ldots + a_1 s_{j-1} + a_0 s_j \quad \text{dla} \quad j = 0,1,2,3,\ldots$$

3. Właściwość okna

Jeśli okno o długości m będzie przesuwane wzdłuż sekwencji pseudolosowej, to każda z $2^m - 1$ niezerowych kombinacji binarnych m-elementowych wystąpi tylko raz. Właściwość okna ilustruje poniższy przykład dla sekwencji generowanej przez wielomian czwartego stopnia, $m = 4$

0 0 0 1 0 0 1 1 0 1 0 1 1 1 1 0 0 0.

4. Rozkład zer i jedynek

Każda sekwencja pseudolosowa $S \in \delta(m)$ zawiera 2^{m-1} jedynek i $2^{m-1}-1$ zer.

5. Właściwość addytywna

Suma dwóch sekwencji w zbiorze $\delta(m)$ należy również do zbioru $\delta(m)$. Elementy sekwencji dodaje się modulo 2.

6. Suma sekwencji i jej przesunięcia cyklicznego

Suma sekwencji pseudolosowej $S \in \delta(m)$ i jej przesunięcia cyklicznego daje sekwencję należącą do zbioru $\delta(m)$.

7. Funkcja autokorelacji

Funkcja autokorelacji procesu stacjonarnego charakteryzuje związki probabilistyczne podczas przesunięcia w czasie. Funkcję autokorelacji $k(i)$ sekwencji $s_0, s_1, \ldots, s_{m-1}$ definiuje się następująco

$$k(i) = \frac{1}{m}\sum_{j=0}^{m-1} s_j \cdot s_{i+j}, \quad i = 0, \pm 1, \pm 2, \ldots \tag{1.4.3}$$

Dla sekwencji binarnych funkcję autokorelacji wyznaczamy, biorąc pod uwagę sekwencją i jej przesunięcie cykliczne o i pozycji. Wtedy powyższy wzór przyjmuje postać

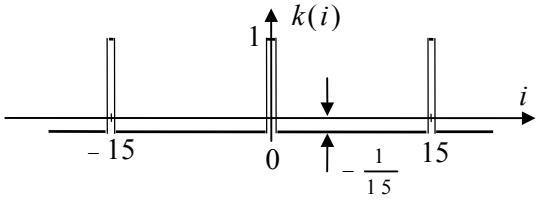
$$k(i) = \frac{A-D}{N},$$

gdzie:

A – liczba pozycji zgodnych, zawierających jednakowe znaki w sekwencji i jej przesunięciu cyklicznym,

D – liczba pozycji niezgodnych, zawierających różne znaki,

N – okres sekwencji.



Rys. 1.4.4. Funkcja autokorelacji sekwencji pseudolosowej o okresie 15

Funkcja autokorelacji sekwencji pseudolosowej o długości $N = 2^m - 1$ wynosi

$$k(jN) = 1 \quad \text{dla} \quad j = 0, \pm 1, \pm 2, \ldots,$$

$$k(i) = -\frac{1}{N} \quad \text{dla} \quad i \neq jN.$$

Funkcję autokorelacji dla sekwencji pseudolosowej o okresie $N = 15$ pokazano na rys. 1.4.4.

8. Ciągi jednakowych znaków

Sekwencję pseudolosową można podzielić na ciągi jednakowych znaków (runs). Na przykład sekwencja o okresie 31 dzieli się na szesnaście takich ciągów zaznaczonych przez podkreślenie.

0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1

W każdej sekwencji pseudolosowej:

- połowa wszystkich ciągów ma długość jednego znaku,
- jedna czwarta wszystkich ciągów ma długość dwóch znaków,
- jedna ósma wszystkich ciągów ma długość trzech znaków itd., dopóki te ułamki dają całkowitą liczbę ciągów.

Ponadto najdłuższy ciąg zer ma długość $m - 1$ znaków, a najdłuższy ciąg jedynek ma długość m znaków. W każdym przypadku liczba ciągów zer jest równa liczbie ciągów jedynek. Liczbę ciągów zer lub jedynek można obliczyć z zależności: 2^{m-k-2}, gdzie k jest długością sekwencji i spełnia zależność: $0 < k < m - 2$. Rozkład ciągów jednakowych znaków dla wyżej podanej sekwencji pseudolosowej pokazano w tabeli 1.4.2.

Tabela 1.4.2. Rozkład ciągów jednakowych znaków dla sekwencji o długości 31 znaków

Długość ciągu	Ciągi zer	Ciągi jedynek
1	4	4
2	2	2
3	1	1
4	1	0
5	0	1

Sekwencje pseudolosowe mają właściwości zbliżone do ciągów losowych, chociaż nie są w pełni ciągami losowymi. Mogą one jednak być generowane w komputerach i znajdują liczne zastosowania w kryptografii i kodach korekcyjnych.

1.5. Konstrukcja i struktura ciał rozszerzonych

1.5.1. Ciała rozszerzone

Ciała proste nie są stosowane bezpośrednio do konstrukcji kodów. Służą one natomiast do konstrukcji ciał rozszerzonych. Niech ciało $K^{(m)}$ będzie rozszerzeniem stopnia m ciała K. Ciało $K^{(m)}$ jest rozszerzeniem ciała K, jeśli zbiór elementów ciała K jest podzbiorem zbioru elementów ciała $K^{(m)}$ i działania w ciele K pochodzą od działań w ciele $K^{(m)}$. Ciało K nazywamy podciałem ciała $K^{(m)}$.

W algebrze ciał skończonych ciała rozszerzone $GF(q)$ można konstruować nad ciałami prostymi $GF(p)$ lub ciałami rozszerzonymi niższego stopnia. Liczba elementów q ciała rozszerzonego $GF(q)$ nad ciałem prostym $GF(p)$ wynosi $q=p^m$, gdzie m jest *stopniem rozszerzenia*. Ciała rozszerzone nie są ciałami liczbowymi, a ich elementy oznaczamy zwykle za pomocą symboli nieliczbowych.

Przykładami ciał rozszerzonych mogą być ciała: $GF(8)$ i $GF(16)$. Ciało $GF(8)$ jest rozszerzeniem trzeciego stopnia nad ciałem $GF(2)$, a ciało $GF(16)$ może być rozszerzeniem czwartego stopnia nad ciałem $GF(2)$ lub rozszerzeniem drugiego stopnia nad ciałem $GF(4)$.

Do konstrukcji ciała rozszerzonego $K^{(m)}$ używa się *elementu algebraicznego* α względem ciała K. Element α nazywamy algebraicznym względem ciała K, gdy istnieje niezerowy wielomian o współczynnikach z ciała K, którego pierwiastkiem jest element α. *Stopniem elementu algebraicznego* α względem ciała K nazywamy stopień wielomianu nierozkładalnego nad ciałem K, którego pierwiastkiem jest α.Taki wielomian nierozkładalny nazywamy *wielomianem minimalnym* elementu α.

Do konstrukcji ciała rozszerzonego najwygodniej używać elementów o rzędzie multyplikatywnym p^m-1, gdyż elementy te generują całą grupę multyplikatywną ciała. Elementy takie nazywamy *elementami pierwotnymi* ciała. W każdym ciele skończonym istnieje $\varphi(q-1)$ elementów pierwotnych, gdzie $\varphi(x)$ oznacza funkcję Eulera.

Przykładem najprostszego ciała rozszerzonego jest ciało $GF(4)$. Jest to rozszerzenie drugiego stopnia nad ciałem $GF(2)$. Elementy ciała $GF(4)$ można oznaczyć: 0, 1, α, α^2. Tabliczki działań ciała $GF(4)$ pokazano w tabeli 1.5.1.

Element algebraiczny α ma wielomian minimalny

$$m_1(x) = x^2 + x + 1 \quad \text{nad } GF(2).$$

Wielomian ten jest wielomianem pierwotnym nad $GF(2)$. Symbol $m_1(x)$ jest przyjętym oznaczeniem wielomianu minimalnego. Stopień elementu algebraicznego α wynosi dwa. Łatwo sprawdzić, że α jest pierwiastkiem wielomianu minimalnego. Jeśli w wielomianie minimalnym za x podstawimy α, otrzymamy

$$\alpha^2 + \alpha + 1 = 0.$$

Biorąc wartości sum z tabliczki dodawania ciała $GF(4)$, można stwierdzić, że powyższe równanie jest spełnione.

Tabela 1.5.1. Tabliczki dodawania i mnożenia ciała $GF(4)$

+	0	1	α	α^2
0	0	1	α	α^2
1	1	0	α^2	α
α	α	α^2	0	1
α^2	α^2	α	1	0

·	0	1	α	α^2
0	0	0	0	0
1	0	1	α	α^2
α	0	α	α^2	1
α^2	0	α^2	1	α

Metody obliczania tabliczek działań ciał rozszerzonych będą omówione w następnych punktach.

1.5.2. Konstrukcja rozszerzonych ciał skończonych

W przypadku ciał skończonych do konstrukcji ciał rozszerzonych stosuje się wielomiany pierwotne. Wtedy pierwiastek wielomianu pierwotnego α jest elementem pierwotnym ciała, a niezerowe elementy ciała rozszerzonego w postaci multyplikatywnej możemy zapisać jako potęgi elementu pierwotnego. Na przykład elementami ciała rozszerzonego $GF(q)$ będą: $0,1,\alpha,a^2,\ldots,\alpha^{q-2}$. Konstrukcja ciała rozszerzonego ma na celu wyznaczenie tabliczek dodawania i mnożenia ciała rozszerzonego.

Postać multyplikatywna elementów ciała skończonego pozwala wyznaczyć iloczyn dwóch dowolnych elementów ciała. Iloczyn dwóch elementów ciała skończonego α^i i α^j wynosi

$$\alpha^i \cdot \alpha^j = \alpha^{i+j(\mathrm{mod}\ q-1)}. \tag{1.5.1}$$

Obliczając iloczyny dla kolejnych elementów ciała, można wyznaczyć całą tabliczkę mnożenia. Tabliczkę mnożenia ciała $GF(8)$ pokazano w tabeli 1.5.2.

Aby wyznaczyć tabliczkę dodawania, należy zapisać elementy ciała skończonego w postaci addytywnej. Może to być postać macierzowa, wektorowa lub wielomianowa.

Niech $p(x)$ będzie wielomianem pierwotnym nad ciałem $GF(2)$, który posłuży do konstrukcji ciała rozszerzonego

$$p(x) = x^m + a_{m-1}x^{m-1} + \ldots + a_1 x + a_0. \tag{1.5.2}$$

Pierwiastek α wielomianu $p(x)$ można obliczyć z macierzy Cayleya–Hamiltona (1.5.3). W macierzy tej przekątna leżąca na prawo od przekątnej głównej zawiera jedynki, a ostatni wiersz jest utworzony ze współczynników wielomianu pierwotnego. Pierwiastek α jest elementem pierwotnym ciała rozszerzonego, dlatego też postać macierzową elementów ciała rozszerzonego można obliczyć, biorąc kolejne potęgi elementu pierwotnego.

$$\alpha = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & -a_3 & \dots & -a_{m-1} \end{bmatrix}. \tag{1.5.3}$$

Do obliczenia postaci wektorowej elementów ciała rozszerzonego można wykorzystać sekwencję pseudolosową generowaną przez wielomian pierwotny $p(x)$ nad ciałem podstawowym. Wielomianową postać elementów rozszerzonego ciała skończonego wyznaczamy z kongruencji

$$x^i \equiv R_i (\text{mod p}(x)). \tag{1.5.4}$$

Sposób obliczania elementów ciała rozszerzonego i tabliczek działań pokażemy na przykładzie ciała $GF(2^3)$ generowanego przez wielomian pierwotny $p(x)$ trzeciego stopnia nad $GF(2)$

$$p(x) = x^3 + x + 1. \tag{1.5.5}$$

Wielomian ten umożliwia konstrukcję ciała rozszerzonego $GF(2^3)$ zawierającego osiem elementów. Niżej pokazano sposoby wyznaczania elementów ciała rozszerzonego $GF(2^3)$ w postaci addytywnej za pomocą macierzy, wektorów i wielomianów.

1.5.3. Elementy ciała skończonego w postaci macierzy

Element pierwotny ciała rozszerzonego $GF(2^3)$ można obliczyć z macierzy Cayleya–Hamiltona. W tym celu podstawiamy współczynniki wielomianu pierwotnego (1.5.5) do (1.5.3) i otrzymujemy

$$\alpha = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Kolejne elementy ciała w postaci macierzowej wyznaczamy, wykorzystując element pierwotny α. Na przykład α^2 będzie

$$\alpha^2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

Pozostałe elementy ciała rozszerzonego $GF(8)$ będą miały postać:

$$0=\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix},\qquad 1=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix},\qquad \alpha^3=\begin{bmatrix}1&1&0\\0&1&1\\1&1&1\end{bmatrix},$$

$$\alpha^4=\begin{bmatrix}0&1&1\\1&1&1\\1&0&1\end{bmatrix},\qquad \alpha^5=\begin{bmatrix}1&1&1\\1&0&1\\1&0&0\end{bmatrix},\qquad \alpha^6=\begin{bmatrix}1&0&1\\1&0&0\\0&1&0\end{bmatrix}.$$

Elementy ciała w postaci macierzowej umożliwiają wyznaczenie sumy dwóch dowolnych elementów ciała rozszerzonego. Na przykład suma elementów α i α^2 będzie

$$\alpha+\alpha^2=\begin{bmatrix}0&1&0\\0&0&1\\1&1&0\end{bmatrix}+\begin{bmatrix}0&0&1\\1&1&0\\0&1&1\end{bmatrix}=\begin{bmatrix}0&1&1\\1&1&1\\1&0&1\end{bmatrix}=\alpha^4.$$

W podobny sposób można wyznaczyć całą tabliczkę dodawania. Tabliczkę dodawania ciała *GF*(8) pokazano w tabeli 1.5.2.

1.5.4. Elementy ciała skończonego w postaci wektorów

Wektorowa postać elementów ciała rozszerzonego jest wygodna, gdy konstruujemy ciało metodami programowymi. W celu wyznaczenia wektorowej postaci elementów ciała skończonego *GF*(8) wykorzystujemy sekwencję pseudolosową generowaną przez wielomian pierwotny służący do konstrukcji ciała rozszerzonego. Aby wygenerować sekwencję okresową, piszemy zależność rekurencyjną stowarzyszoną z wielomianem (1.5.5)

$$s_{j+3}=s_j+s_{j+1},\qquad j=0,1,2,3,\ldots$$

Zakładając ciąg początkowy 100, otrzymamy następującą sekwencję pseudolosową

$$1\,0\,0\,1\,0\,1\,1\,1\,0\,0\ldots$$

Wielomian (1.5.5) daje rozszerzenie trzeciego stopnia, dlatego też wektory odpowiadające elementom ciała rozszerzonego będą zawierały po trzy współrzędne. Biorąc kolejne grupy trzyelementowe, z powyższej sekwencji pseudolosowej otrzymamy elementy ciała rozszerzonego w postaci wektorowej:

$$0=[000],\quad 1=[100],\quad \alpha=[001],\quad \alpha^2=[010],$$
$$\alpha^3=[101],\quad \alpha^4=[011],\quad \alpha^5=[111],\quad \alpha^6=[110].$$

Zbiór wektorów jest uzupełniany wektorem zerowym.

Gdy ciągiem początkowym sekwencji pseudolosowej będzie ciąg 100, wówczas współrzędne wektorów odpowiadają pierwszym kolumnom macierzy tych samych elementów ciała. Posługując się elementami ciała w postaci wektorowej, można wyznaczyć sumę dowolnych elementów ciała oraz całą tabliczkę dodawania. Na przykład suma dwóch elementów ciała α^2 i α^3 wynosi

$$\alpha^2 + \alpha^3 = [010] + [101] = [111] = \alpha^5.$$

Tablica 1.5.2. Tabliczki mnożenia i dodawania ciała $GF(8)$

·	0	1	α	α^2	α^3	α^4	α^5	α^6
0	0	0	0	0	0	0	0	0
1	0	1	α	α^2	α^3	α^4	α^5	α^6
α	0	α	α^2	α^3	α^4	α^5	α^6	1
α^2	0	α^2	α^3	α^4	α^5	α^6	1	α
α^3	0	α^3	α^4	α^5	α^6	1	α	α^2
α^4	0	α^4	α^5	α^6	1	α	α^2	α^3
α^5	0	α^5	α^6	1	α	α^2	α^3	α^4
α^6	0	α^6	1	α	α^2	α^3	α^4	α^5

+	0	1	α	α^2	α^3	α^4	α^5	α^6
0	0	1	α	α^2	α^3	α^4	α^5	α^6
1	1	0	α^3	α^6	α	α^5	α^4	α^2
α	α	α^3	0	α^4	1	α^2	α^6	α^5
α^2	α^2	α^6	α^4	0	α^5	α	α^3	1
α^3	α^3	α	1	α^5	0	α^6	α^2	α^4
α^4	α^4	α^5	α^2	α	α^6	0	1	α^3
α^5	α^5	α^4	α^6	α^3	α^2	1	0	α
α^6	α^6	α^2	α^5	1	α^4	α^3	α	0

1.5.5. Elementy ciała skończonego w postaci wielomianów

Elementy ciała rozszerzonego w postaci wielomianów są używane do badania struktury ciała i mogą również służyć do obliczenia tabliczki dodawania. Aby obliczyć elementy ciała rozszerzonego $GF(8)$ w formie wielomianów, wykorzystujemy kongruencję (1.5.4)

$$x^i \equiv R_i (\text{mod } \text{p}(x)),$$

gdzie R_i oznacza resztę z dzielenia x^i przez wielomian $p(x)$. Reszta ta jest elementem ciała rozszerzonego.

Dla przykładu obliczmy element α^4. Aby rozwiązać kongruencję

$$x^4 \equiv R_4 \left(\text{mod } x^3 + x + 1\right),$$

dzielimy x^4 przez $x^4 + x + 1$ i bierzemy resztę z tego dzielenia. Dzielenie wielomianów ilustruje poniższy przykład:

$$\begin{array}{rl} & x \\ \hline x^3 + x + 1 \;\big|\; x^4 & \\ & x^4 + x^2 + x \\ \hline & x^2 + x \end{array}$$

Element α^4, jak i następne elementy, można obliczyć również z iloczynu

$$\alpha^4 = \alpha^3 \cdot \alpha = (1+\alpha)\alpha = \alpha^2 + \alpha.$$

Wyniki obliczeń przedstawiono w tabeli 1.5.3. Pierwsza kolumna zawiera kongruencje, z których wyznaczono poszczególne elementy ciała w postaci wielomianów. W drugiej kolumnie zapisano elementy ciała w postaci potęg elementu pierwotnego ciała. Trzecia kolumna pokazuje elementy ciała w formie wektorów utworzonych z wielomianów. Współrzędne tych wektorów odpowiadają pierwszym wierszom macierzy wyrażających te same elementy ciała.

Tabela 1.5.3. Obliczanie elementów ciała $GF(8)$ w postaci wielomianów

Kongruencje	Potęgi elementu pierwotnego	Wektory $[\alpha^0\ \alpha^1\ \alpha^2]$
$x^0 \equiv 1 \left(\mathrm{mod}\ x^3 + x + 1\right)$	$\alpha^0 = 1$	$1 = [1\,0\,0]$
$x^1 \equiv x \left(\mathrm{mod}\ x^3 + x + 1\right)$	$\alpha^1 = \alpha$	$\alpha = [0\,1\,0]$
$x^2 \equiv x^2 \left(\mathrm{mod}\ x^3 + x + 1\right)$	$\alpha^2 = \alpha^2$	$\alpha^2 = [0\,0\,1]$
$x^3 \equiv x + 1 \left(\mathrm{mod}\ x^3 + x + 1\right)$	$\alpha^3 = 1 + \alpha$	$\alpha^3 = [1\,1\,0]$
$x^4 \equiv x^2 + x \left(\mathrm{mod}\ x^3 + x + 1\right)$	$\alpha^4 = \alpha + \alpha^2$	$\alpha^4 = [0\,1\,1]$
$x^5 \equiv x^2 + x + 1 \left(\mathrm{mod}\ x^3 + x + 1\right)$	$\alpha^5 = 1 + \alpha + \alpha^2$	$\alpha^5 = [1\,1\,1]$
$x^6 \equiv x^2 + 1 \left(\mathrm{mod}\ x^3 + x + 1\right)$	$\alpha^6 = 1 + \alpha^2$	$\alpha^6 = [1\,0\,1]$

Po wyznaczeniu elementów ciała rozszerzonego można przystąpić, podobnie jak w poprzednich przypadkach, do obliczenia tabliczki dodawania. I tak np. sumę dwóch elementów ciała obliczamy następująco

$$\alpha^2 + \alpha^4 = \alpha^2 + \alpha + \alpha^2 = \alpha.$$

Tabliczkę dodawania ciała *GF*(8) pokazano w tabeli 1.5.2.

1.5.6. Ciała rozszerzone nad niebinarnymi ciałami prostymi

Nad każdym ciałem prostym $GF(p)$ można skonstruować ciała rozszerzone $GF(p^m)$. Charakterystyka ciała rozszerzonego $GF(p^m)$ jest równa charakterystyce ciała podstawowego, czyli p. Liczba p w takich ciałach jest równa zero, ponieważ $p(\mathrm{mod}\ p) = 0$.

Gdy posługujemy się ciałami skończonymi, niekiedy musimy wyznaczyć element przeciwny. Dla ciał o charakterystyce dwa z (1.2.5) mamy: $\alpha^i + \alpha^i = 0$, skąd $-\alpha^i = \alpha^i$. Dla ciał o charakterystyce $p>2$ wzór na element przeciwny można wyznaczyć z zależności

$$\alpha^{q-1} - 1 = \left(\alpha^{(q-1)/2} - 1\right)\left(\alpha^{(q-1)/2} + 1\right).$$

Lewa strona równania wynosi zero, a ponieważ $\alpha^{(q-1)/2} \neq 1$, więc musi być spełniony warunek, że $\alpha^{(q-1)/2} + 1 = 0$ i $-1 = \alpha^{(q-1)/2}$. Zatem dla ciał o charakterystyce $p>2$ element przeciwny wynosi

$$-\alpha^i = (-1)\alpha^i = \alpha^{i+(q-1)/2}. \tag{1.5.6}$$

Ciała rozszerzone nad ciałami prostymi niebinarnymi konstruuje się tak samo jak nad ciałem binarnym. Rozważmy dla przykładu konstrukcję ciała rozszerzonego $GF(3^2)$ nad ciałem trzyelementowym $GF(3)$. Weźmy w tego celu wielomian pierwotny drugiego stopnia nad $GF(3)$

$$x^2 + x + 2.$$

Elementy ciała rozszerzonego $GF(9)$ zapiszmy w postaci wektorowej. Zależność rekurencyjna stowarzyszona z powyższym wielomianem będzie

$$s_{j+2} = s_j + 2s_{j+1}.$$

Z zależności tej otrzymamy następującą sekwencję pseudolosową

1 0 1 2 2 0 2 1 1 0 ...

Tablica 1.5.4. Tabliczki mnożenia i dodawania ciała $GF(9)$

·	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7
0	0	0	0	0	0	0	0	0	0
1	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7
α	0	α	α^2	α^3	α^4	α^5	α^6	α^7	1
α^2	0	α^2	α^3	α^4	α^5	α^6	α^7	1	α
α^3	0	α^3	α^4	α^5	α^6	α^7	1	α	α^2
α^4	0	α^4	α^5	α^6	α^7	1	α	α^2	α^3
α^5	0	α^5	α^6	α^7	1	α	α^2	α^3	α^4
α^6	0	α^6	α^7	1	α	α^2	α^3	α^4	α^5
α^7	0	α^7	1	α	α^2	α^3	α^4	α^5	α^6

+	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7
0	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7
1	1	α^4	α^7	α^3	α^5	0	α^2	α	α^6
α	α	α^7	α^5	1	α^4	α^6	0	α^3	α^2
α^2	α^2	α^3	1	α^6	α	α^5	α^7	0	α^4
α^3	α^3	α^5	α^4	α	α^7	α^2	α^6	1	0
α^4	α^4	0	α^6	α^5	α^2	1	α^3	α^7	α
α^5	α^5	α^2	0	α^7	α^6	α^3	α	α^4	1
α^6	α^6	α	α^3	0	1	α^7	α^4	α^2	α^5
α^7	α^7	α^6	α^2	α^4	0	α	1	α^5	α^3

Niezerowe elementy ciała rozszerzonego $GF(3)$ wyznaczamy, biorąc po dwa kolejne elementy sekwencji pseudolosowej:

$$0=[00],\quad 1=[10],\quad \alpha=[01],$$
$$\alpha^2=[12],\quad \alpha^3=[22],\quad \alpha^4=[20],$$
$$\alpha^5=[02],\quad \alpha^6=[21],\quad \alpha^7=[11].$$

Posługując się tymi wyrażeniami, można obliczyć tabliczkę dodawania ciała $GF(9)$. Współrzędne elementów ciała $GF(9)$ dodajemy zgodnie z zasadami rachowania w ciele $GF(3)$, stosując wzór (1.2.1). Tabliczkę mnożenia obliczamy, korzystając ze wzoru (1.2.2). Tabliczki działań skonstruowanego ciała $GF(9)$ pokazano w tabeli 1.5.4.

1.5.7. Wielomiany minimalne elementów ciała

Dla każdego ciała rozszerzonego $GF(p^m)$ jego elementy można zapisać w postaci potęg elementu pierwotnego α : $0,1,\alpha, a^2,\ldots,\alpha^{p^m-2}$. Elementy pierwotne ciała rozszerzonego mają rząd multyplikatywny równy p^m-1. Rząd multyplikatywny elementów niepierwotnych ciała rozszerzonego jest dzielnikiem p^m-1.

Dla każdego elementu ciała można wyznaczyć wielomian minimalny. Wielomianem minimalnym $m_i(x)$ elementu ciała α^i jest wielomian najniższego stopnia taki, że α^i jest pierwiastkiem tego wielomianu

$$m_i\left(\alpha^i\right)=0. \tag{1.5.7}$$

Wielomian minimalny elementu ciała można znaleźć obliczając kolejno:
- stopień wielomianu minimalnego,
- współczynniki wielomianu.

Stopień wielomianu *k* jest najmniejszą liczbą całkowitą taką, że

$$i\cdot p^k \equiv i\cdot(\mathrm{mod}\ q-1). \tag{1.5.8}$$

Współczynniki wielomianu można obliczyć po skonstruowaniu ciała.

Rozważmy ciało mające szesnaście elementów $GF(2^4)$, utworzone za pomocą wielomianu pierwotnego x^4+x+1 nad $GF(2)$. Elementy tego ciała w postaci wektorowej będą następującej postaci:

$$0=[0000],\quad 1=[1000],\quad \alpha=[0001],\quad \alpha^2=[0010],$$
$$\alpha^3=[0100],\quad \alpha^4=[1001],\quad \alpha^5=[0011],\quad \alpha^6=[0110],$$
$$\alpha^7=[1101],\quad \alpha^8=[1010],\quad \alpha^9=[0101],\quad \alpha^{10}=[1011],$$
$$\alpha^{11}=[0111],\quad \alpha^{12}=[1111],\quad \alpha^{13}=[1110],\quad \alpha^{14}=[1100].$$

Wyznaczmy wielomian minimalny dla elementu α^7. Najpierw wyznaczamy stopień wielomianu minimalnego. W tym celu podstawiamy do kongruencji (1.5.8): $i = 7,\ p = 2$ i $q = 16$

$$7 \cdot 2^k \equiv 7 \pmod{15}.$$

Zależność ta jest spełniona dla najmniejszej wartości $k = 4$. Wielomian minimalny elementu ciała α^7 jest zatem czwartego stopnia i można go zapisać w postaci

$$m_7(x) = x^4 + a x^3 + b x^2 + c x + d.$$

Następnie obliczamy współczynniki tego wielomianu. Za zmienną podstawiamy element ciała α^7

$$\left(\alpha^7\right)^4 + a\left(\alpha^7\right)^3 + b\left(\alpha^7\right)^2 + c\left(\alpha^7\right) + d = 0,$$
$$\alpha^{13} + a\alpha^6 + b\alpha^{14} + c\alpha^7 + d = 0.$$

Do ostatniego wzoru podstawiamy wektory odpowiadające elementom ciała i mnożymy współczynniki wielomianu przez te wektory

$$[1110] + a[0110] + b[1100] + c[1101] + d[1000] = [0000],$$
$$[0\,a\,a\,0] + [b\,b\,0\,0] + [c\,c\,0\,c] + [d\,0\,0\,0] = [1110].$$

Dla poszczególnych współrzędnych wektorów można napisać układ czterech równań:

$$b + c + d = 1, \quad a + b + c = 1, \quad a = 1, \quad c = 0.$$

Po rozwiązaniu tego układu równań otrzymamy współczynniki wielomianu minimalnego:

$$a = 1, \quad b = 0, \quad c = 0, \quad d = 1.$$

Wielomian minimalny elementu ciała α^7 będzie

$$m_7(x) = x^4 + x^3 + 1.$$

Wielomian odwrotny do powyższego będzie wielomianem minimalnym elementu ciała $\alpha^{q^m - 1 - i} = \alpha^8$. Wielomian ten ma postać

$$m_8(x) = x^4 + x + 1.$$

Obliczony wielomian minimalny $m_7(x)$ jest wielomianem czwartego stopnia i ma cztery pierwiastki: $\alpha^7, \alpha^{14}, \alpha^{13}$ i α^{11}. Pierwiastki te tworzą szereg cykliczny zwany warstwą cyklotomiczną. Dla ciał o charakterystyce dwa każdy następny pierwiastek tej warstwy jest kwadratem pierwiastka poprzedniego. Kwadrat ostatniego pierwiastka daje pierwiastek pierwszy, np. $(\alpha^{11})^2 = \alpha^{22 \pmod{15}} = \alpha^7$.

Generalnie wielomian minimalny $m_i(x)$ stopnia k nad $GF(q)$ ma następujące pierwiastki:

$$\alpha^{i}, \alpha^{ip}, \alpha^{ip^2}, \ldots, \alpha^{ip^{k-1} \pmod{q-1}}. \tag{1.5.9}$$

Elementy tego ciągu są nazywane elementami sprzężonymi i mają taki sam rząd multyplikatywny. Posługując się powyższym ciągiem, można rozbić na warstwy cyklotomiczne niezerowe pierwiastki każdego ciała rozszerzonego. Aby utworzyć te warstwy, bierzemy kolejne elementy ciała, które nie występują w warstwach poprzednich. Pierwszą warstwę tworzy element 1. Dla rozważanego ciała $GF(2^4)$ otrzymamy następujące warstwy cyklotomiczne:

$$\begin{gathered} 1; \\ \alpha, \alpha^2, \alpha^4, \alpha^8; \\ \alpha^3, \alpha^6, \alpha^{12}, \alpha^9; \\ \alpha^5, \alpha^{10}; \\ \alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}. \end{gathered}$$

Każda z tych warstw odpowiada jednemu wielomianowi minimalnemu, którego pierwiastkami będą elementy warstwy. W powyższym przykładzie warstwa druga, trzecia i piąta utworzą wielomiany czwartego stopnia, warstwa czwarta wielomian drugiego stopnia, a jedynka, znajdująca się w warstwie pierwszej, utworzy wielomian pierwszego stopnia.

Znajomość rozkładu pierwiastków na warstwy cyklotomiczne pozwala obliczyć wielomiany minimalne. Jeśli znamy pierwiastki wielomianu $x_1, x_2, \ldots, x_n$, to wielomian można obliczyć ze znanego w algebrze wzoru

$$p(x) = (x - x_1)(x - x_2) \ldots (x - x_n). \tag{1.5.10}$$

Obliczmy dla przykładu wielomian minimalny elementu α^3 ciała $GF(2^4)$

$$m_3(x) = (x + \alpha^3)(x + \alpha^6)(x + \alpha^{12})(x + \alpha^9).$$

Obliczenie powyższego iloczynu wymaga znajomości elementów ciała w postaci addytywnej lub tabliczki dodawania. Korzystając z wyżej obliczonych elementów ciała w postaci wektorowej, po wykonaniu mnożenia otrzymamy

$$m_3(x) = x^4 + x^3 + x^2 + x + 1.$$

Wielomiany minimalne elementów ciała $GF(2^4)$ i odpowiadające im warstwy cyklotomiczne pokazano w tabeli 1.5.5. W ostatniej kolumnie tej tabeli podano rzędy multyplikatywne elementów warstw *e*.

Tabela 1.5.5. Wielomiany minimalne elementów ciała $GF(2^4)$

$m_i(x)$	Pierwiastki $m_i(x)$	e
$m_0(x) = x+1$	1	1
$m_1(x) = x^4 + x + 1$	$\alpha, \alpha^2, \alpha^4, \alpha^8$	15
$m_3(x) = x^4 + x^3 + x^2 + x + 1$	$\alpha, \alpha^3, \alpha^6, \alpha^{12}, \alpha^9$	5
$m_5(x) = x^2 + x + 1$	α^5, α^{10}	3
$m_7(x) = x^4 + x^3 + 1$	$\alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}$	15

Wielomiany, których pierwiastki mają rząd multyplikatywny $p^m - 1$, są wielomianami pierwotnymi. Są to wielomiany: $x+1$, $x^4 + x + 1$, $x^2 + x + 1$ i $x^4 + x^3 + 1$.

1.5.8. Struktura ciał rozszerzonych

W teorii ciał skończonych znane jest twierdzenie, zgodnie z którym zbiór wszystkich pierwiastków dwumianu $x^{q-1} - 1$ stanowi zbiór wszystkich niezerowych elementów ciała $GF(q)$. Możemy zatem napisać

$$x^{q-1} - 1 = (x-1)(x-\alpha)(x-\alpha^2)\ldots(x-\alpha^{q-1}). \tag{1.5.11}$$

Ponieważ wielomiany minimalne elementów ciała skończonego można obliczyć ze wzoru (1.5.10), zatem iloczyn (1.5.11) jest równy iloczynowi wszystkich wielomianów minimalnych elementów ciała. W przypadku ciała $GF(2^4)$ będzie to

$$x^{15} + 1 = (x+1)(x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)(x^4 + x^3 + 1).$$

Iloczyn ten zawiera wszystkie nierozkładalne wielomiany stopnia czwartego. Słuszne jest również ogólne twierdzenie, że każdy nierozkładalny wielomian stopnia m nad ciałem $GF(q^m)$ jest dzielnikiem dwumianu $x^{q^m-1} - 1$.

Rozkład dwumianu $x^{q^m-1} - 1$ nad ciałem $GF(q)$ na iloczyn wielomianów nierozkładalnych, zwany też faktoryzacją, umożliwia poznanie struktury ciała. Podczas faktoryzacji dwumianu typu $x^{q^m-1} - 1$ obowiązują następujące twierdzenia:

- Każdy wielomian nierozkładalny nad ciałem $GF(q)$, dzielący dwumian $x^{q^m-1} - 1$, ma stopień równy m lub stopień będący podzielnikiem liczby m. Wielomiany te rozkładają całkowicie dwumian $x^{q^m-1} - 1$ i nie ma on innych podzielników. Na przykład wielomiany nad ciałem $GF(2^4)$ mają stopnie: 1, 2 i 4.
- Dla każdego naturalnego m istnieje przynajmniej jeden wielomian pierwotny stopnia m. Liczbę wielomianów pierwotnych stopnia m można obliczyć z (1.3.3).

Zbiór wielomianów nierozkładalnych, które są dzielnikami dwumianu $x^{q^m-1}-1$, można rozbić na grupy, uwzględniając rząd multyplikatywny pierwiastków tych wielomianów. Na przykład wielomiany będące dzielnikami dwumianu $x^{15}-1$, podane w tabeli 1.5.5, można rozbić na cztery grupy

$$x^{15}-1=\phi_1(x)\phi_3(x)\phi_5(x)\phi_{15}(x),$$

gdzie $\phi_1(x)=m_0(x),\ \phi_3(x)=m_5(x),\ \phi_5(x)=m_3(x),\ \phi_{15}(x)=m_1(x)\cdot m_7(x).$

W powyższym zbiorze wielomianów $\phi_1(x)$, $\phi_3(x)$ i $\phi_5(x)$ są wielomianami samoodwrotnymi, a wielomian $\phi_{15}(x)$ jest iloczynem wielomianów pierwotnych, odwrotnych względem siebie. Wielomiany te generują ciała izomorficzne.

Tabela 1.5.6. Wielomiany minimalne ciał rozszerzonych nad $GF(2)$

m	$p(x)$	i	M	m	$p(x)$	i	M	m	$p(x)$	i	M
1	11	0	1	6	1001001	7	9	8	111110011	5	51
2	11	0	1		1101	9	7		101101001	7	255
	111	1	3		1101101	11	63		110111101	9	85
3	11	0	1		111	21	3		111100111	11	255
	1011	1	7	7	11	0	1		100101011	13	255
4	11	0	1		10001001	1	127		111010111	15	17
	10011	1	15		10001111	3	127		10011	17	15
	111	3	3		10011101	5	127		101100101	19	255
	11111	5	5		11110111	7	127		110001011	21	85
5	11	0	1		10111111	9	127		101100011	23	255
	100101	1	31		11010101	11	127		100011011	25	51
	111101	3	31		10000011	13	127		100111111	27	85
	110111	5	31		11001011	19	127		101011111	37	255
6	11	0	1		11100101	21	127		111000011	43	255
	1000011	1	63	8	11	0	1		100111001	45	17
	1010111	3	21		100011101	1	255		11111	51	5
	1100111	5	63		101110111	3	85		111	85	3

Nad każdym ciałem $GF(q)$ istnieje $\varphi(q-1)/m$ wielomianów pierwotnych stopnia m. Za pomocą każdego z tych wielomianów można utworzyć ciało zawierające q^m elementów. Ciała takie są ciałami izomorficznymi. Izomorfizm systemów algebraicznych jest wzajemnie jednoznacznym odwzorowaniem jednego systemu na inny, podobny system z zachowaniem działań. Aby posługiwać się ciałami izomorficznymi, wystarczy skonstruować jedno ciało i we wszystkich działaniach uwzględnić odwzorowanie:

$$\omega: \quad \alpha \to \alpha^{i}, \tag{1.5.12}$$

gdzie α^{i} jest dowolnym elementem pierwotnym ciała oraz $(i, q-1) = 1$.

Odwzorowanie izomorficzne nie zmienia struktury grupy multyplikatywnej, a tym samym tabliczki mnożenia. Zauważmy, że odwzorowanie

$$\omega: \quad \alpha \to \alpha^{p^i}, \quad i = 0, 1, \ldots, m-1$$

jest automorfizmem, ponieważ jego elementy są pierwiastkami tego samego wielomianu.

Dla ciał rozszerzonych niewielkich stopni wielomiany minimalne można znaleźć w tablicach publikowanych w książkach z zakresu teorii kodów korekcyjnych [1.2, 2.4]. W tabeli 1.5.6 podano wielomiany minimalne ciał rozszerzonych $GF(2^m)$ dla $m \le 8$. Wielomiany minimalne $p(x)$ podano w formie współczynników, nie uwzględniając wielomianów odwrotnych. Parametr i jest indeksem wielomianu minimalnego $m_i(x)$, a parametr M – okresem sekwencji generowanej przez wielomian.

1.6. Sekwencje okresowe i rozszerzenia ciał nad ciałami rozszerzonymi

1.6.1. Wielomiany i sekwencje okresowe nad ciałami rozszerzonymi

Ciała rozszerzone można konstruować nad każdym ciałem skończonym. Jedno z twierdzeń algebry ciał skończonych mówi, że nad każdym ciałem skończonym istnieje przynajmniej jeden wielomian pierwotny stopnia m, gdzie m jest liczbą całkowitą dodatnią. Istnieją więc warunki do konstrukcji każdego ciała rozszerzonego o dowolnym stopniu rozszerzenia.

Aby omówić metodę konstrukcji ciał rozszerzonych nad ciałami rozszerzonymi niższego stopnia, należy w pierwszej kolejności zapoznać się z wielomianami nad ciałami rozszerzonymi. Wielomiany nad ciałami rozszerzonymi zachowują się tak samo jak wielomiany nad ciałami prostymi, dlatego też wszystkie działania i właściwości podane w p. 1.3 mogą być tu stosowane.

Rozważmy faktoryzację dwumianu $x^{q^2-1}-1$ nad $GF(4)$; $q = 4$, $m = 2$. Dwumian ten można rozłożyć w następujący sposób

$$x^{q^2-1}-1 = x^{15}+1 = (x+1)(x+\alpha)(x+\alpha^2)(x^2+x+\alpha^2)(x^2+\alpha x+\alpha)\cdot$$
$$\cdot(x^2+x+\alpha)(x^2+\alpha^2 x+\alpha^2)(x^2+\alpha^2 x+1)(x^2+\alpha x+1).$$

Wśród otrzymanych wielomianów nierozkładalnych istnieją wielomiany pierwotne pierwszego i drugiego stopnia oraz wielomiany niepierwotne stopnia drugiego. Klasyfikację tych wielomianów podano w tabeli 1.6.1. W tabeli tej pokazano wielomiany nierozkładalne, ich pierwiastki i rzędy multyplikatywne pierwiastków e. Pierwiastki wielomianów są elementami ciała $GF(4^2)$. Tabliczki działań i elementy ciała $GF(4^2)$ podano w p. 1.6.3.

Tabela 1.6.1. Wielomiany nierozkładalne drugiego stopnia nad $GF(4)$

Typ wielomianu	Wielomian	Pierwiastki	e	Wielomian odwrotny	Pierwiastki	e
pierwotny	$x+1$	1	1	samoodwrotny		
pierwotny	$x+\alpha^2$	ω^5	3	$x+\alpha$	ω^{10}	3
pierwotny	$x^2+x+\alpha^2$	ω,ω^4	15	$x^2+\alpha x+\alpha$	ω^{11},ω^{14}	15
pierwotny	$x^2+x+\alpha$	ω^2,ω^8	15	$x^2+\alpha^2 x+\alpha^2$	ω^7,ω^{13}	15
niepierwotny	$x^2+\alpha x+1$	ω^3,ω^{12}	5	samoodwrotny		
niepierwotny	$x^2+\alpha^2 x+1$	ω^6,ω^9	5	samoodwrotny		

Na wielomianach nad ciałami rozszerzonymi można wykonywać działania algebraiczne tak samo jak na wielomianach nad ciałami prostymi. Przykład dzielenia wielomianów $(x^3 + x^2 + \alpha)/(x + \alpha)$ nad ciałem $GF(4)$ przedstawiono niżej. Z prawej strony pokazano dzielenie tych samych wielomianów zapisanych w formie współczynników. Tabliczki działań dla ciała $GF(4)$ podano w tabeli 1.5.1.

$$
\begin{array}{rl|l}
 & x^2 + \alpha^2 x + 1 & \\
x + \alpha & x^3 + x^2 + 0 + \alpha & \\
 & x^3 + \alpha x^2 & \\
\hline
 & \alpha^2 x^2 + 0 & \\
 & \alpha^2 x^2 + x & \\
\hline
 & x + \alpha & \\
 & x + \alpha & \\
\hline
 & 0 &
\end{array}
\qquad\qquad
\begin{array}{rl|l}
 & 1 \;\; \alpha^2 \;\; 1 & \\
1 \;\; \alpha & 1 \;\; 1 \;\; 0 \;\; \alpha & \\
 & 1 \;\; \alpha & \\
\hline
 & \alpha^2 \;\; 0 & \\
 & \alpha^2 \;\; 1 & \\
\hline
 & 1 \;\; \alpha & \\
 & 1 \;\; \alpha & \\
\hline
 & 0 &
\end{array}
$$

1.6.2. Właściwości sekwencji pseudolosowych nad ciałami rozszerzonymi

Zasady generowania sekwencji okresowych nad ciałami prostymi i rozszerzonymi są takie same. Generowanie sekwencji okresowych nad ciałami prostymi przedstawiono w p. 1.4. Podane tam wzory i układ generatora mogą być również zastosowane do opisu i generowania sekwencji nad ciałami rozszerzonymi. Trzeba tylko pamiętać, że elementy układu generatora pokazanego na rys. 1.4.2 w przypadku ciał rozszerzonych są elementami wielostanowymi, a generowane sekwencje sekwencjami niebinarnymi.

Wielomiany pierwotne stopnia m nad $GF(q)$ generują sekwencje o maksymalnym okresie, zwane sekwencjami pseudolosowymi. Okres sekwencji pseudolosowej określa wzór (1.4.2).

Niech wielomianem generującym sekwencję pseudolosową będzie wielomian pierwotny nad $GF(4)$

$$x^2 + x + \alpha.$$

Zależność rekurencyjna stowarzyszona z tym wielomianem będzie

$$s_{j+2} = \alpha\, s_j + s_{j+1}, \quad j = 0,1,2,\ldots$$

Działania w tym wzorze są zgodne z zasadami obowiązującymi w ciele $GF(4)$. Tabliczki działań dla ciała $GF(4)$ podano w tabeli 1.5.1. Zakładając dwa elementy początkowe s_0 i s_1, możemy z powyższego wzoru obliczyć sekwencję pseudolosową

$$011\alpha^2 10\alpha\alpha\ 1\alpha\ 0\alpha^2\alpha^2\alpha\ \alpha^2 0\ 1\ldots \tag{1.6.1}$$

Okres sekwencji pseudolosowej jest określony wzorem (1.4.2). Okres powyższej se-

kwencji wynosi 15.

Zakładamy, że wielomian pierwotny $p(x)$ stopnia m nad $GF(q)$ generuje zbiór $\delta(m)$ sekwencji okresowych, zawierający $2^m - 1$ sekwencji o okresie $2^m - 1$ oraz sekwencję zerową.

Właściwości pseudolosowych sekwencji binarnych omówiono w p.1.4.3. Niektóre właściwości sekwencji nad ciałami rozszerzonymi pokrywają się z właściwościami sekwencji binarnych. Są to właściwości o następujących numerach podanych w p. 1.4.3:

1. Przesunięcie cykliczne.
2. Rekurencja.
3. Właściwość okna.
5. Właściwość addytywna.
6. Suma sekwencji i jej przesunięcia cyklicznego.

Różnice występują we właściwościach 4 i 7.

4. Rozkład znaków

W każdej sekwencji pseudolosowej nad ciałem $GF(q)$ zero pojawia się $q^{m-1} - 1$ razy, a każdy inny element niezerowy pojawia się q^{m-1} razy.

Dla sekwencji (1.6.1) $q^{m-1} = 4$. W sekwencji tej zero pojawia się trzy razy, a pozostała elementy po cztery razy.

7. Struktura sekwencji pseudolosowej i korelacja między elementami

Każda sekwencja pseudolosowa nad ciałem rozszerzonym $GF(q)$ ma formę:

$$b,\alpha b,\alpha^2 b,\ldots,\alpha^{q-2} b, \tag{1.6.2}$$

gdzie b jest sekwencją o długości $(q^m - 1)/(q - 1)$, a α – elementem pierwotnym ciała.

Na przykład sekwencję pseudolosową (1.6.1) nad ciałem $GF(4)$ można podzielić na trzy podokresy:

$$\underbrace{11\alpha^2 10}_{b}\underbrace{\alpha\alpha 1\alpha 0}_{b\alpha}\underbrace{\alpha^2\alpha^2\alpha\alpha^2 0}_{b\alpha^2}.$$

W celu przeanalizowania korelacji między elementami sekwencji niebinarnych rozważmy sekwencję pseudolosową (1.6.1), dla której $q = 4$, a $m = 2$, oraz jej przesunięcie cykliczne w lewo:

$$\begin{array}{cccccccccccccccc} 0 & 1 & 1 & \alpha^2 & 1 & 0 & \alpha & \alpha & 1 & \alpha & 0 & \alpha^2 & \alpha^2 & \alpha & \alpha^2, \\ 1 & 1 & \alpha^2 & 1 & 0 & \alpha & \alpha & 1 & \alpha & 0 & \alpha^2 & \alpha^2 & \alpha & \alpha^2 & 0. \end{array}$$

Korelację między elementami sekwencji określa się na podstawie częstości wystę-

powania par elementów, należących do sekwencji i jej przesunięcia cyklicznego. O tej częstości decyduje to, czy liczba przesunięć cyklicznych sekwencji jest wielokrotnością podokresu wyrażonego wzorem $(q^m - 1)/(q - 1)$. Korelację między elementami sekwencji i elementami sekwencji przesuniętej cyklicznie charakteryzuje tabela 1.6.2.

Przesunięcie jednokrotne nie jest wielokrotnością wyrażenia $(q^m - 1)/(q - 1)$, więc w przypadku tej sekwencji mamy:

- $q^{m-2} - 1 = 0$, co oznacza, że para (0, 0) nie wystąpi ani razu,
- $q^{m-2} = 1$, a więc każda inna para elementów zawierająca różne elementy (α^i, α^j) pojawia się raz, co widać na powyższym przykładzie.

Tabela 1.6.2. Korelacja elementów sekwencji pseudolosowej

s	(0, 0)	(α^i, α^i)	(α^i, α^j)
$s = 0$	$q^{m-1} - 1$	q^{m-1}	0
s jest wielokrotnością $\frac{q^m - 1}{q - 1}$	$q^{m-1} - 1$	0	q^{m-1}
s nie jest wielokrotnością $\frac{q^m - 1}{q - 1}$	$q^{m-2} - 1$	0	q^{m-2}

W tabeli 1.6.2 w pierwszej kolumnie podano liczbę przesunięć cyklicznych s. W następnych kolumnach mamy częstość występowania par elementów, należących do sekwencji i jej przesunięcia cyklicznego. W kolumnie drugiej są częstotliwości występowania par elementów zerowych, w kolumnie trzeciej – par jednakowych elementów, a w kolumnie czwartej – par różnych elementów.

1.6.3. Rozszerzenia ciał nad ciałami rozszerzonymi

Nad ciałami rozszerzonymi można, podobnie jak nad ciałami prostymi, konstruować ciała rozszerzone wyższego rzędu. Do konstrukcji takich ciał stosuje się metody opisane w p. 1.5. Niech przykładem konstrukcji ciała rozszerzonego wyższego rzędu będzie rozszerzenie drugiego stopnia nad ciałem $GF(4)$. Ciało rozszerzone skonstruujemy, zapisując elementy ciała za pomocą wektorów.

Do konstrukcji ciała rozszerzonego przyjmijmy z tabeli 1.6.1 jeden z wielomianów pierwotnych stopnia drugiego nad $GF(4)$

$$x^2 + x + \alpha^2.$$

Zależność rekurencyjna stowarzyszona z tym wielomianem będzie

$$s_{j+2} = \alpha^2 s_j + s_{j+1}, \quad j = 0, 1, 2, \ldots$$

Tabela 1.6.3. Tabliczki mnożenia i dodawania ciała $GF(4^2)$

·	0	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}
ω	0	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1
ω^2	0	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω
ω^3	0	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2
ω^4	0	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3
ω^5	0	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4
ω^6	0	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5
ω^7	0	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6
ω^8	0	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7
ω^9	0	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8
ω^{10}	0	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9
ω^{11}	0	ω^{11}	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}
ω^{12}	0	ω^{12}	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}
ω^{13}	0	ω^{13}	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}
ω^{14}	0	ω^{14}	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}

+	0	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}
0	0	1	ω	ω^2	ω^3	ω^4	ω^5	ω^6	ω^7	ω^8	ω^9	ω^{10}	ω^{11}	ω^{12}	ω^{13}	ω^{14}
1	1	0	ω^4	ω^8	ω^{14}	ω	ω^{10}	ω^{13}	ω^9	ω^2	ω^7	ω^5	ω^{12}	ω^{11}	ω^6	ω^3
ω	ω	ω^4	0	ω^5	ω^9	1	ω^2	ω^{11}	ω^{14}	ω^{10}	ω^3	ω^8	ω^6	ω^{13}	ω^{12}	ω^7
ω^2	ω^2	ω^8	ω^5	0	ω^6	ω^{10}	ω	ω^3	ω^{12}	1	ω^{11}	ω^4	ω^9	ω^7	ω^{14}	ω^{13}
ω^3	ω^3	ω^{14}	ω^9	ω^6	0	ω^7	ω^{11}	ω^2	ω^4	ω^{13}	ω	ω^{12}	ω^5	ω^{10}	ω^8	1
ω^4	ω^4	ω	1	ω^{10}	ω^7	0	ω^8	ω^{12}	ω^3	ω^5	ω^{14}	ω^2	ω^{13}	ω^6	ω^{11}	ω^9
ω^5	ω^5	ω^{10}	ω^2	ω	ω^{11}	ω^8	0	ω^9	ω^{13}	ω^4	ω^6	1	ω^3	ω^{14}	ω^7	ω^{12}
ω^6	ω^6	ω^{13}	ω^{11}	ω^3	ω^2	ω^{12}	ω^9	0	ω^{10}	ω^{14}	ω^5	ω^7	ω	ω^4	1	ω^8
ω^7	ω^7	ω^9	ω^{14}	ω^{12}	ω^4	ω^3	ω^{13}	ω^{10}	0	ω^{11}	1	ω^6	ω^8	ω^2	ω^5	ω
ω^8	ω^8	ω^2	ω^{10}	1	ω^{13}	ω^5	ω^4	ω^{14}	ω^{11}	0	ω^{12}	ω	ω^7	ω^9	ω^3	ω^6
ω^9	ω^9	ω^7	ω^3	ω^{11}	ω	ω^{14}	ω^6	ω^5	1	ω^{12}	0	ω^{13}	ω^2	ω^8	ω^{10}	ω^4
ω^{10}	ω^{10}	ω^5	ω^8	ω^4	ω^{12}	ω^2	1	ω^7	ω^6	ω	ω^{13}	0	ω^{14}	ω^3	ω^9	ω^{11}
ω^{11}	ω^{11}	ω^{12}	ω^6	ω^9	ω^5	ω^{13}	ω^3	ω	ω^8	ω^7	ω^2	ω^{14}	0	1	ω^4	ω^{10}
ω^{12}	ω^{12}	ω^{11}	ω^{13}	ω^7	ω^{10}	ω^6	ω^{14}	ω^4	ω^2	ω^9	ω^8	ω^3	1	0	ω	ω^5
ω^{13}	ω^{13}	ω^6	ω^{12}	ω^{14}	ω^8	ω^{11}	ω^7	1	ω^5	ω^3	ω^{10}	ω^9	ω^4	ω	0	ω^2
ω^{14}	ω^{14}	ω^3	ω^7	ω^{13}	1	ω^9	ω^{12}	ω^8	ω	ω^6	ω^4	ω^{11}	ω^{10}	ω^5	ω^2	0

Przyjmując dwa elementy początkowe 1 i 0, można wygenerować sekwencję okresową:

$$10\alpha^2\alpha^2 1\alpha^2 0\alpha\alpha\alpha^2\alpha\ 011\alpha 10\alpha^2\ldots$$

Wykorzystując tę sekwencję pseudolosową, możemy zapisać elementy ciała $GF(4^2)$ w postaci wektorowej:

$$\begin{array}{llll} 0 = [0\ 0], & 1 = [1\ 0], & \omega = [0\ \alpha^2], & \omega^2 = [\alpha^2\ \alpha^2], \\ \omega^3 = [\alpha^2\ 1], & \omega^4 = [1\ \alpha^2], & \omega^5 = [\alpha^2\ 0], & \omega^6 = [0\ \alpha], \\ \omega^7 = [\alpha\ \alpha], & \omega^8 = [\alpha\ \alpha^2], & \omega^9 = [\alpha^2\ \alpha], & \omega^{10} = [\alpha\ 0], \\ \omega^{11} = [0\ 1], & \omega^{12} = [1\ 1], & \omega^{13} = [1\ \alpha], & \omega^{14} = [\alpha\ 1]. \end{array}$$

Wyrażenia te umożliwiają obliczenie tabliczki dodawania ciała $GF(4^2)$.

Tabliczkę mnożenia obliczamy, korzystając z zależności (1.2.2). Tabliczki działań ciała $GF(2^4)$ pokazano w tabeli 1.6.3. Podczas dodawania elementów ciała $GF(4^2)$ współrzędne wektorów należy dodawać zgodnie z zasadami dodawania w ciele $GF(4)$. Na przykład

$$\omega^2 + \omega^4 = \left[\alpha^2\ \alpha^2\right] + \left[1\,\alpha^2\right] = \left[\alpha\ 0\right] = \omega^{10}.$$

Tabliczki działań ciała $GF(4)$ pokazano w tabeli 1.5.1.

1.7. Realizacja działań w ciałach skończonych

1.7.1. Logarytmy Zecha

Logarytmy Zecha ułatwiają programową realizację dodawania w ciałach skończonych. Przyjmujemy, że elementy ciała skończonego $GF(q)$ charakterystyki p wyrażamy za pomocą potęg elementu pierwotnego $\alpha: 0, 1, \alpha, \alpha^2, \ldots, \alpha^{q-2}$. Wykładniki potęg są liczbami całkowitymi liczonymi modulo $q-1$. Element zerowy w postaci potęgowej można zapisać za pomocą symbolu $\alpha^{-\infty} = 0$.

Logarytm Zecha oznaczamy przez $Z(x)$ i definiujemy za pomocą równania

$$\alpha^{Z(x)} = \alpha^x + 1. \tag{1.7.1}$$

Dla ciał charakterystyki 2: $Z(0) = -\infty$, a $Z(-\infty) = 0$.

Dla ciał charakterystyki $p>2$: $Z((q-1)/2) = -\infty$, a $Z(-\infty) = 0$.

Dodawanie z wykorzystaniem logarytmu Zecha wykonuje się następująco

$$\alpha^x + \alpha^y = \alpha^x\left(1 + \alpha^{y-x}\right) = \alpha^{x+Z(y-x)} \quad \text{dla} \quad y > x. \tag{1.7.2}$$

Implementacja tej metody dodawania wymaga, aby utworzyć tablice logarytmów Zecha albo na bieżąco obliczać wartości logarytmów Zecha. Logarytmy Zecha można obliczać z zależności:

$$Z\left((q-1-x)p^i \pmod{q-1}\right) = \left(Z(x) - x\right)p^i \pmod{q-1}, \tag{1.7.3}$$

$$Z\left(x\, p^i \pmod{q-1}\right) = Z(x)\, p^i \pmod{q-1}. \tag{1.7.4}$$

Wzory te łatwo dają się wyprowadzić. W celu wyprowadzenia wzoru (1.7.3) wykorzystujemy przekształcenia

$$\alpha^{Z(x)-x} = \left(\alpha^x + 1\right)\alpha^{-x} = \alpha^{-x} + 1.$$

Ponieważ dla każdego ciała $GF(q)$, $\alpha^{q-1} = 1$, zatem

$$\alpha^{Z(x)-x} = \alpha^{q-1-x} + 1.$$

Obie strony równania podnosimy do potęgi p^i

$$\left(\alpha^{Z(x)-x}\right)^{p^i} = \left(\alpha^{q-1-x} + 1\right)^{p^i}.$$

Prawa strona wyrażenia jest dwumianem Newtona. Rozważmy rozwinięcie dwumianu Newtona nad ciałem skończonym

$$(a+b)^p = a^p + \binom{p}{1} a^{p-1}b + \binom{p}{2} a^{p-2}b^2 + \ldots + \binom{p}{p-1} a\,b^{p-1} + b^p.$$

Ponieważ w ciałach skończonych charakterystyki p: $p(\mathrm{mod}\ p) \equiv 0$, więc każdy ze składników zawierający czynnik p będzie równy 0, dlatego też: $(a+b)^p = a^p + b^p$.

Uwzględniając powyższy rezultat, otrzymamy

$$\alpha^{(Z(x)-x)p^i} = \alpha^{(p-1-x)p^i} + 1 = \alpha^{Z\left((q-1-x)p^i\right)}.$$

Następnie porównujemy wykładniki potęg pierwszego i ostatniego wyrażenia i wprowadzamy funkcję mod $(q-1)$. W wyniku tych działań otrzymujemy wzór (1.7.3)

$$Z\left((q-1-x)p^i\,(\mathrm{mod}\ q-1)\right) = \left(Z(x)-x\right)p^i\,(\mathrm{mod}\ q-1).$$

Wzór (1.7.4) wyprowadzamy, wykorzystując dwumian Newtona nad ciałem skończonym

$$\left(\alpha^x + 1\right)^p = \alpha^{px} + 1.$$

Wykorzystując definicję logarytmu Zecha, możemy napisać

$$\alpha^{Z(x)p} = \alpha^{Z(xp)}.$$

Porównując wykładniki potęg i wprowadzając funkcję mod $(q-1)$, otrzymamy wzór (1.7.4)

$$Z\left(x\,p^i\,(\mathrm{mod}\ q-1)\right) = Z(x)\,p^i\,(\mathrm{mod}\ q-1).$$

Z każdego ze wzorów (1.7.3) i (1.7.4) można obliczyć po około połowie logarytmów Zecha ciała skończonego. Trzeba jednak za pomocą innych metod znaleźć wartość logarytmów Zecha dla jednego lub kilku elementów ciała. Dla ciała $GF(8)$ wystarczy znać logarytm Zecha jednego elementu ciała. Ze wzrostem liczby elementów ciała wzrasta też liczba logarytmów Zecha, które są niezbędne do wykonania obliczeń za pomocą tej metody. Liczba ta ma związek z liczbą wielomianów nierozkładalnych, będących czynnikami dwumianu $x^{p^m-1} - 1$ i wyraża się wzorem: $A/2+B$, gdzie B jest liczbą wielomianów samoodwrotnych, a A – liczbą pozostałych wielomianów.

P r z y k ł a d 1.7.1.

Obliczanie logarytmów Zecha.

W przykładzie obliczymy logarytmy Zecha dla ciała $GF(2^3)$, korzystając ze wzorów (1.7.3) i (1.7.4). Przyjmujemy, że dla rozważanego ciała $\alpha + 1 = \alpha^3$. Z definicji logarytmu Zecha mamy $Z(1) = 3$. Do wzorów (1.7.3) i (1.7.4) podstawiamy $q = 8$, $x = 1$, $p = 2$. Ze wzoru (1.7.3) możemy obliczyć logarytmy Zecha dla trzech elementów. Podstawiamy w tym celu $i = 0, 1, 2$:

$$i = 0, \quad Z(6) = 3 - 1 = 2;$$
$$i = 1, \quad Z(6 \cdot 2 \pmod 7) = (3-1)2 = 4, \quad Z(5) = 4;$$
$$i = 2, \quad Z(6 \cdot 4 \pmod 7) = (3-1)4 \pmod 7, \quad Z(3) = 1.$$

Dla innych wartości *i* wyniki będą się powtarzać.

Ze wzoru (1.7.4), podstawiając $i = 1, 2,$ otrzymamy logarytmy Zecha dla kolejnych dwóch elementów ciała:

$$i = 1, \quad Z(1 \cdot 2) = 3 \cdot 2, \quad Z(2) = 6;$$
$$i = 2, \quad Z(1 \cdot 4) = 3 \cdot 4 \pmod 7, \quad Z(4) = 5.$$

Podobnie jak poprzednio, dla innych wartości *i* wyniki będą się powtarzać. Wszystkie wartości logarytmów Zecha dla ciała $GF(2^3)$ zebrano w tabeli 1.7.1.

Tabela 1.7.1. Logarytmy Zecha dla ciała $GF(8)$

x	$-\infty$	0	1	2	3	4	5	6
$Z(x)$	0	$-\infty$	3	6	1	5	4	2

Mając tabelę logarytmów Zecha, można łatwo obliczyć sumę dwóch dowolnych elementów ciała ze wzoru 1.7.2

$$\alpha^x + \alpha^y = \alpha^x \left(1 + \alpha^{y-x}\right) = \alpha^{x+Z(y-x)} \quad \text{dla} \quad y > x.$$

Na przykład

$$\alpha^2 + \alpha^5 = \alpha^{2+Z(5-2)} = \alpha^{2+Z(3)} = \alpha^3.$$

Tabela 1.7.2. Logarytmy Zecha dla ciał charakterystyki dwa

q	Logarytmy Zecha dla $x = 1, 2, ..., q-2$
4	2 1
8	3 6 1 5 **4** 2
16	4 8 14 1 **10** 13 9 2 7 **5** 12 11 6 3
32	20 9 26 18 **8** 21 29 5 2 **16** 12 11 17 27 **25** 10 13 4 30 **1** 6 24 28 22 **15** 3 14 23 7 **19**
64	6 12 32 24 **62** 1 26 48 45 **61** 25 2 35 52 **23** 33 47 27 56 **59** 42 50 15 4 **11** 7 18 41 60 **46** 34 3 16 31 **13** 54 44 49 43 **55** 28 21 39 37 **9** 30 17 8 38 **22** 53 14 51 36 **40** 19 58 57 20 **29** 10 5

Przedstawione tu zasady obliczania logarytmów Zecha można łatwo zastosować do programowej realizacji działań w ciałach skończonych. W tabeli 1.7.2 zamieszczono wartości logarytmów Zecha dla kilku ciał charakterystyki dwa. W tabeli tej pominięto wartości logarytmów Zecha dla elementów ciała 0 i 1. Aby ułatwić posługiwania się tabelą, co piątą wartość wydrukowano czcionką pogrubioną.

1.7.2. Programowa realizacja działań w ciałach skończonych

Elementy rozszerzonego ciała skończonego mogą mieć postać macierzy, wektorów lub wielomianów i nie nadają się bezpośrednio do programowej realizacji działań algebraicznych. Aby było możliwe zastosowanie komputerowych technik obliczeniowych w tej dziedzinie, należy przedstawić elementy ciała w postaci liczbowej i określić działania na tych liczbach.

W tym celu można przyjąć odwzorowanie zbioru elementów ciała $GF(q)$ na q-elementowy zbiór całkowitych liczb dodatnich:

$$\sigma : \quad \left\{0,1,\alpha,\alpha^2,\ldots,\alpha^{q-2}\right\} \to \left\{0,1,2,3,\ldots,q-1\right\}.$$

Odwzorowanie to określa funkcja:

$$\sigma\left(\alpha^x\right) = \begin{cases} x+1 & \text{dla} \quad \alpha^x \neq 0, \\ 0 & \text{dla} \quad \alpha^x = 0. \end{cases} \tag{1.7.5}$$

Tak więc zerowy element ciał odwzorowuje się na zero, a elementy niezerowe α^x odwzorowują się na liczby równe $x+1$. Odwzorowanie to jest wzajemnie jednoznaczne i izomorficzne.

Dla odwzorowania σ istnieje odwzorowanie odwrotne σ^{-1}

$$\sigma^{-1}: \quad \left\{0,1,2,3,\ldots,q-1\right\} \to \left\{0,1,\alpha,\alpha^2,\ldots,\alpha^{q-2}\right\},$$

przy czym

$$\sigma^{-1}\left(x\right) = \begin{cases} \alpha^{x-1} & \text{dla} \quad x > 0, \\ 0 & \text{dla} \quad x = 0. \end{cases} \tag{1.7.6}$$

Dla większości zastosowań wystarczy zdefiniować cztery funkcje:

1. Sumę $S(x,y)$.
2. Element przeciwny $OE(x)$.
3. Iloczyn $P(x,y)$.
4. Element odwrotny $IE(x)$.

Po uwzględnieniu powyższych odwzorowań można obliczyć te cztery funkcje. Wyznaczono je w [2.2]:

$$S(x,y)=\begin{cases} y+Z(x-y)-1(\text{mod } q-1)+1 \quad \text{dla} \quad x,y\neq 0 \quad \text{i} \quad x>y, \\ x+y \quad \text{dla} \quad x=0 \quad \text{lub} \quad y=0, \\ 0 \quad \text{dla} \quad x\neq 0 \quad \text{i} \quad y=OE(x), \end{cases} \tag{1.7.7}$$

$$OE(x)=\begin{cases} \left(x+(q-1)/2\right)(\text{mod } q-1) \quad \text{dla} \quad x\neq 0 \quad \text{i} \quad p>2, \\ x \quad \text{dla} \quad x\neq 0 \quad \text{i} \quad p=2, \\ 0 \quad \text{dla} \quad x=0, \end{cases} \tag{1.7.8}$$

$$P(x,y)=\begin{cases} 1+\left(x+y-2(\text{mod } q-1)\right) \quad \text{dla} \quad x>0 \quad \text{i} \quad y>0, \\ 0 \quad \text{dla} \quad x=0 \quad \text{lub} \quad y=0, \end{cases} \tag{1.7.9}$$

$$IE(x)=\begin{cases} q+1-x \quad \text{dla} \quad x>1, \\ 1 \quad \text{dla} \quad x=1. \end{cases} \tag{1.7.10}$$

Do obliczenia sumy $S(x,y)$ wykorzystuje się logarytmy Zecha opisane w poprzednim punkcie. Za pomocą powyższych wzorów można w dowolnym języku programowania napisać procedury realizujące algorytmy działań w rozszerzonych ciałach skończonych.

Realizację programową dodawania i mnożenia elementów ciała $GF(8)$ ilustrują poniższe przykłady napisane w języku Pascal. W programie obliczającym sumę elementów ciała wartości logarytmów Zecha są przechowywane w tablicy ZT. Zamiast tej tablicy do programu można dołączyć instrukcje do obliczania logarytmów Zecha napisanych zgodnie ze wzorami (1.7.3) i (1.7.4).

Funkcja do obliczania sumy elementów ciała $GF(8)$ według wzoru (1.7.7):

```
const
  q=8;
  ZT:array[1..(q-2)] of word =(3,6,1,5,4,2);   {logarytmy Zecha}

function S(x,y:integer):integer;           {suma elementów ciała}
var
  h:integer;
begin
  if x*y=0
    then S:=x+y
    else if x-y=0
           then S:=0
           else begin
                  if y>x then
                           begin
                             h:=x;
```

```
                    x:=y;
                    y:=h;
                  end;
              S:=((y+ZT[x-y]-1) mod (q-1))+1;
            end;
end;
```

Funkcja do obliczania iloczynu elementów ciała $GF(8)$ według wzoru (1.7.9):

```
const
  q=8;
function P(x,y:integer):integer;        {iloczyn elementów ciała}
begin
  if (x=0) or (y=0) then P:=0
                    else P:=1+((x+y-2)mod(q-1));
end;
```

Więcej informacji o programowej realizacji działań w ciałach skończonych można znaleźć w [2.2].

1.7.3. Układy mnożenia i dzielenia wielomianów

Działania w ciałach skończonych można realizować za pomocą układów logicznych. Układy realizujące operacje na elementach ciała $GF(2)$ konstruuje się bezpośrednio za pomocą standardowych układów logicznych. Podobnie, za pomocą układów logicznych, można konstruować urządzenia realizujące działania w ciałach rozszerzonych charakterystyki dwa i dlatego są one najczęściej stosowane w praktyce.

W rozdziale 1.4 podano konstrukcję generatora sekwencji okresowych, który zbudowano opierając się na rejestrze przesuwnym ze sprzężeniem zwrotnym. Tutaj omówimy dodatkowo układy mnożenia i dzielenia wielomianów. Układy te stanowią podstawę do konstrukcji urządzeń kodowych i kryptograficznych. Schematy ogólne układów dotyczą dowolnych ciał skończonych, a przykłady odnoszą się do ciała $GF(2)$.

[8]

Układ mnożenia wielomianów

Rozważmy mnożenie dowolnego wielomianu wejściowego

$$a(x) = a_k x^k + a_{k-1} x^{k-1} + \ldots + a_1 x + a_0$$

przez wielomian stały

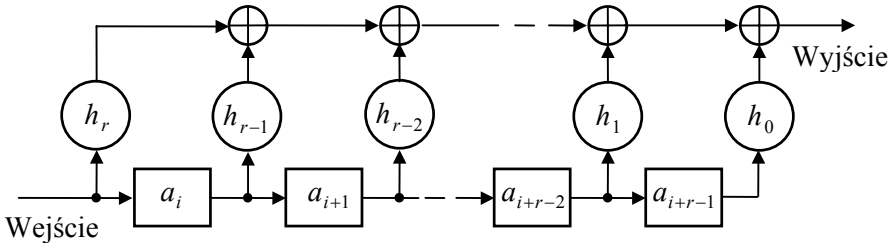
$$h(x) = h_r x^r + h_{r-1} x^{r-1} + \ldots + h_1 x + h_0 .$$

W wyniku mnożenia otrzymamy

$$a(x)h(x) = a_k h_r x^{k+r} + \left(a_{k-1} h_r + a_k h_{r-1}\right) x^{k+r-1} +$$

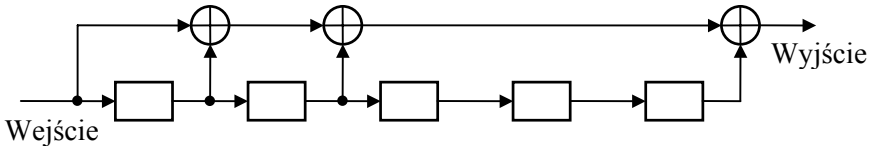
$$+\left(a_{k-2}h_r + a_{k-1}h_{r-1} + a_k h_{r-2}\right)x^{k+r-2} + \ldots$$
$$+\left(a_0 h_2 + a_1 h_1 + a_2 h_0\right)x^2 + \left(a_0 h_1 + a_1 h_0\right)x + a_0 h_0 .$$

Schemat ogólny układu mnożącego pokazano na rys. 1.7.1. Zastosowane elementy opisano na rys. 1.4.1, w p. 1.4.2. Układ mnożący zawiera rejestr przesuwny, zawierający *r* przerzutników, który na początku jest wyzerowany. Kiedy na wejście zostanie podany pierwszy element a_k wielomianu $a(x)$, na wyjściu pojawi się wartość odpowiadająca pierwszemu współczynnikowi $a_k h_r$ iloczynu $a(x)h(x)$. W tym czasie wszystkie przerzutniki rejestru są w stanie 0. Po impulsie taktującym na wejściu pojawi się wartość a_{k-1}, a a_k zostanie wpisane do pierwszego przerzutnika. Na wyjściu zaś pojawi się wartość $a_{k-1}h_r + a_k h_{r-1}$, to jest wartość równa drugiemu współczynnikowi iloczynu. Dalej układ pracuje w podobny sposób. Po $r+k$ impulsach zegarowych zawartość rejestru będzie: $0, 0, 0, \ldots, a_0$, a na wyjściu pojawi się wartość $a_0 h_0$, która jest równa ostatniemu współczynnikowi iloczynu.



Rys. 1.7.1. Układ mnożenia wielomianów

Układ mnożenia wielomianu wejściowego przez wielomian $x^5 + x^4 + x^3 + 1$ pokazano na rys.1.7.2.

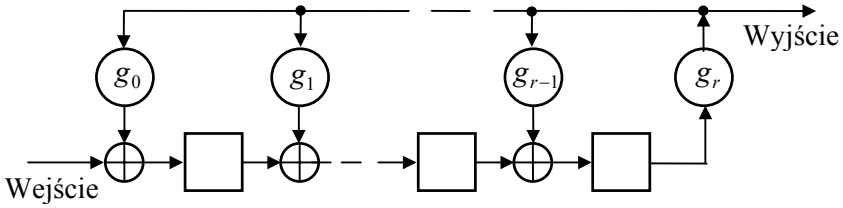


Rys. 1.7.2. Układ mnożenia przez wielomian $x^5 + x^4 + x^3 + 1$

Układ dzielenia wielomianów

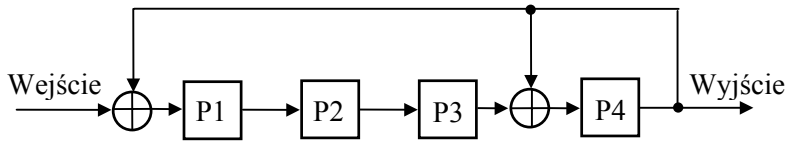
Do dzielenia dowolnych wielomianów wejściowych przez wielomian stały stosuje się rejestry przesuwne ze sprzężeniem zwrotnym. Schemat ogólny układu do dzielenia przez wielomian $g(x) = g_r x^r + g_{r-1}x^{r-1} + \ldots + g_1 x + g_0$ pokazano na rys. 1.7.3.

Działanie układu dzielącego wielomiany prześledzimy na przykładzie układu dzielenia dowolnego wielomianu przez wielomian $x^4 + x^3 + 1$, który pokazano na rys. 1.7.4.



Rys. 1.7.3. Układ do dzielenia wielomianów

Działanie układu pokazanego na rys. 1.7.4 opisano w tabeli 1.7.3. Jest to przykład dzielenia wielomianu $x^7 + x^6 + x^5 + x^3 + x^2 + x + 1$ przez wielomian $x^4 + x^3 + 1$, zapisanych w formie współczynników. Z lewej strony pokazano przykład dzielenia tych wielomianów w postaci pisemnej, a z prawej – tabelę opisującą działanie układu dzielącego. W kolejnych kolumnach tabeli podano: impulsy taktujące *T*, elementy wejściowe, stany czterech przerzutników rejestru P1÷P4 i ciąg wyjściowy. Wiersze tabeli zawierają stany przerzutników po impulsie taktującym.



Rys. 1.7.4. Układ dzielenia przez wielomian x^4+x^3+1

Tabela 1.7.3. Działanie układu dzielącego przez wielomian x^4+x^3+1

```
                1 0 1 1
1 1 0 0 1 | 1 1 1 0 1 1 1 1
            1 1 0 0 1
            -----------
              0 1 0 0 1
              0 0 0 0 0
              ---------
                1 0 0 1 1
                1 1 0 0 1
                ---------
                  1 0 1 0 1
                  1 1 0 0 1
                  ---------
                    1 1 0 0
```

T	We.	P1	P2	P3	P4	Wy.
1	1	0	0	0	0	0
2	1	1	0	0	0	0
3	1	1	1	0	0	0
4	0	1	1	1	0	0
5	***1***	**0**	**1**	**1**	**1**	1
6	***1***	**0**	**0**	**1**	**0**	0
7	***1***	**1**	**0**	**0**	**1**	1
8	***1***	**0**	**1**	**0**	**1**	1
		0	**0**	**1**	**1**	

Na początku rejestr jest wyzerowany, a na wejście podany pierwszy najbardziej znaczący element dzielnej. Po impulsie taktującym wartość ta zostanie wpisana do pierwszego przerzutnika rejestru P1, a na wejściu pojawi się drugi element dzielnej. Dalej układ działa podobnie.

Po czterech impulsach taktujących nastąpi właściwy proces dzielenia. Elementy wyjściowe zaczną oddziaływać na stan rejestru poprzez sprzężenie zwrotne. Przytoczony przykład pokazuje korelację pomiędzy dzieleniem pisemnym i stanami przerzutników rejestru pokazanymi w tabeli. Dla pokazania tych zależności odpowiadające sobie ciągi bitów napisano czcionką pogrubioną. Podobnie kursywą pogrubioną zaznaczono odpowiadające sobie elementy wejściowe. Ciągi w rejestrze są w odwrotnej kolejności niż odpowiadające im ciągi w przykładzie dzielenia, ponieważ najbardziej znaczące elementy ciągu znajdują się w rejestrze z prawej strony.

Po siedmiu impulsach taktujących dzielenie będzie zakończone, a w rejestrze pozostanie reszta z dzielenia. W przypadku reszty zerowej rejestr będzie wyzerowany.

1.7.4. Układy realizujące działania arytmetyczne w ciałach rozszerzonych

Realizacja działań w ciałach rozszerzonych wymaga skonstruowania układów wielostanowych. Stosunkowo łatwo można skonstruować takie układy, realizujące działania w ciałach charakterystyki dwa. Liczba stanów konstruowanych układów musi być równa liczbie elementów stosowanego ciała. Układy mnożenia i dzielenia w ciałach rozszerzonych będą takie same jak pokazane na rys. 1.7.1 i 1.7.3, jeśli zastosowane tam elementy potraktujemy jako elementy wielostanowe.

Rozważmy realizację działań w ciele $GF(2^4)$. Element pamięciowy dla tego ciała będzie zawierał cztery przerzutniki. Ponadto podczas konstrukcji urządzeń wykorzystuje się układy dodawania i mnożenia przez stałą.

w czasie konstrukcji układów wielostanowych posługujemy się elementami ciała w postaci wektorowej. W tym celu zapiszmy elementy ciała $GF(2^4)$ w postaci wektorowej, stosując metodę podaną w p. 1.5.5. Niech wielomian $x^4 + x + 1$ będzie wielomianem generującym ciało rozszerzone. Elementy ciała rozszerzonego obliczamy z zależności (1.5.4) i otrzymujemy:

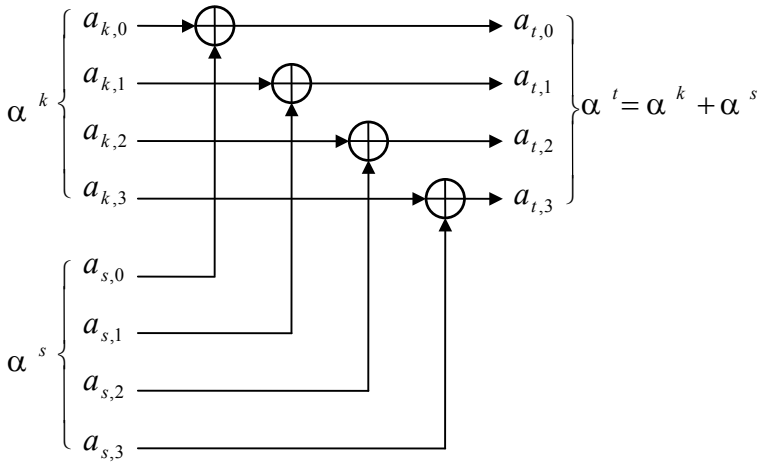
$$
\begin{array}{llll}
0 = [0000], & 1 = [1000], & \alpha = [0100], & \alpha^2 = [0010], \\
\alpha^3 = [0001], & \alpha^4 = [1100], & \alpha^5 = [0110], & \alpha^6 = [0011], \\
\alpha^7 = [1101], & \alpha^8 = [1010], & \alpha^9 = [0101], & \alpha^{10} = [1110], \\
\alpha^{11} = [0111], & \alpha^{12} = [1111], & \alpha^{13} = [1011], & \alpha^{14} = [1001].
\end{array}
$$

Układ sumatora

Dodawanie elementów ciała, przedstawionych w postaci wektorowej, realizuje się przez dodawanie współrzędnych wektorów

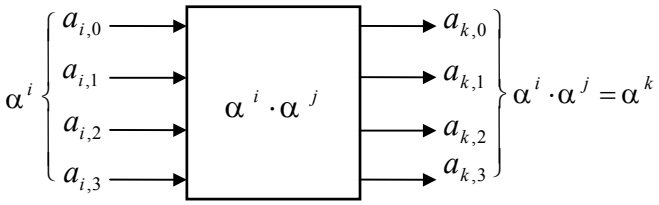
$$\alpha^k + \alpha^s = \left[\left(a_{k,0} + a_{s,0}\right), \ldots, \left(a_{k,3} + a_{s,3}\right)\right].$$

Operacja ta może być bezpośrednio zrealizowana za pomocą bramek Ex-OR w układzie pokazanym na rys. 1.7.5.



Rys. 1.7.5. Sumator elementów ciała $GF(16)$

Układ mnożenia przez stałą



Rys. 1.7.6. Schemat blokowy układu mnożenia przez stałą

Mnożenie przez stałą powoduje zmiany bazy przestrzeni wektorowej. Układ mnożenia przez stałą przedstawia schemat blokowy pokazany na rys. 1.7.6.

Projektowanie układu mnożenia przez stałą pokażemy na przykładzie ciała $GF(4^2)$. Załóżmy, że konstruujemy układ mnożenia przez α^{12}. Do konstrukcji układu wykorzystujemy elementy ciała reprezentowane przez zbiór czterech wektorów niezależnych: 1, α, α^2 i α^3. Po pomnożenie każdego z tych elementów przez α^{12} otrzymamy kolejno elementy ciała: α^{12}, α^{13}, α^{14} i 1. Współrzędne wektorów reprezentujących te dwie grupy elementów ciała $GF(16)$, pokazano w tabeli 1.7.4.

Wykorzystując tabelę 1.7.4, współrzędne wektorów wyjściowych możemy wyrazić jako kombinacje współrzędnych wektorów wejściowych. Otrzymamy wówczas zbiór równań:

$$a_{k,0} = a_{i,0} + a_{i,1} + a_{i,2} + a_{i,3},$$
$$a_{k,1} = a_{i,0},$$

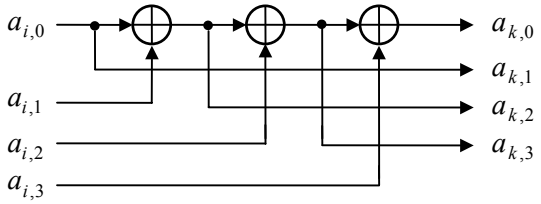
$$a_{k,2} = a_{i,0} + a_{i,1},$$
$$a_{k,3} = a_{i,0} + a_{i,1} + a_{i,2}.$$

Tabela 1.7.4. Elementy wejściowe i wyjściowe układu mnożącego

Elementy wejściowe				
α^i	$a_{i,0}$	$a_{i,1}$	$a_{i,2}$	$a_{i,3}$
1	1	0	0	0
α	0	1	0	0
α^2	0	0	1	0
α^3	0	0	0	1

Elementy wyjściowe				
α^k	$a_{k.0}$	$a_{k,1}$	$a_{k,2}$	$a_{k,3}$
α^{12}	1	1	1	1
α^{13}	1	0	1	1
α^{14}	1	0	0	1
1	1	0	0	0

Na podstawie tych równań można skonstruować układ mnożenia przez stały element α^{12}. Układ taki pokazano na rys. 1.7.7.



Rys. 1.7.7. Układ mnożenia przez α^{12}

Podczas projektowania tego typu układów należy uwzględnić problem optymalizacji. Jako kryterium optymalizacji można np. przyjąć liczbę bramek.

Część 2
Kody korekcyjne

Kodowanie korekcyjne zabezpiecza informację przed błędami w systemach transmisyjnych i pamięciach, dzięki czemu zwiększa się niezawodność systemów informatycznych. Możliwość konstrukcji kodów korekcyjnych przewidział Shannon w 1948 r. W następnych latach nastąpił szybki rozwój teorii kodów, która stała się jedną z najważniejszych dziedzin teorii informacji. Pierwszą książką, ujmującą systematycznie wiadomości z dziedziny kodów korekcyjnych, była monografia [2.4], której pierwsze wydanie ukazało się w 1961 r.

2.1. Elementy transmisji danych

2.1.1. System transmisji danych

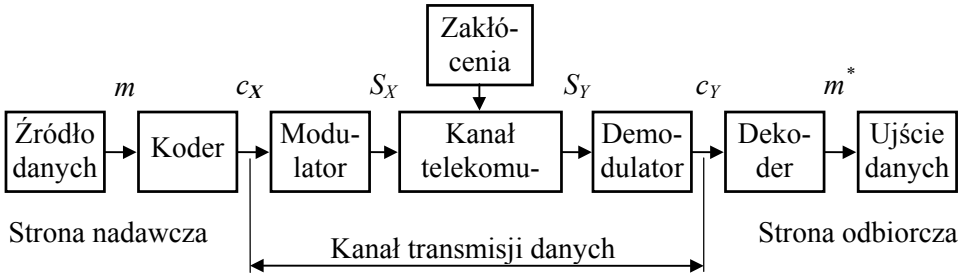
Wiadomości przesyłane w systemach teleinformatycznych mają charakter sygnałów dyskretnych, zwanych też cyfrowymi, a kanały służące do przesyłania takich sygnałów nazywa się kanałami transmisji danych. Kanały transmisji danych są stosowane w sieciach komputerowych, systemach sterowania procesami przemysłowymi itp.

Na sygnał w kanale telekomunikacyjnym działają zakłócenia, które powodują błędy transmisyjne. Skutki zakłóceń można opisać, podając stopień zniekształcenia sygnału cyfrowego na wyjściu demodulatora. Stosuje się w tym celu elementową stopę błędów. *Elementowa stopa błędów* jest prawdopodobieństwem przekłamania elementarnego sygnału cyfrowego w czasie transmisji. Przeciętna stopa błędów w istniejących kanałach telekomunikacyjnych bez korekcji błędów wynosi $10^{-2} \div 10^{-5}$. Wartość ta jest zbyt duża, aby można było zapewnić efektywną pracę systemów informatycznych, których ogniwo stanowi system transmisji danych. W systemach transmisji danych wymaga się stopy błędów rzędu $10^{-6} \div 10^{-9}$.

W kanałach transmisji danych, wykorzystujących standardowe protokoły liniowe, poprawę wierności transmisji osiąga się w wyniku detekcji błędów po stronie odbiorczej i retransmisji błędnych bloków. W tym celu musi być utworzony informacyjny kanał sprzężenia zwrotnego. Większą efektywność tych metod można uzyskać dzięki zastosowaniu kodów korekcyjnych. Kody korekcyjne są też jedyną metodą poprawienia wierności transmisji wszędzie tam, gdzie są trudności z utworzeniem kanału sprzężenia zwrotnego, np. w łączności satelitarnej. Za pomocą kodów korekcyjnych można również zabezpieczać dane przechowywane w pamięciach komputerowych.

Konfigurację systemu transmisji danych z zastosowaniem kodowego podsystemu korekcyjnego pokazano na rys. 2.1.1. Do typowego systemu transmisyjnego dodano

koder i dekoder. Koder realizuje proces kodowania i jest umieszczony między źródłem danych a łączem transmisji danych. Koder przekształca ciąg wiadomości m w ciąg kodowy c_X, który jest następnie poddany procesowi modulacji.



Rys. 2.1.1. System transmisyjny z kodem korekcyjnym

Kanały transmisji danych tworzy się na łączach telekomunikacyjnych, do których dodaje się modulator i demodulator. Do tworzenia łączy transmisji danych używa się różnych kanałów telekomunikacyjnych takich jak: kanały telefoniczne pojedyncze i grupowe, kanały radiowe i łącza światłowodowe. Modulator przekształca sygnały cyfrowe $\boldsymbol{c_X}$ w sygnały S_X dostosowane do parametrów kanału telekomunikacyjnego pod względem pasma i amplitudy. Najczęściej wykorzystuje się w tym celu dyskretną modulację częstotliwości (FSK) lub fazy (PSK) sygnału nośnego.

Sygnał wyjściowy kanału transmisyjnego S_Y podlega procesowi demodulacji, w czasie której następuje odtworzenie postaci cyfrowej sygnału. W technicznej realizacji do modulacji i demodulacji sygnału służą modemy. Konstrukcję i parametry modemów określają zalecenia CCITT (The International Telegraph and Telephone Consultative Committee). Modemy wraz z łączem telekomunikacyjnym tworzą kanał cyfrowy nazywany *kanałem transmisji danych*.

Cyfrowy sygnał wyjściowy kanału transmisji danych c_Y zwykle różni się od sygnału wejściowego c_X. Miejsca, w których występują różnice, nazywają się *błędami transmisyjnymi*. Dekoder układu transmisyjnego koryguje te błędy. W tym celu dekoder wykorzystuje określoną metodę korekcji, która zależy od charakterystyki kanału transmisyjnego, i odtwarza sygnał kodowy. Na podstawie sygnału dekodera estymowany jest sygnał wejściowy m. Jeśli sygnał wyjściowy m^* nie ma takiej samej postaci jak sygnał wejściowy m, oznacza to, że wystąpiły błędy niekorygowalne.

Głównym problemem inżynierskim jest takie zaprojektowanie pary kodera i dekodera, aby:

- osiągnąć wymaganą stopę błędów transmisyjnych,
- przesyłać dane z możliwie największą szybkością.

W projektowaniu kodera i dekodera musi się uwzględniać parametry kanału transmisyjnego, a przede wszystkim rodzaj występujących w kanale zakłóceń i powodowanych przez nie błędów.

2.1.2. Zakłócenia i błędy w kanałach transmisyjnych

Podczas projektowania systemów transmisji danych na potrzeby sieci teleinformatycznych wymagana jest znajomość opisu statystycznego występujących w kanale błędów. Błędy transmisyjne są powodowane zniekształceniami sygnałów w kanale i zakłóceniami wywoływanymi przez czynniki zewnętrzne, np. w przypadku kanałów przewodowych zakłócenia są indukowane przez zewnętrzne pola elektromagnetyczne. Błędy powstające wskutek działania zakłóceń mają charakter losowy i można je opisać za pomocą funkcji stochastycznych.

W kanałach telekomunikacyjnych występują dwa rodzaje zakłóceń: multyplikatywne Z_M i addytywne Z_A. Sposób oddziaływania zakłóceń na sygnał użyteczny określa zależność

$$S_Y(t) = Z_M(t)\, S_X(t) + Z_A,$$

gdzie $S_X(t)$ jest sygnałem wejściowym kanału, a $S_Y(t)$ – sygnałem wyjściowym.

Przyczyną zakłóceń multyplikatywnych są zmiany parametrów kanału. Parametry te można w pewnym stopniu korygować i w ten sposób eliminować błędy przez nie powodowane. Przyczyną zakłóceń addytywnych są szumy cieplne i sygnały indukowane przez pola zewnętrzne. Zakłócenia addytywne dzielą się na fluktuacyjne i impulsowe. Zakłócenia fluktuacyjne mają postać ciągłych w czasie procesów przypadkowych. Ich widmo pokrywa zwykle całe pasmo kanału, a rozkład amplitud jest gaussowski.

Zakłócenia impulsowe pochodzą od czynników zewnętrznych i cechuje je skupienie energii w przypadkowych okresach czasu i pewnym paśmie częstotliwości. Rozkład amplitudowy zakłóceń impulsowych w kanałach telefonicznych można aproksymować rozkładem hiperbolicznym. Do statystycznego opisu czasu trwania tych zakłóceń i przerw między nimi nadają się stochastyczne procesy Poissona.

Znajomość zjawisk fizycznych w kanale pozwala ocenić, jakie zakłócenia mogą w nim dominować i jakich możemy spodziewać się błędów. Podczas projektowania systemów kodowych ważne są jednak nie same zakłócenia, ale skutki działania zakłóceń w kanale, to jest intensywność błędów i ich rozkład czasowy na wyjściu demodulatora. Jeśli w kanale dominują błędy spowodowane zakłóceniami fluktuacyjnymi, to rozkład błędów jest zbliżony do prostokątnego i nazywamy je *błędami losowymi* (random errors). Zakłócenia impulsowe powodują powstanie *błędów grupowych*, nazywanych też błędami seryjnymi (burst errors).

Do opisu kanałów transmisji danych w praktyce najczęściej używa się sygnałów binarnych, a parametry kanału opisuje się za pomocą macierzy uwzględniającej prawdopodobieństwa przekłamania tych sygnałów. Jeśli prawdopodobieństwa przekłamania jedynek i zer są takie same, kanał nazywamy *kanałem symetrycznym*.

Aby skonstruować dobry kod, nie wystarcza znajomość elementowej stopy błędów, która nic nie mówi o konfiguracji błędów. Błędy często mają tendencję do grupowania się w pakiety. Wynika to z charakteru zakłóceń impulsowych i przerw

w transmisji. W wyniku tego powstają pakiety błędów rozciągające się na przestrzeni od kilku do kilkudziesięciu bitów. Dlatego też, aby zaprojektować efektywny kod gwarantujący istotną poprawę wierności transmisji, trzeba znać najbardziej prawdopodobne sekwencje błędów.

Projektantom systemów kodowych najwygodniej jest posługiwać się wynikami badań statystycznych kanałów. Badania takie obejmują, między innymi, pomiary stopy błędów i prawdopodobieństwa błędów grupowych. Statystyki te mają istotne znaczenie w projektowaniu systemów do korekcji błędów transmisyjnych.

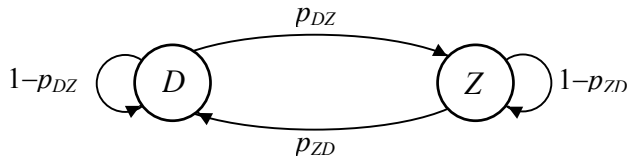
Charakterystyki błędów w kanale transmisyjnym przedstawia się najczęściej w postaci funkcji *S/N*, gdzie *S* jest mocą sygnału, a *N* mocą zakłóceń. Typowy wykres elementowej stopy błędów dla kanału transmisyjnego z błędami losowymi pokazano na rys. 2.7.1.

2.1.3. Model binarnego kanału transmisji danych

Aby analizować działanie kanałów transmisji danych, musimy określić model matematyczny kanału z zakłóceniami. Podstawą do budowy modelu są ustalone zbiory parametrów, wyznaczone na podstawie eksperymentalnych badań statystycznych kanałów. Modele takie służą do symulacji komputerowej kanałów rzeczywistych i są stosowane do rozwiązywania problemów projektowych, konstrukcyjnych i eksploatacyjnych oraz do badania efektywności zastosowanych metod ochrony informacji przed błędami.

Modele matematyczne mogą być opisowe lub przyczynowe. Modele opisowe charakteryzują się czysto matematycznym podejściem do zjawiska powstawania błędów, a modele przyczynowe uwzględniają również zjawiska fizyczne towarzyszące powstawaniu błędów.

Modelem przyczynowym kanału binarnego jest dwustanowy model Gilberta, opierający się na teorii łańcuchów Markowa, i jego uogólnienie. Modele te uwzględniają w pewnym stopniu fizyczną naturę błędów występujących w kanale rzeczywistym.



Rys. 2.1.2. Dwustanowy model Gilberta

W klasycznym dwustanowym modelu Gilberta zakłada się możliwość pozostawania kanału w jednym z dwóch stanów: w dobrym *D* lub złym *Z*. W stanie dobrym prawdopodobieństwo błędów wynosi $p_D \approx 0$, a w stanie złym występują błędy niezależne z prawdopodobieństwem $p_Z >> p_D$. Model ten pokazano na rys. 2.1.2. Prawdopodobieñstwa przejœæ miêdzy stanami opisuje macierz

$$M_G = \begin{bmatrix} 1 - p_{DZ} & p_{DZ} \\ p_{ZD} & 1 - p_{ZD} \end{bmatrix}.$$

Skonstruowany w ten sposób model dla niezbyt dużych p_{DZ} i p_{ZD} generuje błędy seryjne o średniej długości równej $1/p_{ZD}$ oraz serie elementów bezbłędnych o średniej długości $1/p_{DZ}$. W błędzie seryjnym prawdopodobieństwo przekłamania jednego elementu wynosi p_Z. Elementową stopę błędów określa zależność

$$p_e = P_D \cdot p_D + P_Z \cdot p_Z,$$

gdzie P_D i P_Z są prawdopodobieństwami granicznymi pozostawania kanału w stanach D i Z, które wynoszą:

$$P_D = \frac{p_{DZ}}{p_{DZ} + p_{ZD}}, \quad P_Z = \frac{p_{ZD}}{p_{DZ} + p_{ZD}}.$$

Model ten nie jest dokładnym przybliżeniem kanałów rzeczywistych. Znacznie lepsze wyniki dają wielostanowe modele Gilberta. Jednak analiza takich modeli jest znacznie bardziej pracochłonna.

2.2. Charakterystyka kodów

2.2.1. Typy kodów korekcyjnych

Kodowaniem informacji nazywamy wzajemnie jednoznaczne przyporządkowanie elementów zbioru informacyjnego elementom zbioru sygnałów, za pomocą których informacje te będą albo przesyłane kanałem transmisyjnym, albo zapisywane w pamięciach. Kod jest zatem zbiorem zakodowanych informacji. Kodowanie stosuje się w celu kompresji informacji, szyfrowania informacji lub przedstawienia informacji w formie odpornej na błędy. Kody umożliwiające wykrycie lub korekcję błędów nazywają się *kodami korekcyjnymi* lub nadmiarowymi i będą one przedmiotem dalszych rozważań.

Istnieje wiele typów kodów korekcyjnych. Historycznie kody te dzielą się na dwa typy: *kody blokowe* i *kody rekurencyjne*, nazywane również kodami splotowymi.

Kody blokowe wymagają rozbicia ciągu informacyjnego na bloki *k*-elementowe i wykonania operacji kodowania na każdym bloku niezależnie od innych bloków. Podczas kodowania do bloku informacji jest dołączona *sekwencja kontrolna* umożliwiająca wykrycie lub korekcję błędów.

Kody splotowe nie wymagają podziału informacji na bloki, a kodowanie odbywa się na bieżąco, w takt napływającej informacji. Elementy kodu są uzależnione od bieżącego elementu informacji oraz od pewnej liczby elementów poprzednich. Koder kodu splotowego przyjmuje ciąg informacyjny i przetwarza go na ciąg kodowy o większej liczbie znaków.

W literaturze dotyczącej teorii kodowania rozróżnia się *kody liniowe* i *kody cykliczne*. Obie te grupy kodów spełniają kryterium liniowości. Właściwość liniowości oznacza, że suma dwóch dowolnych wektorów kodowych daje wektor należący do zbioru wektorów kodowych tego kodu.

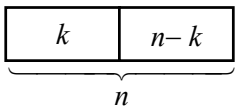
Kody liniowe i cykliczne różnią się konstrukcją i strukturą matematyczną. Do konstrukcji kodów liniowych wykorzystuje się grupy addytywne i wektorowe przestrzenie liniowe, a do konstrukcji kodów cyklicznych – pierścienie wielomianów nad ciałami skończonymi.

W praktyce najczęściej używa się blokowych kodów cyklicznych i dlatego kodom tym poświęcono najwięcej miejsca. Na początku będą przedstawione kody binarne, a następnie kody nad ciałami rozszerzonymi.

2.2.2. Struktura kodu blokowego

Kodowanie informacji za pomocą kodu blokowego wymaga podziału ciągu informacji na bloki *k*-elementowe. W wyniku kodowania w koderze powstaje ciąg *n*-elementowy, gdzie $n>k$. Ciąg *n*-elementowy na wyjściu kodera nazywamy *wektorem kodowym* lub słowem kodowym, a *n* jest długością wektora kodowego lub po prostu długością kodu. Kody blokowe oznaczamy symbolem (n,k). Strukturę wektora ko-

dowego pokazano na rys 2.2.1. Wektor kodowy zawiera k elementów informacyjnych i $n-k$ elementów kontrolnych, zwanych też nadmiarowymi lub elementami kontroli parzystości. W kodach binarnych elementami kodu są bity.



Rys. 2.2.1. Wektor kodu blokowego (n,k)

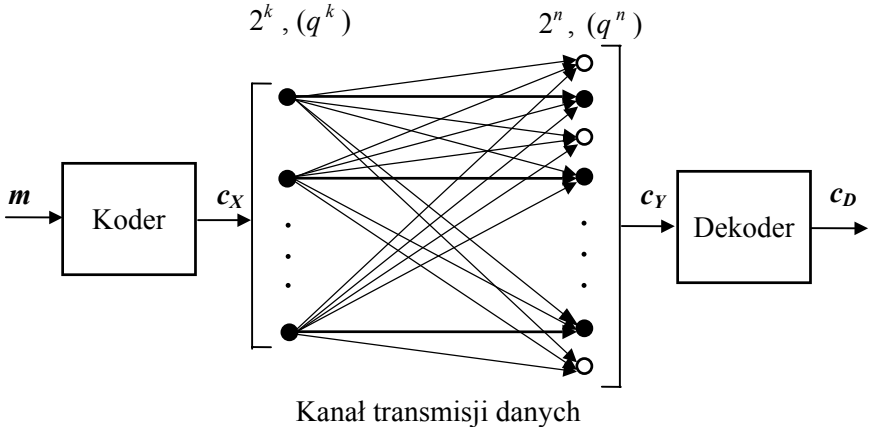
Część kontrolna wektora kodowego zawiera dodatkową informację umożliwiającą korekcję błędów transmisyjnych. Z drugiej strony część kontrolna zwiększa objętość przesyłanej informacji. Do oceny względnej długości części informacyjnej wektora kodowego stosuje się współczynnik nazywany sprawnością kodu (code rate). *Sprawność kodu R* jest stosunkiem liczby znaków wprowadzonych do kodera k do liczby znaków wyjściowych kodera n

$$R = k/n. \tag{2.2.1}$$

Nadmiar informacji wprowadzonej w procesie kodowania można ocenić za pomocą *nadmiaru kodowego* równego 1–R.

Kod blokowy, w którym można odróżnić elementy informacyjne od elementów kontrolnych nazywamy *kodem systematycznym*.

Przeanalizujmy system transmisyjny z korekcją błędów pokazany na rys. 2.1.1. Z systemu tego można wyodrębnić podsystem kodowy pokazany na rys. 2.2.2. Jeśli na wejście kodera są podawane binarne bloki informacji zawierające po k bitów, to koder wygeneruje 2^k różnych wektorów kodowych o długości n, które zostaną podane na wejście kanału transmisyjnego. Ten zbiór wektorów nazywamy *kodem*.



Rys. 2.2.2. Elementy podsystemu kodowego

Podczas transmisji może nastąpić zmiana niektórych bitów wektorów kodowych w wyniku działania zakłóceń w kanale. Wskutek tego 2^k-elementowy zbiór wektorów wejściowych kanału transmisyjnego zamieni się w zbiór wyjściowy, zawierający 2^n elementów. 2^n-elementowy zbiór wyjściowy nazywa się *przestrzenią wektorową* nad ciałem binarnym. Przestrzeń wektorowa oprócz wektorów kodowych zawiera wektory, które nie są wektorami kodowymi. Ilustruje to rys. 2.2.2. Na tym rysunku wektory kodowe oznaczono ciemnymi punktami, a wektory niekodowe – punktami jasnymi. W nawiasach podano liczby wektorów nad ciałem rozszerzonym q-elementowym.

Jeśli przyjmiemy oznaczenia z rys. 2.2.2, to wektor wejściowy dekodera $\boldsymbol{c}_Y$ będzie sumą wektora wyjściowego kodera $\boldsymbol{c}_X$ i wektora błędów $\boldsymbol{e}$

$$\boldsymbol{c}_Y = \boldsymbol{c}_X + \boldsymbol{e}. \tag{2.2.2}$$

W wyniku nałożenia się błędów na wektory kodowe podczas transmisji mogą wystąpić następujące zdarzenia:

1. Wektor kodowy przechodzi przez kanał bez zmiany.
2. Wektor kodowy zostaje zamieniony na inny wektor kodowy.
3. Wektor kodowy zostaje zamieniony na wektor niekodowy.

Przypadek pierwszy ma miejsce, gdy nie ma błędów transmisyjnych, a pozostałe zdarzenia są spowodowane błędami transmisyjnymi.

Gdy wektor kodowy zostanie zmieniony w inny wektor kodowy, wówczas dekoder nie ma możliwości odróżnienia błędnie odebranego wektora i nie może wykryć błędu. Wektor taki powstaje w wyniku nałożenia się wektora błędu na wektor kodowy, gdy wektor błędu ma postać wektora kodowego. Zatem liniowy kod blokowy nie wykrywa tylko takich ciągów błędów, które są same ciągami kodowymi. Błędy takie są niekorygowalne.

Przestrzeń wektorowa oprócz elementów kodu zawiera $2^n - 2^k$ ciągów niekodowych, które nie zostały nadane. Powstają one po stronie odbiorczej w wyniku zdarzenia trzeciego. Dekoder jest tak skonstruowany, że odróżnia ciągi kodowe od ciągów niekodowych. Ciągi niekodowe umożliwiają dekoderowi detekcję błędów transmisyjnych. Ale dekoder, analizując ciąg odebrany, może również znaleźć ciąg kodowy, różniący się od ciągu odebranego najmniejszą liczbą pozycji, i przyjąć, że ten ciąg został właśnie nadany. Działanie to jest równoważne lokalizacji i korekcji błędów. Taka strategia dekodowania nazywa się *dekodowaniem z maksymalną wiarygodnością* (maximum likelihood decoding) i jest powszechnie stosowana w praktyce.

Omawiane wyżej zbiory: wektorów kodowych, wektorów niekodowych i przestrzeń wektorowa mają odzwierciedlenie w strukturach algebraicznych. Pokazano to w tabeli 2.2.1. Kod liniowy jest podgrupą, a kod cykliczny ideałem. Pojęcia grupy addytywnej, podgrupy i pierścienia opisano w p. 1.1.1.

Ideał jest podzbiorem pierścienia. Podzbiór S pierścienia R nazywa się podpierścieniem, jeśli jest zamknięty względem operacji dodawania i mnożenia oraz jest pierścieniem względem tych operacji. Podzbiór J pierścienia R nazywa się *ideałem* (dwu-

stronnym) tego pierścienia, jeżeli J jest podpierścieniem pierścienia R i dla wszystkich $a \in J$ i $r \in R$ zachodzi $a\,r \in J$ i $r\,a \in J$.

Tabela 2.2.1. Struktury algebraiczne stosowane w kodach blokowych

Zbiór	Liczba elementów	Kod liniowy	Kod cykliczny
Przestrzeń wektorowa	$2^n, \left(q^n\right)$	grupa addytywna, ciało	pierścień, ciało
Wektory kodowe	$2^k, \left(q^k\right)$	podgrupa	ideał
Wektory niekodowe	$2^n - 2^k, \left(q^n - q^k\right)$	warstwy	klasy reszt

W teorii blokowych kodów cyklicznych wykorzystuje się pojęcie pierścienia wielomianów. *Pierścieniem wielomianów* modulo wielomian stopnia n jest zbiór wielomianów, stopnia nie większego od $n-1$, z określonymi operacjami dodawania i mnożenia.

Dla dowolnej pary wielomianów $a(x)$ i $b(x)$ istnieje para wielomianów $q(x)$ i $r(x)$ spełniająca zależność

$$a(x) = b(x)\,q(x) + r(x), \tag{2.2.3}$$

gdzie stopień wielomianu $r(x)$ jest mniejszy od stopnia wielomianu $b(x)$.

Powyższa zależność jest znana jako *algorytm dzielenia Euklidesa* i można ją zapisać w postaci kongruencji

$$r_i(x) \equiv a_i(x) \left(\mathrm{mod}\; b(x)\right). \tag{2.2.4}$$

Wielomiany $a_i(x)$, które mają tą samą resztę $r_i(x)$ otrzymaną z dzielenia przez wielomian $b(x)$, nazywamy i-tą klasą reszt modulo wielomian $b(x)$. Nad ciałem binarnym różnych klas reszt może być 2^n, gdzie n jest stopniem wielomianu $b(x)$. Na klasach reszt można zdefiniować operacje dodawania i mnożenia i utworzyć pierścień klas reszt modulo wielomian $b(x)$.

Teoria kodów cyklicznych opiera się na strukturze ciał rozszerzonych opisanej w p. 1.5.8, a wielomian $b(x)$ jest w tym przypadku dwumianem $x^n - 1$. Faktoryzacja tego dwumianu umożliwia obliczenie wielomianów generujących kody cykliczne, które oznaczamy przez $g(x)$.

Zbiór wszystkich możliwych wielokrotności wielomianu $g(x)$, będącego podzielnikiem dwumianu $x^n - 1$, jest ideałem, a wielomian $g(x)$ nazywamy wielomianem generującym ideał. Kody cykliczne z matematycznego punktu widzenia są ideałami.

Pierścień wielomianów modulo $x^n - 1$ można rozłożyć na warstwy względem ideału. W wyniku tego rozkładu powstają klasy reszt modulo $g(x)$.

2.2.3. Zdolność detekcyjna i korekcyjna kodu

W punkcie tym sformułowano podstawowe pojęcia umożliwiające zdefiniowanie zdolności detekcyjnej i korekcyjnej kodu liniowego. Do tego celu służą parametry określające odległość wektorów kodowych i ich wagę.

Odległość Hamminga $d_H(u,v)$ między dwoma wektorami kodowymi u i v jest liczbą pozycji, na których występują różne współrzędne w wektorach. Ilustruje to przykład

$$u = [1001011100101],$$

$$v = [1100010110111],$$

$$d_H(u,v) = 5.$$

Waga Hamminga $w(u)$ wektora kodowego u jest liczbą niezerowych współrzędnych wektora. Dla powyższego wektora u waga Hamminga wynosi: $w(u) = 7$.

Odległość między wektorami jest równa wadze sumy wektorów, to jest

$$d_H(u,v) = w(u+v).$$

Dla podanych wyżej wektorów będzie:

$$u+v = [0101001010010],$$

$$d_H(u,v) = 5, \quad w(u+v) = 5.$$

Również można zauważyć, że waga wektora kodowego jest równa odległości od wektora zerowego.

Odległość Hamminga i wagę Hamminga dla kodów nad ciałami rozszerzonymi oblicza się tak samo jak dla kodów binarnych.

W teorii kodów ważną rolę odgrywa *odległość minimalna* między wektorami kodowymi, oznaczana przez d. Odległość minimalna decyduje o możliwości detekcji i korekcji błędów kodu blokowego.

P r z y k ł a d 2.2.1.

Kod z bitem parzystości.

Najprostszym kodem detekcyjnym jest kod z kontrolą parzystości. Do ciągów informacyjnych można dodawać jeden lub więcej bitów parzystości. Kod z jednym bitem parzystości $(n, n-1)$ ma odległość minimalną $d = 2$ i umożliwia wykrycie jednego błędu oraz wszystkich błędów nieparzystych. Dodanie dodatkowego bitu parzystości znowu zwiększa odległość minimalną o jeden. Kody z kontrolą parzystości są powszechnie stosowane do kontroli poprawności danych w komputerach.

Kody mogą być używane do wykrywania błędów transmisyjnych lub do wykrywania i korygowania błędów. Zdolność detekcyjną kodu l określa zależność

$$l = d - 1. \tag{2.2.5}$$

Kod blokowy o określonym parametrze d może wykryć wszystkie ciągi błędów o wadze $d-1$ lub mniejszej.

Możliwości korekcji błędów określa zdolność korekcyjna kodu t:

$$t = E\left[\frac{d-1}{2}\right], \tag{2.2.6}$$

gdzie funkcja E oznacza największą liczbę całkowitą nie większą od $(d-1)/2$. Dla zadanej zdolności korekcyjnej kodu t jego odległość minimalna powinna wynosić

$$d \geq 2t + 1. \tag{2.2.7}$$

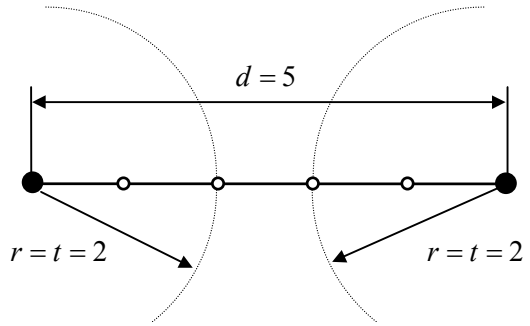
W przypadku, gdy kod jest używany jednocześnie do detekcji a błędów i korekcji b błędów, jego odległość minimalna powinna wynosić

$$d \geq a + b + 1, \tag{2.2.8}$$

gdzie $a \geq b$. Na przykład kod posiadający $d = 7$ może jednocześnie wykryć i skorygować liczby błędów pokazane w kolumnach poniższej tabeli.

a	3	4	5	6
b	3	2	1	0

Zdolność korekcyjną kodu liniowego można zinterpretować graficznie w sposób pokazany na rys. 2.2.3. Na tym rysunku pokazano dwa wektory kodowe w postaci czarnych punktów, między którymi odległość minimalna $d = 5$. Zdolność korekcyjna takiego kodu $t = 2$. Jeśli kod koryguje wszystkie błędy ≤ 2, oznacza to, że punkty odpowiadające wektorom kodowym można otoczyć w przestrzeni wielowymiarowej „nie przecinającymi się kulami" o promieniu $r = 2$. Wektory powstające w wyniku błędów, nie przekraczających zdolności korekcyjnej kodu, nie pojawią się na zewnątrz tych kul, co umożliwia przyporządkowanie ich właściwym wektorom kodowym.



Rys. 2.2.3. Interpretacja korekcyjnych właściwości kodu

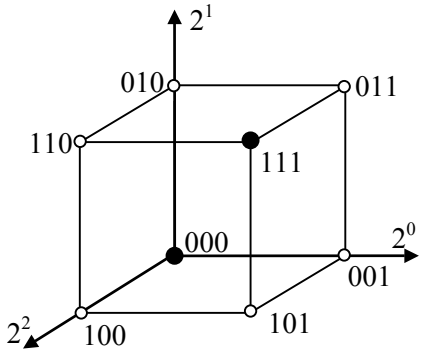
2.2.4. Geometryczna interpretacja kodu

Geometryczne przedstawienie kodu umożliwia interpretację pojęć używanych do jego opisu. Aby przedstawić zbiór wszystkich n-pozycyjnych ciągów binarnych, można zastosować n-wymiarową, dyskretną przestrzeń liniową nad $GF(2)$. Zbiór ciągów w takiej przestrzeni można sobie wyobrazić jako wierzchołki wielościanu foremnego o krawędziach mających długość jeden. Wektorom kodowym będą wówczas odpowiadać odcinki łączące wierzchołki z początkiem układu współrzędnych i skierowane do wierzchołków wielokąta. Jednak graficzne przestawienie takich przestrzeni jest możliwe tyko dla niewielkich wartości n.

P r z y k ł a d 2.2.2.

Geometryczne przedstawienie kodu $(3,1)$.

Kod $(3,1)$ posiada dwa wektory kodowe: $u=[000]$ i $v=[111]$. Ma on jeden bit informacyjny i dwa bity kontrolne. Odległość minimalna kodu $d=3$, jego zdolność detekcyjna $l=2$, a zdolność korekcyjna $t=1$. Kod ten może wykryć jeden lub dwa błędy albo skorygować jeden błąd. Kod możemy przedstawić w postaci sześcianu foremnego, pokazanego na rys. 2.2.4. Na tym rysunku końce wektorów kodowych są zaznaczone ciemnymi punktami, a wektorów nie należących do kodu punktami jasnymi.



Rys. 2.2.4. Graficzne przedstawienie kodu (3,1)

W dyskretnej przestrzeni liniowej odległość między wektorami interpretuje się jako liczbę krawędzi dzielących wierzchołki odpowiadające wektorom kodowym. Z rysunku widać, że dla kodu $(3,1)$ odległość minimalna $d=3$.

Wektory niekodowe powstają w wyniku błędów transmisyjnych. W przypadku jednokrotnego błędu wektor kodowy [000] może zmienić się w jeden z wektorów: [100], [010] lub [001]. Podobnie wektor kodowy [111] może zmienić się w jeden z wektorów: [011], [101] lub [110]. Podobnie można określić, jakie wektory otrzymamy, kiedy wystąpią błędy wielokrotne.

Błędy pojedyncze pojawiają się zwykle z największym prawdopodobieństwem. Stosując dekodowanie z największą wiarygodnością, dekoder wykona następujące

przyporządkowania: wektory odebrane [100], [010] i [001] zostaną zinterpretowane jako wektor kodowy [000], a wektory odebrane [011], [101] i [110] zostaną zinterpretowane jako wektor [111]. W ten sposób będzie skorygowany pojedynczy błąd.

2.2.5. Syndrom

Dekoder kodu blokowego dokonuje estymacji wektorów kodowych na podstawie wektorów odebranych na wyjściu kanału transmisyjnego. W algorytmach dekodowania wykorzystuje się *syndrom błędów*, zwany też po prostu syndromem. Syndrom błędów jest tak skonstruowany, że jego postać zależy od lokalizacji błędów, a nie od wektora wejściowego dekodera.

Każdy kod binarny (n, k) ma $n-k$ pozycji kontrolnych, które umożliwiają kontrolę błędów. Można zatem utworzyć zbiór 2^{n-k} syndromów razem z syndromem zerowym, z których każdy będzie odpowiadał określonej kombinacji błędów. Z drugiej strony każdy n-elementowy wektor kodowy może zawierać $\binom{n}{i}$ różnych kombinacji błędnych, gdzie i jest liczbą błędów w wektorze kodowym. Jeśli uwzględnimy wszystkie kombinacje błędów od 0 do t, to całkowita liczba kombinacji błędnych będzie

$$\sum_{i=0}^{t}\binom{n}{i}, \quad \binom{n}{i}=\frac{n!}{(n-i)!\cdot i!},$$

gdzie t jest maksymalną liczbą błędów.

Jeśli kod blokowy (n, k) może korygować wszystkie kombinacje błędów nie zawierające więcej niż t błędów, to liczba syndromów nie może być mniejsza niż całkowita liczba kombinacji błędnych

$$2^{n-k} \geq \sum_{i=0}^{t}\binom{n}{i}. \tag{2.2.9}$$

Wyrażenie powyższe może służyć do oceny jakości kodu i nosi nazwę *granicy Hamminga*. Kody binarne, dla których zależność ta jest spełniona ze znakiem równości, nazywają się *kodami doskonałymi* (perfect codes). Kod doskonały ma więc liczbę syndromów taką samą jak liczbę kombinacji błędnych zawierających t lub mniej błędów. Z zależności tej można również obliczyć maksymalną liczbę pozycji kontrolnych k dla kodu blokowego o długości wektora kodowego n i zdolności korekcyjnej t.

Do oceny kodów stosuje się również pojęcia: *kod optymalny* i *kod dobry*. Podczas oceny kodu bierze się pod uwagę liczbę elementów informacyjnych przy ustalonej długości wektora kodowego i odległości minimalnej. Kod jest tym lepszy, im więcej elementów informacyjnych zawiera wektor kodowy, gdyż wtedy może przekazywać więcej informacji.

2.3. Kody liniowe

2.3.1. Definicja kodu liniowego

Blokowym kodem liniowym (n,k) nazywamy kod opisany układem $n-k$ liniowo niezależnych równań:

$$\sum_{i=1}^{n} b_{ij}\, a_i = 0, \quad \text{gdzie} \quad j = 1, 2, \ldots, n-k. \tag{2.3.1}$$

Współczynniki b_{ij} przyjmują wartości 0 lub1, a_i zaś są elementami wektora kodowego. Można pokazać, że zbiór ciągów kodowych takiego kodu jest podgrupą grupy addytywnej tworzącej przestrzeń wektorową nad $GF(2)$. Tak zdefiniowany kod liniowy jest pod względem konstrukcji kodem prostym, gdyż wykorzystuje on tylko jedno działanie dodawanie.

Blokowy kod liniowy można również opisać za pomocą macierzy generującej. Wówczas w procesie kodowania i dekodowania występuje mnożenie, a kod jest podprzestrzenią przestrzeni liniowej nad ciałem skończonym. Kod liniowy można wtedy zdefiniować inaczej. Zbiór wektorów n-elementowych jest kodem liniowym wtedy i tylko wtedy, gdy stanowi on podprzestrzeń przestrzeni wektorowej zawierającej wszystkie możliwe wektory n-elementowe.

Kody liniowe mogą być rozdzielne i nierozdzielne. W niniejszym podręczniku ograniczymy się do kodów binarnych, gdyż te są najczęściej stosowane w praktyce. Aby zrozumieć konstrukcję i właściwości kodów liniowych, prześledźmy prosty przykład.

P r z y k ł a d 2.3.1.

Generowanie liniowego kodu blokowego (5,2).

Właściwości kodu opiszemy na przykładzie liniowego kodu rozdzielnego $(n,k) = (5,2)$. Wektor tego kodu $\left[a_1\, a_2\, a_3\, a_4\, a_5\right]$ zawiera dwa bity informacji: a_1 i a_2 oraz trzy bity kontrolne: a_3, a_4 i a_5. Bity kontrolne kodu kontrolują poprawność (parzystość) ciągu informacyjnego i obliczamy je podczas kodowania informacji za pomocą równań kontrolnych. Przyjmujemy, że kod ten jest określony następującym układem liniowo niezależnych równań kontrolnych:

$$\begin{aligned} a_3 &= a_1, \\ a_4 &= a_2, \\ a_5 &= a_1 + a_2. \end{aligned} \tag{2.3.2}$$

Na razie pomijamy sposób wyznaczania równań kontrolnych. Metoda ich obliczania jest różna dla różnych typów kodów.

Aby wyznaczyć zbiór wszystkich wektorów kodowych, należy podstawić za bity informacyjne a_1 i a_2 wszystkie kombinacje dwóch symboli binarnych i obliczyć

z (2.3.2) elementy kontrolne wektorów kodowych. Obliczone wektory dla kolejnych ciągów informacyjnych będą miały postać przedstawioną w poniższej tabelce.

Informacja	00	01	10	11
Wektor kodowy	00000	01011	10101	11110

Otrzymany zbiór kodowy spełnia właściwości grupy addytywnej. Jest on podgrupą grupy addytywnej utworzonej ze wszystkich wektorów 5-elementowych nad $GF(2)$. Cała grupa będzie zawierała $2^5=32$ wektory. Grupę tę można rozłożyć na warstwy względem podgrupy kodowej. Rozkład taki pokazano w tabeli 2.3.1, która nazywa się *tablicą standardową kodu liniowego*. Wektory kodowe znajdują się w drugim wierszu i są zaznaczone pogrubioną czcionką. Za element tworzący warstwę można przyjąć każdy ciąg nie występujący w wyższych warstwach. W zastosowaniach kodowych za element tworzący warstwę przyjmuje się najbardziej prawdopodobne wektory błędów. Na pierwszym miejscu będą to błędy pojedyncze.

Tabela 2.3.1. Standardowa tablica kodu liniowego (5,2)

Informacje	00	01	10	11
Wektory kodowe	**00000**	**01011**	**10101**	**11110**
Warstwy	***10000***	11011	00101	01110
	01000	00011	11101	10110
	00100	01111	10001	11010
	00010	01001	10111	11100
	00001	01010	10100	11111
	11000	10011	01101	00110
	01100	00111	11001	10010

Elementy tworzące warstwy, odpowiadające błędom transmisyjnym, znajdują się w drugiej kolumnie i są wydrukowane pogrubioną kursywą. Każda warstwa zawiera wektory, które powstały w wyniku sumowania elementów tworzących warstwę z wektorami kodowymi. Odzwierciedlają one ciągi odebrane powstałe w wyniku działania błędów na wektory kodowe w kanale transmisyjnym.

W tablicy standardowej brak elementów powtarzających się, a elementy znajdujące się w tym samym wierszu mają jednakowe syndromy. Tablica ta może służyć do dekodowania korekcyjnego. W tym celu wystarczy stwierdzić, w której kolumnie jest wektor odebrany, i przyporządkować mu odpowiedni ciąg kodowy, znajdujący się w drugim wierszu tabeli.

Odległość minimalną kodu liniowego można wyznaczyć, badając wagi wektorów kodowych. Odległość minimalna kodu liniowego d jest równa najmniejszej wadze

Hamminga niezerowych wektorów kodowych. Na przykład dla kodu pokazanego w tabeli 2.3.1 wagi niezerowych wektorów kodowych wynoszą: 3, 3 i 4. Zatem odległość minimalna tego kodu jest $d = 3$.

2.3.2. Macierzowy opis kodu liniowego

Liniowy kod blokowy (n, k) jest *k*-wymiarową podprzestrzenią przestrzeni wektorowej *n*-wymiarowej. Można zatem znaleźć zbiór *k* liniowo niezależnych wektorów, które wygenerują zbiór wszystkich wektorów kodowych. Zbiór ten nazywamy *bazą przestrzeni kodowej*. Kod liniowy (n, k) może być jednoznacznie określony przez dowolny zbiór *k* liniowo niezależnych wektorów, stanowiących bazę przestrzeni kodowej. Baza ta jest podstawą do konstrukcji macierzy generującej kod $\boldsymbol{G}$.

Macierz generująca (generator matrix) $\boldsymbol{G}$ o wymiarach $k \times n$ zawiera *k* wierszy i *n* kolumn. Kod liniowy jest zbiorem wszystkich liniowych kombinacji wierszy macierzy $\boldsymbol{G}$. Zbiór ten nazywa się *przestrzenią wierszową* (row space) macierzy. Wymiar przestrzeni wierszowej nosi nazwę *rzędu wierszowego* macierzy (row rank).

Macierz $\boldsymbol{G}$ jest używana do kodowania informacji. Wyznaczamy ją w następujący sposób. Współczynniki prawych stron równań kontrolnych (2.3.1) zapisujemy w postaci macierzy współczynników $\boldsymbol{P}$ o wymiarach $k \times (n-k)$, gdzie *k* jest liczbą wierszy, a $n-k$ liczbą kolumn macierzy. Element p_{ij} oznacza współczynnik *i* równania *j*.

$$\boldsymbol{P}_{k,n-k} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1,n-k} \\ p_{21} & p_{22} & \cdots & p_{2,n-k} \\ \cdots & \cdots & \cdots & \cdots \\ p_{k1} & p_{k2} & \cdots & p_{k,n-k} \end{bmatrix}. \tag{2.3.3}$$

Z macierzy $\boldsymbol{P}$ wyznaczamy macierz generująca $\boldsymbol{G}$ o wymiarach $k \times n$. Macierz generująca może mieć różną postać. Macierz tę zwykle przedstawiamy w postaci kanonicznej:

$$\boldsymbol{G}_{k,n} = \left[\boldsymbol{I}_k \cdot \boldsymbol{P}_{k,n-k} \right], \tag{2.3.4}$$

$$\boldsymbol{G}_{k,n} = \begin{bmatrix} \boldsymbol{g_1} \\ \boldsymbol{g_2} \\ \cdots \\ \boldsymbol{g_k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{11} & p_{12} & \cdots & p_{1,n-k} \\ 0 & 1 & \cdots & 0 & p_{21} & p_{22} & \cdots & p_{2,n-k} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & p_{k1} & p_{k2} & \cdots & p_{k,n-k} \end{bmatrix}, \tag{2.3.5}$$

gdzie $\boldsymbol{I}_k$ jest macierzą jednostkową stopnia *k*.

Macierz można zapisać w postaci blokowej lub tablicy. Blokowa postać macierzy pokazana w (2.3.4) nazywa się postacią bazową. Wzór (2.3.5) przedstawia macierz

w formie blokowej za pomocą wektorów, odpowiadających wierszom macierzy, oraz w postaci tablicy.

Dowolną macierz $\boldsymbol{G}$ nad ciałem binarnym da się sprowadzić do postaci kanonicznej za pomocą elementarnych operacji wierszowych:

1. Zamiana wierszy miejscami.
2. Dodanie wiersza do dowolnego innego wiersza.

Operacje te nie zmieniają przestrzeni wierszowej macierzy i nazywamy je niezmienniczymi względem zbioru ciągów kodowych.

Wiersze macierzy generującej $\boldsymbol{G}$ określają zbiór wektorów bazowych kodu. Każdy wektor kodowy jest liniową kombinacją tych wektorów bazowych. Zapis macierzowy kodu jest bardziej zwięzły niż lista wektorów kodowych. Na przykład binarny kod liniowy (50,30), mający ponad 10^9 wektorów, można opisać macierzą o wymiarach 30×50.

Dla każdej macierzy generującej $\boldsymbol{G}$ o wymiarach $k \times n$ istnieje *macierz kontrolna* (parity check matrix) $\boldsymbol{H}$ o wymiarach $(n-k) \times n$ taka, że wiersze macierzy $\boldsymbol{G}$ są ortogonalne do wierszy macierzy $\boldsymbol{H}$, to jest $\boldsymbol{G} \cdot \boldsymbol{H}^T = 0$, gdzie $\boldsymbol{H}^T$ jest transponowaną lub przestawioną macierzą kontrolną $\boldsymbol{H}$.

$$\boldsymbol{H}_{n-k,n} = \left[\boldsymbol{P^T}_{n-k,k} \cdot \boldsymbol{I}_{n-k} \right],$$

$$\boldsymbol{H}_{n-k,n} = \begin{bmatrix} \boldsymbol{h_1} \\ \boldsymbol{h_2} \\ \dots \\ \boldsymbol{h_{n-k}} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{21} & \dots & p_{k1} & 1 & 0 & \dots & 0 \\ p_{12} & p_{22} & \dots & p_{k2} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ p_{1,n-k} & p_{2,n-k} & \dots & p_{k,n-k} & 0 & 0 & \dots & 1 \end{bmatrix}. \qquad (2.3.6)$$

Do obliczania syndromu wektora odebranego w procesie dekodowania służy macierz transponowana $\boldsymbol{H^T}$. Macierz transponowana $\boldsymbol{H}^T$ ma wymiary $n \times (n-k)$, a jej wiersze są kolumnami macierzy $\boldsymbol{H}$. Macierz ta ma postać:

$$\boldsymbol{H^T}_{n,n-k} = \begin{bmatrix} \boldsymbol{P}_{k,n-k} \\ \boldsymbol{I}_{n-k} \end{bmatrix}, \quad \boldsymbol{H^T}_{n,n-k} = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1,n-k} \\ p_{21} & p_{22} & \dots & p_{2,n-k} \\ \dots & \dots & \dots & \dots \\ p_{k,1} & p_{k,2} & \dots & p_{k,n-k,} \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}. \qquad (2.3.7)$$

Równania kontrolne kodu liniowego są bezpośrednio związane z macierzą kontrolną. Widać to, jeśli porówna się pierwsze dwa elementy wierszy macierzy $\boldsymbol{H}$ z przykładu 2.3.2 i równania kontrolne z przykładu 2.3.1.

P r z y k ł a d 2.3.2.
Macierzowa postać kodu liniowego (5,2) z przykładu 2.3.1.
Dla kodu (5,2) z przykładu 2.3.1 otrzymamy następującą macierz $\boldsymbol{P}$:

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Z macierzy $\boldsymbol{P}$ wyznaczamy kolejno macierz generującą kodu $\boldsymbol{G}$, kontrolną $\boldsymbol{H}$ i macierz transponowaną $\boldsymbol{H^T}$:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{H^T} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

2.3.3. Kodowanie informacji

Kodowanie informacji polega na liniowym odwzorowaniu ciągu informacyjnego w ciąg kodowy

$$\left[a_1, a_2, \ldots, a_k\right] \rightarrow \left[a_1, a_2, \ldots, a_k, a_{k+1}, \ldots, a_n\right].$$

Kodowanie można wykonać za pomocą równań kontrolnych lub za pomocą macierzy generującej $\boldsymbol{G}$. Sposób kodowania z zastosowaniem równań kontrolnych pokazano w p. 2.3.1. Obecnie przedstawimy kodowanie z zastosowaniem macierzy generującej.

W przypadku zastosowania macierzy generującej $\boldsymbol{G}$ wektor kodowy na wyjściu kodera $\boldsymbol{c_X}$ obliczamy z zależności

$$\boldsymbol{c}_X = \boldsymbol{m} \cdot \boldsymbol{G} = \left[m_1, m_2, \ldots, m_k\right] \cdot \begin{bmatrix} \boldsymbol{g_1} \\ \boldsymbol{g_2} \\ \ldots \\ \boldsymbol{g_k} \end{bmatrix} = m_1\boldsymbol{g_1} + m_2\boldsymbol{g_2} + \ldots + m_k\boldsymbol{g_k}, \tag{2.3.8}$$

gdzie $\boldsymbol{m}$ jest nadawanym ciągiem informacyjnym, a $\boldsymbol{g}_i$ są wierszami macierzy generującej $\boldsymbol{G}$.

P r z y k ł a d 2.3.3.
Obliczanie wektora kodowego kodu (5,2) z przykładu 2.3.2.

Dla kodu (5,2) i ciągu informacyjnego $m = 10$ z (2.3.8) otrzymamy następujący wektor kodowy:

$$\boldsymbol{c}_X = \boldsymbol{m}\boldsymbol{G} = [10] \cdot \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} = 1 \cdot [10101] + 0 \cdot [01011] = [10101].$$

2.3.4. Dekodowanie ciągów odebranych

Procesy zachodzące w kanale i zbiór wektorów na wejściu dekodera opisano w p. 2.2.2. Zadaniem dekodera jest usunięcie błędów transmisyjnych. Sposób dekodowania odebranych wektorów kodowych zależy od przeznaczenia kodu. Rozróżniamy dekodowanie detekcyjne i korekcyjne. W procesie dekodowania wykorzystuje się syndrom wektora odebranego lub macierz $\boldsymbol{H}$.

W przypadku kodów liniowych elementy syndromu wektora odebranego oblicza się z zależności

$$s_j = \sum_{i=1}^{n} b_{ij}\, a_i, \quad \text{gdzie} \quad j = 1, 2, \ldots, n-k. \tag{2.3.9}$$

Wzór ten jest podobny do wzoru (2.3.1). Gdy wektor odebrany nie ma błędów, wówczas syndrom jest równy zero.

Jeśli kod ma wykrywać błędy, to dekodowanie sprowadza się do obliczenia syndromu i stwierdzenia, czy syndrom jest zerowy, czy też nie. Zerowy syndrom oznacza, że nie wystąpiły błędy wykrywalne przez kod a syndrom niezerowy, że wystąpiły błędy transmisyjne.

W przypadku kodu korekcyjnego w procesie dekodowania należy wykryć błędy, a następnie je zlokalizować i skorygować. Dekodowanie można wykonać, posługując się standardową tablicą lub syndromem wektora odebranego. Tablicę standardową opisano w p. 2.3.1. Używanie tablicy kodu jest niepraktyczne w przypadku dużych kodów.

Dekodowanie za pomocą syndromu wymaga następujących czynności:

1. Obliczenia syndromu wektora odebranego.
2. Wyznaczenia wektora błędów związanego z obliczonym syndromem.
3. Korekcji błędów wektora odebranego.

Taka metoda dekodowania wymaga użycia tablicy dekodowania zawierającej syndromy i odpowiadające im ciągi błędów. Znane są również inne metody dekodowania korekcyjnego, jak dekodowanie wykorzystujące wagi wektorów odebranych (step-by-step decoding) [2.4] lub wspomniana już metoda macierzowa.

Jeśli mamy zlokalizowane elementy błędne, to można wykonać korekcję wektora odebranego. Aby wyznaczyć wektor na wyjściu dekodera $\boldsymbol{c_D}$, należy do wektora odebranego $\boldsymbol{c_Y}$ dodać wektor błędów $\boldsymbol{e}$

$$\boldsymbol{c}_D = \boldsymbol{c}_Y + \boldsymbol{e}. \tag{2.3.10}$$

P r z y k ł a d 2.3.4.

Dekodowanie kodu liniowego (5,2) z przykładu 2.3.3.

Aby pokazać dekodowaniem z zastosowaniem syndromu, posłużymy się kodem opisanym w przykładzie 2.3.1. Na początku wyznaczmy wzory do obliczania elementów syndromu. Otrzymujemy je z (2.3.9), stosując równania (2.3.2):

$$\begin{aligned} s_1 &= a_1 + a_3, \\ s_2 &= a_2 + a_4, \\ s_3 &= a_1 + a_2 + a_5. \end{aligned} \tag{2.3.11}$$

Aby przeprowadzić proces dekodowania, do tabeli 2.3.1 dodajemy kolumnę zawierającą wartości syndromów. Po tej modyfikacji otrzymujemy tablicę dekodowania pokazaną w tabeli 2.3.2.

Tabela 2.3.2. Tablica dekodowania kodu liniowego (5,2)

Informacje	00	01	10	11	Syndromy $s_1\ s_2\ s_3$
Wektory kodowe	**00000**	**01011**	**10101**	**11110**	000
Warstwy	***10000***	11011	00101	01110	101
	01000	00011	11101	10110	011
	00100	01111	10001	11010	100
	00010	01001	10111	11100	010
	00001	01010	10100	11111	001
	11000	10011	01101	00110	110
	01100	00111	11001	10010	111

Ponieważ syndrom zależy od wektora błędu, a nie od wektora kodowego, to dla wszystkich elementów warstwy syndrom ma taką samą wartość. Ta właściwość syndromu umożliwia nam znalezienie wektora błędów bezpośrednio po odnalezieniu warstwy, w której leży wektor odebrany. Wtedy wektorem błędów jest po prostu ten element tworzący warstwę, który ma ten sam syndrom co wektor odebrany. W tabeli 2.3.2 elementy tworzące warstwę znajdują się w drugiej kolumnie i są wydrukowane pogrubioną kursywą.

Jeśli wyznaczymy wektor błędów, to skorygowany wektor kodowy obliczamy z zależności (2.3.10). Załóżmy, że na wejściu dekodera podano wektor $\boldsymbol{c}_O = [a_1\ a_2\ a_3\ a_4\ a_5] = [11010]$. Syndrom dla tego wektora obliczony z (2.3.11) będzie $\boldsymbol{s} = [s_1\ s_2\ s_3] = [1\ 0\ 0]$, a element tworzący warstwę i odpowiadający obliczonemu syndromowi jest równy [0 0 1 0 0]. Po korekcji otrzymamy wektor odpowiadający wektorowi nadanemu [11010]+[00100]=[11110]. W czasie dekodowania wyko-

rzystuje się bezpośrednio kolumny drugą i ostatnią tabeli 2.3.2. Pozostałe kolumny w praktyce pomija się.

Innym sposobem wyznaczania syndromu jest metoda macierzowa. Korzystamy w tym celu z macierzy transponowanej $\boldsymbol{H}^{\boldsymbol{T}}$, a syndrom obliczamy ze wzoru

$$\boldsymbol{s} = \boldsymbol{c}_Y \boldsymbol{H}^{\boldsymbol{T}}, \tag{2.3.12}$$

gdzie $\boldsymbol{c}_{\boldsymbol{Y}}$ jest wektorem na wejściu dekodera. Po podstawieniu (2.2.2) otrzymamy

$$\boldsymbol{s} = \left(\boldsymbol{c}_X + \boldsymbol{e}\right)\boldsymbol{H}^{\boldsymbol{T}} = \boldsymbol{c}_X \boldsymbol{H}^{\boldsymbol{T}} + \boldsymbol{e}\boldsymbol{H}^{\boldsymbol{T}} = \boldsymbol{m}\boldsymbol{G}\boldsymbol{H}^{\boldsymbol{T}} + \boldsymbol{e}\boldsymbol{H}^{\boldsymbol{T}} = \boldsymbol{e}\boldsymbol{H}^{\boldsymbol{T}}. \tag{2.3.13}$$

Ponieważ $\boldsymbol{m}\boldsymbol{G}\boldsymbol{H}^{\boldsymbol{T}} = 0,$ wartość syndromu wektora odebranego będzie równa syndromowi wektora błędu. Po podstawienie współrzędnych wektora błędów i macierzy transponowanej otrzymamy

$$\boldsymbol{s} = \boldsymbol{e}\boldsymbol{H}^{T} = \left[e_1\ e_2 \ldots e_n\right] \cdot \begin{bmatrix} \boldsymbol{h}_1^T \\ \boldsymbol{h}_2^T \\ \cdots \\ \boldsymbol{h}_n^T \end{bmatrix} = e_1 \boldsymbol{h}_1^T + e_2 \boldsymbol{h}_2^T + \ldots + e_n \boldsymbol{h}_n^T. \tag{2.3.14}$$

Syndrom wektora odebranego jest równy sumie wierszy transponowanej macierzy kontrolnej odpowiadającej pozycjom błędów.

P r z y k ł a d 2.3.5.
Dekodowanie wektora odebranego kodu liniowego (5,2) metodą macierzową.
Obliczmy syndrom dla wektora odebranego $\boldsymbol{c}_Y = [c_1\ c_2\ c_3\ c_4\ c_5] = [1\,1\,0\,1\,0]$

$$\boldsymbol{s} = \boldsymbol{c}_Y \boldsymbol{H}^{\boldsymbol{T}} = \left[1\,1\,0\,1\,0\right] \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \left[1\,0\,1\right] + \left[0\,1\,1\right] + \left[0\,1\,0\right] = \left[1\,0\,0\right].$$

Jeśli mamy wektor odebrany $\boldsymbol{c}_{\boldsymbol{Y}}$ i wektor błędów $\boldsymbol{e}$, możemy wykonać korekcję wektora odebranego. Wektor skorygowany obliczamy z (2.3.10)

$$\boldsymbol{c}_D = \boldsymbol{c}_Y + \boldsymbol{e} = [1\,1\,0\,1\,0\ ] + [0\,0\,1\,0\,0] = [1\,1\,1\,1\,0].$$

2.3.5. Liniowe kody Hamminga

Liniowe kody Hamminga są kodami korekcyjnymi rozdzielnymi nad $GF(2)$. Kody te mają odległość minimalną $d = 3$ i mogą korygować pojedyncze błędy. Kody Hamminga są kodami doskonałymi i są one optymalne dla binarnych kanałów symetrycznych.

Długość wektora kodowego i liczba pozycji informacyjnych kodu wynoszą $(n,k) = (2^m - 1, 2^m - m - 1)$, gdzie m jest liczbą pozycji kontrolnych. Parametry kilku liniowych kodów Hamminga podano w tabeli 2.3.3.

Tabela 2.3.3. Parametry liniowych kodów Hamminga

m	2	3	4	5	6
(n, k)	(3,1)	(7,4)	(15,11)	(31,26)	(63,57)

Konstrukcja liniowego kodu Hamminga pokażemy na przykładzie kodu (7,4). Aby skonstruować kod, należy wyznaczyć układ równań kontrolnych. W tym celu zapisujemy wszystkie możliwe pozycje błędów w postaci liczb binarnych, co pokazano w tabeli 2.3.4.

Tabela 2.3.4. Sekwencje kontrolne pozycji błędnych

Pozycja błędu	Sekwencja kontrolna
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Jako pozycje kontrolne przyjmujemy pozycje z sekwencjami kontrolnymi zawierającymi ciągi niezależne. Są to pozycje: 1, 2 i 4. Struktura wektora kodowego będzie miała postać pokazaną w tabeli 2.3.5. W pierwszym wierszu znajdują się symbole współrzędnych wektora kodowego, a w drugim – typ współrzędnej, gdzie r_i oznacza elementy kontrolne, a m_i – elementy informacji.

Tabela 2.3.5. Wektor kodowy liniowego kodu Hamminga (7,4)

a_1	a_2	a_3	a_4	a_5	a_6	a_7
r_1	r_2	m_1	r_3	m_2	m_3	m_4

Z tabeli 2.3.4 widać, że element kontrolny r_1 kontroluje parzystość elementów wektora kodowego: a_1, a_3, a_5 i a_7, na podstawie czego możemy napisać równanie kontrolne dla elementu a_1. Podobnie zapisując pozostałe równania kontrolne, otrzymamy:

$$\begin{aligned} a_1 &= a_3 + a_5 + a_7, \\ a_2 &= a_3 + a_6 + a_7, \\ a_4 &= a_5 + a_6 + a_7. \end{aligned} \tag{2.3.15}$$

Kodowanie

W trakcie kodowania informacji obliczamy wektory kodowe dla wszystkich możliwych ciągów informacyjnych korzystając z równań kontrolnych (2.3.15). Na przykład dla ciągu informacyjnego $\left[a_3\ a_5\ a_6\ a_7\right]=\left[1\,0\,1\,0\right]$ otrzymamy następujące bity kontrolne:

$$a_1 = 1,\ a_2 = 0,\ a_4 = 1.$$

Wektor kodowy będzie miał postać

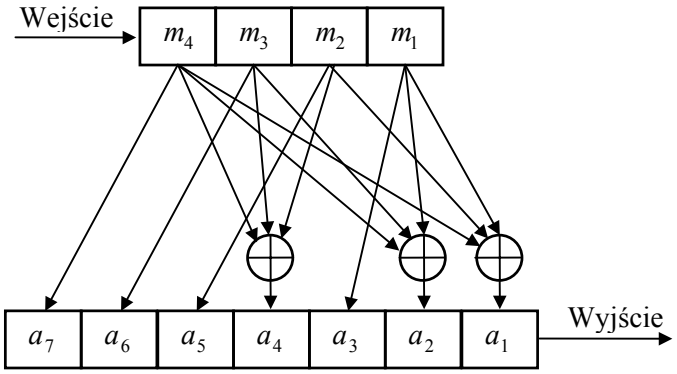
$$\left[a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\right]=\left[1\,0\,1\,1\,0\,1\,0\right].$$

Wszystkie wektory kodowe kodu pokazano w tabeli 2.3.6.

Tabela 2.3.6. Wektory kodowe liniowego kodu Hamminga (7,4)

a_1	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1
a_2	0	1	1	0	0	1	1	0	1	0	0	1	1	0	0	1
a_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
a_4	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1
a_5	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
a_6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
a_7	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Koder kodu Hamminga można zrealizować programowo lub za pomocą układów logicznych. Realizację kodera za pomocą układów logicznych pokazano na rys 2.3.1.



Rys. 2.3.1. Schemat blokowy kodera liniowego kodu Hamminga (7,4)

Koder na rys. 2.3.1 zawiera dwa rejestry: wejściowy i wyjściowy. Proces kodowania odbywa się za pomocą sumatorów po wprowadzeniu informacji do rejestru wejściowego. Wektor kodowy można pobrać z rejestru wyjściowego.

Dekodowanie

Proces dekodowania liniowego kodu Hamminga przebiega podobnie, jak to opisano w punkcie 2.3.4. Składa się on z następujących czynności:

1. Obliczenia syndromu wektora odebranego.
2. Wyznaczenia wektora błędów związanego z obliczonym syndromem.
3. Korekcji błędów.

Elementy syndromu obliczamy z (2.3.9), stosując równania (2.3.15):

$$s_1 = a_1 + a_3 + a_5 + a_7,$$
$$s_2 = a_2 + a_3 + a_6 + a_7,$$
$$s_3 = a_4 + a_5 + a_6 + a_7.$$

Syndrom $\boldsymbol{s} = [s_3\ s_2\ s_1]$ w tym przypadku wyznacza pozycję błędną wektora kodowego w postaci liczby binarnej.

Załóżmy, że wektorem odebranym kodu (7,4) jest wektor

$$[a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7] = [1\,0\,1\,1\,1\,1\,0].$$

Syndrom tego wektora będzie: $\boldsymbol{s} = [1\,0\,1]$. Liczbę tę zamieniamy na liczbę dziesiętną i otrzymujemy w ten sposób numer pozycji błędnej wektora kodowego:

$$1\,0\,1_2 = 5_{10}.$$

Wektor błędów ma więc postać [0000100]. Aby skorygować wektor odebrany, dodajemy do niego wektor błędów

$$[1\,0\,1\,1\,1\,1\,0] + [0\,0\,0\,0\,1\,0\,0] = [1\,0\,1\,1\,0\,1\,0].$$

Skorygowanym wektorem kodowym jest wektor [1011010].

Dekodowanie liniowego kodu Hamminga można wykonać również innymi metodami podanymi w p. 2.3.4.

2.4. Kody cykliczne

2.4.1. Charakterystyka kodów cyklicznych

Kody cykliczne są podklasą kodów liniowych i znalazły największe zastosowania praktyczne. Popularność kodów cyklicznych wynika z następujących ich zalet:

- istnieją efektywne algebraiczne metody konstrukcji kodów cyklicznych o wymaganych właściwościach,
- realizacja koderów i dekoderów kodów cyklicznych za pomocą rejestrów przesuwnych ze sprzężeniem zwrotnym jest stosunkowo prosta.

W algebrze kodów cyklicznych ciągi informacyjne i kodowe zapisuje się w postaci wielomianów, a właściwości kodów opisuje się za pomocą pojęć z zakresu pierścieni wielomianów i ciał Galois.

Nazwa kodów cyklicznych pochodzi od właściwości przesunięcia cyklicznego, którą spełniają wektory kodowe. Stąd wywodzi się też definicja kodu cyklicznego.

Kod (n,k) jest kodem cyklicznym, jeśli każdy wektor kodowy

$$\boldsymbol{c} = \left[a_{n-1}, a_{n-2}, \ldots, a_1, a_0\right]$$

po *i*-tym przesunięciu cyklicznym daje wektor

$$\boldsymbol{c}_i = \left[a_{n-1-i}, a_{n-2-i}, \ldots, a_1, a_0, a_{n-1}, a_{n-2}, \ldots, a_{n-i}\right]$$

będący również wektorem kodowym tego kodu.

Wektor kodowy $\boldsymbol{c}$ można zapisać w postaci wielomianu

$$c(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0. \tag{2.4.1}$$

Właściwość przesunięcia cyklicznego dla wielomianowej postaci słów kodowych definiuje się w następujący sposób. Jeśli $c(x)$ jest wielomianem kodowym stopnia $n-1$, to reszta z dzielenia $x^i\, c(x)$ przez $x^n - 1$ jest również wektorem kodowym tego kodu.

Dalsze rozważania będą dotyczyć wielomianów nad ciałami charakterystyki 2. Ponieważ w ciałach charakterystyki dwa $-1=1$, więc zamiast $x^n - 1$ można podstawić $x^n + 1$. Aby pokazać prawdziwość powyższego twierdzenia, mnożymy (2.4.1) przez x

$$x\,c(x) = a_{n-1}x^n + a_{n-2}x^{n-1} + \ldots + a_1 x^2 + a_0 x.$$

Do prawej strony równania dodajemy dwukrotnie a_{n-1}, co nie zmienia wartości wyrażenia

$$x\,c(x) = a_{n-1}x^n + a_{n-1} + a_{n-2}x^{n-1} + \ldots + a_1 x^2 + a_0 x + a_{n-1},$$

$$x\,c(x) = a_{n-1}\left(x^n + 1\right) + a_{n-2}x^{n-1} + \ldots + a_1 x^2 + a_0 x + a_{n-1}.$$

Po jednokrotnym przesunięciu cyklicznym wektora kodowego ***c*** odpowiadający mu wielomian będzie miał postać

$$c_1(x) = a_{n-2}x^{n-1} + \ldots + a_1x^2 + a_0x + a_{n-1}.$$

Uwzględniając to wyrażenie, otrzymamy

$$x\,c(x) = a_{n-1}\left(x^n + 1\right) + c_1(x).$$

Ponieważ $c_1(x)$ jest stopnia $n-1$, więc nie może się dzielić przez $x^n + 1$ i jest resztą z dzielenia $x\,c(x)$ przez $x^n + 1$. Możemy zatem napisać

$$c_1(x) \equiv x\,c(x)\left(\text{mod } x^n + 1\right).$$

Podobnie po *i*-tym przesunięciu cyklicznym wektora nad ciałem charakterystyki dwa otrzymamy

$$c_i(x) \equiv x^i\,c(x)\left(\text{mod } x^n + 1\right). \tag{2.4.2}$$

P r z y k ł a d 2.4.1.

Kod z bitem parzystości.

Kod z bitem parzystości można traktować jako kod liniowy lub cykliczny. Kod cykliczny z bitem parzystości $(n,k) = (n, n-1)$ generowany przez wielomian generujący $g(x) = x + 1$. Parametry kodu: odległość minimalna kodu $d = 2$, zdolność detekcyjna $l = 1$.

Załóżmy, że informacja jest przekazywana za pomocą liczb zawierających po trzy bity, wówczas $(n,k) = (4,3)$. Kod będzie zawierał osiem wektorów kodowych pokazanych w poniższej tabeli.

Informacja	000	001	010	011	100	101	110	111
Wektor kodowe	0000	0011	0101	0110	1001	1010	1100	1111

Rozważmy wektor kodowy [1010]. Po przesunięcie cyklicznym tego wektora w lewo otrzymamy wektor [0101] należący do zbioru wektorów kodowych. Podczas przesunięcia cyklicznego w lewo pierwszy bit zostaje przesunięty na ostatnią pozycję.

Wektor można również przesunąć cyklicznie, posługując się wielomianami. W tym celu zapiszmy rozważany wektor w postaci wielomianu: $x^3 + x$. Po pomnożeniu wielomianu przez *x* i podstawieniu do (2.4.2) otrzymamy:

$$x^2 + 1 \equiv x\left(x^3 + x\right)\left(\text{mod } x^4 + 1\right).$$

Wielomian $x^2 + 1$ odpowiada wektorowi przesuniętemu [0101].

2.4.2. Wielomiany generujące kody cykliczne

W teorii blokowych kodów cyklicznych wykorzystuje się pojęcie pierścienia klasy reszt modulo $x^n - 1$, opisane w p. 2.2.1. W dalszych rozważaniach ograniczymy się do kodów nad *GF*(2). Pierścień klas reszt modulo $x^n + 1$ jest pierścieniem wielomianów stopnia nie większego niż $n - 1$, które odpowiadają ciągom binarnym o długości *n*.

Ideałem jest podzbiór wielomianów pierścienia generowany przez pewien wielomian $g(x)$, który jest dzielnikiem $x^n + 1$. Ideał ten stanowi kod, a wielomian $g(x)$ nazywamy *wielomianem generującym kod*. Z (2.4.2) i właściwości kongruencji wynika, że wielomian $g(x)$ dzieli bez reszty każdy wielomian odpowiadający wektorowi kodowemu. Stopień wielomianu generującego kod określa liczbę elementów kontrolnych wektora kodowego.

Z powyższych rozważań wynika, że wielomianem generującym kod cykliczny może być każdy wielomian, który jest podzielnikiem $x^n + 1$, gdzie $n = q^m - 1$, a *m* jest liczbą naturalną.

P r z y k ł a d 2.4.2.

Wyznaczenie wielomianów generujących kody cykliczne.

Posługując się tabelą 1.5.6, zapiszmy dwumian $x^{2^3-1} + 1$ w postaci iloczynu wielomianów nierozkładalnych

$$x^{2^3-1} + 1 = x^7 + 1 = (x+1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Każdy z wielomianów otrzymanych z tego rozkładu lub ich iloczyny mogą być zastosowane do generowania kodu cyklicznego:

- $g(x) = x + 1$ generuje kod z bitem parzystości $(n, n-1)$,
- $g(x) = x^3 + x + 1$ generuje kod cykliczny Hamminga (7,4),
- $g(x) = x^3 + x^2 + 1$ generuje kod cykliczny Hamminga (7,4),
- $g(x) = (x+1)(x^3 + x + 1)$ generuje kod cykliczny (7,3),
- $g(x) = (x+1)(x^3 + x^2 + 1)$ generuje kod cykliczny (7,3),
- $g(x) = (x^3 + x + 1)(x^3 + x^2 + 1)$ generuje kod cykliczny (7,1).

Dla każdego kodu cyklicznego (n,k) istnieje *cykliczny kod dualny* $(n, n-k)$. Jeżeli kod cykliczny jest generowany przez wielomian $g(x)$, to wielomianem generującym kod dualny będzie wielomian $g_D(x)$ spełniający zależność

$$g(x)\, g_D(x) \left(\mathrm{mod}\ x^n + 1\right) = 0. \tag{2.4.3}$$

Na przykład dla kodu Hamminga (7,4), generowanego przez wielomian $g(x) = x^3 + x^2 + 1,$ istnieje kod dualny generowany przez wielomian

$$g_D(x) = \frac{x^7 + 1}{x^3 + x^2 + 1} = (x + 1)(x^3 + x + 1) = x^4 + x^3 + x^2 + 1.$$

2.4.3. Algorytm kodowania

Do kodowanie informacji za pomocą kodów cyklicznych można wykorzystać wielomian generujący kod lub macierz generującą kod. W pierwszej kolejności rozważymy kodowanie za pomocą wielomianu generującego.

Wektor kodowy cyklicznego kodu systematycznego ma formę:

$$\boldsymbol{c}_X = \left[m_{n-1}, \ldots, m_{n-k}, r_{n-k-1}, \ldots, r_0\right], \tag{2.4.4}$$

gdzie współrzędne m_i są elementami informacyjnymi, a współrzędne r_i – elementami kontrolnymi.

Gdy mamy wielomian generujący $g(x)$ stopnia $n - k$, to aby obliczyć wektor kodowy systematycznego kodu cyklicznego (n,k), należy wykonać następujące czynności:

1. Wielomian odpowiadający informacji $m(x)$ pomnożyć przez x^{n-k}

$$x^{n-k} m(x).$$

2. Otrzymany iloczyn $x^{n-k} m(x)$ podzielić przez wielomian generujący kod $g(x)$ i wyznaczyć resztę $r(x)$ z tego dzielenia

$$x^{n-k} m(x) = q(x)\, g(x) + r(x).$$

3. Obliczyć wielomian $c_X(x)$, odpowiadający wektorowi kodowemu, dodając $x^{n-k} m(x)$ i resztę $r(x)$

$$c_X(x) = x^{n-k} m(x) + r(x).$$

Napiszmy ciąg informacyjny w postaci wielomianu

$$m(x) = m_{k-1} x^{k-1} + m_{k-2} x^{k-2} + \ldots + m_1 x + m_0.$$

Pomnożenie tego wielomianu przez x^{n-k} jest równoważne z przesunięciem wektora kodowego w lewo o $n - k$ pozycji

$$m(x)\, x^{n-k} = m_{k-1} x^{n-1} + m_{k-2} x^{n-2} + \ldots + m_1 x^{n-k+1} + m_0 x^{n-k}.$$

Dzielenie uzyskanego wyrażenia przez $g(x)$ można zapisać w formie algorytmu Euklidesa (2.2.3)

$$m(x)\, x^{n-k} = q(x)\, g(x) + r(x),$$

gdzie $q(x)$ jest częścią całkowitą, a $r(x)$ resztą z dzielenia w postaci

$$r(x) = r_{n-k-1}x^{n-k-1} + \ldots r_1 x + r_0 .$$

Wzór umożliwiający obliczenia części kontrolnej wektora kodowego możemy również zapisać w formie kongruencji

$$\sum_{i=1}^{k} m_{n-i}x^{n-i} \equiv \sum_{i=1}^{n-k} r_{n-k-i}x^{n-k-i} \left(\text{mod } g(x)\right). \tag{2.4.5}$$

Wielomian z lewej strony kongruencji odpowiada części informacyjnej wektora kodowego, a wielomian z prawej strony – części kontrolnej wektora kodowego, która jest równa reszcie z dzielenia wielomianu informacyjnego przez wielomian generujący kod $g(x)$.

P r z y k ł a d 2.4.3.

Kodowanie informacji za pomocą cyklicznego kodu Hamminga (7,4).

Cykliczne kody Hamminga $(n,k) = (2^m - 1, 2^m - m - 1)$ są generowane przez wielomiany pierwotne stopnia m. Kody te mają następujące parametry:

- odległość minimalna $d = 3$,
- zdolność korekcyjna $t = 1$.

Niech wielomianem generującym kod będzie wielomian pierwotny:

$$g(x) = x^3 + x + 1.$$

Wielomian ten generuje kod $(n,k) = (7,4)$. Obliczmy wektor kodowy dla ciągu informacyjnego $m(x)$=1101. Obliczenia wykonamy, posługując się ciągami współczynników.

1. Wielomian informacyjny $m(x)$ mnożymy przez x^3 i otrzymujemy $x^3 m(x)$= 1101000.

2. Iloczyn $x^3 m(x)$ dzielimy przez wielomian generujący kod $g(x)$=1011.

```
                    1 1 1 1
  1 0 1 1 | 1 1 0 1 0 0 0
            1 0 1 1
            -------
              1 1 0 0
              1 0 1 1
              -------
                1 1 1 0
                1 0 1 1
                -------
                  1 0 1 0
                  1 0 1 1
                  -------
                        1
```

Wynik dzielenia możemy zapisać w postaci algorytmu dzielenia Euklidesa (2.2.3)

$$1101000 = 1111 \cdot 1011 + 1.$$

Część całkowita z dzielenia wynosi 1111, a reszta $r(x)=1$.

3. Resztę otrzymaną z dzielenia $r(x)$ dodajemy do $x^3 m(x)$ i otrzymujemy ciąg $c_X(x)$, odpowiadający współrzędnym wektora kodowego

$$c_X(x) = x^3 m(x) + r(x) = 1101000 + 1 = 1101001.$$

Podobnie można obliczyć pozostałe wektory tego kodu. Za ciągi informacyjne należy przyjąć wszystkie kombinacje liczb zawierających cztery bity. Kod (7,4) będzie miał szesnaście wektorów kodowych.

2.4.4. Uproszczony algorytm dekodowania

W czasie transmisji wektorów kodowych kanałem transmisyjnym powstają błędy transmisyjne. Zadaniem dekodera jest wykrycie lub wykrycie i usunięcie tych błędów.

Możliwości korekcyjne kodu są określone wzorem (2.2.6). Każdy kod cykliczny ma swój algorytm dekodowania, który pozwala skorygować wszystkie błędy korygowalne przez dany kod. W praktyce często używa się algorytmu uproszczonego, wspólnego dla wszystkich kodów cyklicznych. Algorytm ten umożliwia wykrycie i korekcję wszystkich błędów znajdujących się na $n-k$ pozycjach wektora kodowego. Algorytm ten omówimy szczegółowo.

W procesie dekodowania oblicza się syndrom wektora odebranego. Dla kodów cyklicznych syndrom oblicza się, dzieląc wielomian $c_Y(x)$, odpowiadający wektorowi odebranemu $\boldsymbol{c}_Y$, przez wielomian generujący kod $g(x)$. Syndrom $s(x)$ jest równy reszcie z tego dzielenia

$$c_Y(x) = q(x)\,g(x) + s(x). \tag{2.4.6}$$

Syndrom $s(x)$ jest wielomianem stopnia $\le n-k-1$. Jeśli syndrom ma wartość zerową, oznacza to, że wektor odebrany jest wektorem kodowym i w czasie transmisji nie wystąpiły żadne błędy wykrywalne przez kod. Niezerowa wartość syndromu świadczy o tym, że odebrany wektor nie jest wektorem kodowym i zostały wykryte błędy transmisyjne.

Wektor odebrany $\boldsymbol{c}_Y$, jak to pokazano w (2.2.2), jest sumą wektora nadanego $\boldsymbol{c}_X$ i wektora błędów $\boldsymbol{e}$. Wzór ten zapisujemy w postaci wielomianów

$$c_Y(x) = c_X(x) + e(x).$$

Wielomian odpowiadający wektorowi kodowemu $\boldsymbol{c}_X$ dzieli się bez reszty przez wielomian generujący kod $g(x)$, można zatem napisać

$$c_X(x) = m(x)\,g(x).$$

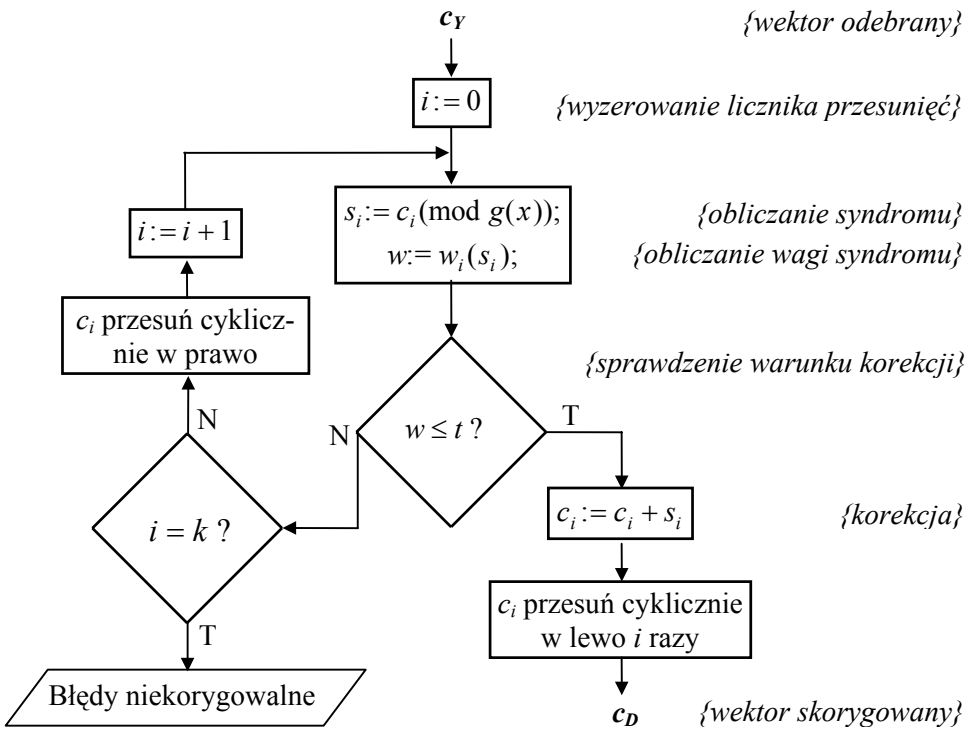
Podstawiając tę zależność do wzoru poprzedniego, otrzymamy

$$c_Y(x) = m(x)\,g(x) + e(x).$$

Porównujemy prawą stronę tego wzoru z prawą stroną wzoru (2.4.6). Po przekształceniach mamy

$$e(x) = \big(m(x) + q(x)\big)\,g(x) + s(x). \tag{2.4.7}$$

Syndrom jest resztą z dzielenia wielomianu odpowiadającego wektorowi błędów $e(x)$ przez wielomian generujący kod $g(x)$. Syndrom zawiera informację o położeniu błędów transmisyjnych, co jest wykorzystywane w trakcie korekcji błędów.



Rys. 2.4.1. Schemat blokowy uproszczonego algorytmu dekodowania

Schemat blokowy algorytmu dekodowania z korekcją błędów pokazano na rys. 2.4.1. Na rysunku zastosowano takie same oznaczenia jak w opisie algorytmu. Parametr i oznacza kolejne cykle dekodowania.

Cały proces dekodowania przebiega następująco. Na początku wyznacza się syndrom wektora odebranego s a następnie oblicza się jego wagę Hamminga $w(s)$. Mogą wówczas wystąpić następujące przypadki:

1. Waga syndromu jest mniejsza lub równa zdolności korekcyjnej kodu, $w(s) \le t$. Oznacza to, że błędy są położone w części kontrolnej wektora kodowego. Wektor

odebrany $\boldsymbol{c}_Y$ może być wtedy skorygowany przez dodaniu syndromu do wektora odebranego. W wyniku tego działania otrzymamy wektor wyjściowy dekodera $\boldsymbol{c}_D$

$$\boldsymbol{c}_D = \boldsymbol{c}_Y + \boldsymbol{s}. \tag{2.4.8}$$

Na podstawie tego wektora można wyznaczyć informację odebraną m^*. Będzie ona równa części informacyjnej wektora $\boldsymbol{c}_D$.

2. Waga syndromu jest większa od zdolności korekcyjnej kodu, $w(s) > t$. Przypadek ten oznacza, że błędy obejmują część informacyjną wektora kodowego. Należy wówczas przesunąć cyklicznie wektor odebrany tak, aby błędy znalazły się w części kontrolnej, a potem go skorygować. W tym celu wykonujemy następujące czynności. Przesuwamy wektor odebrany cyklicznie o jedną pozycję w dowolnym kierunku (np. w prawo), obliczamy syndrom i jego wagę oraz sprawdzamy, czy został spełniony warunek podany w p. 1, czy też warunek podany w p. 2.

- Jeżeli $w(s) \le t$, należy skorygować wektor odebrany zgodnie z p. 1, a następnie przesunąć go cyklicznie w odwrotną stronę (w lewo), aby odtworzyć jego pierwotną postać.
- Jeżeli $w(s) > t$, trzeba ponownie przesuwać cyklicznie wektor odebrany w tę samą stronę, obliczając po każdym przesunięciu syndrom i jego wagę aż do momentu, kiedy $w(s) \le t$. Wtedy należy skorygować wektor odebrany i przesunąć go w odwrotną stronę o taką samą liczbę pozycji.

W przypadku gdy po k przesunięciach cyklicznych nie uda się skorygować wektora odebranego oznacza to, że wystąpiły błędy niekorygowalne co może być zasygnalizowane odpowiednim komunikatem.

P r z y k ł a d 2.4.4.

Dekodowanie wektora odebranego cyklicznego kodu Hamminga (7,4).

Załóżmy, że na wejście kanału został podany wektor kodowy [1 1 0 1 0 0 1] cyklicznego kodu Hamminga (7,4). Wektor ten obliczono w przykładzie 2.4.3. Na wyjściu kanału odebrano wektor z błędem na pozycji trzeciej [1 1 1 1 0 0 1]. Należy wykonać korekcję tego wektora.

W pierwszej kolejności obliczamy syndrom wektora odebranego i jego wagę. W tym celu dzielimy wektor odebrany przez wielomian generujący kod. Dzielenie wykonujemy podobnie jak w przykładzie 2.4.3, a wynik dzielenia zapisujemy w postaci algorytmu Euklidesa

$$1\,1\,1\,1\,0\,0\,1 = 1\,1\,0\,1 \cdot 1\,0\,1\,1 + 1\,1\,0.$$

Otrzymaliśmy: $s(x) = 1\,1\,0$, $w(s) = 2$. Ponieważ dla cyklicznego kodu Hamminga $t = 1$, więc $w(s) > t$, co oznacza, że w wektorze odebranym występuje błąd lub błędy w części informacyjnej, których nie można skorygować.

Przesuwamy cyklicznie wektor odebrany w prawo. Po wykonaniu dzielenia otrzymamy

$$1111100 = 1101 \cdot 1011 + 11,$$

skąd: $s(x)=11$, $w(s)=2$, $w(s)>t$.

Przesuwamy ponownie wektor odebrany w prawo. Po wykonaniu dzielenia w tym przypadku otrzymamy

$$1111\,10 = 110 \cdot 1011 + 100,$$

skąd: $s(x)=100$, $w(s)=1$.

Ponieważ $w(s) = t$, korygujemy przesunięty wektor odebrany $\boldsymbol{c}_{YP}$ korzystając z (2.4.8) i otrzymujemy wektor skorygowany $\boldsymbol{c}_{DP}$ przesunięty o dwie pozycje w prawo

$$\mathbf{c}_{DP} = \mathbf{c}_{YP} + \mathbf{s} = [0\,1\,1\,1\,1\,1\,0] + [0\,0\,0\,0\,1\,0\,0] = [0\,1\,1\,1\,0\,1\,0].$$

Przesuwamy otrzymany wektor o dwie pozycje w lewo i otrzymujemy wektor skorygowany [1101001]. Wektor ten jest równy wektorowi na wejściu kanału transmisyjnego.

2.5. Macierzowy opis kodów cyklicznych

Algorytmy kodowania i dekodowania informacji za pomocą kodów cyklicznych można zwięźle opisać, korzystając z macierzy. Macierzowy opis kodów liniowych podano w p. 2.3.2. W przypadku kodów cyklicznych macierz generującą kod oblicza się, wykorzystując cykliczne właściwości kodów. Zasady ogólne obliczania macierzy kontrolnej na podstawie macierzy generującej oraz algorytmy kodowania i dekodowania są takie same jak dla kodów liniowych.

2.5.1. Wyznaczanie macierzy generującej na podstawie wielomianu generującego kod

Macierz generująca kod $\boldsymbol{G}$ o wymiarach $k \times n$ służy do kodowania informacji. Macierz ta zawiera k niezależnych wektorów i stanowi bazę przestrzeni kodowej. Pierwsze k elementów każdego wiersza macierzy można traktować jako część informacyjną wektora kodowego, a pozostałe $n-k$ elementów jako jego część kontrolną. Strukturę macierzy $\boldsymbol{G}$ w postaci kanonicznej podano w (2.3.4).

Macierz generująca kod jest powiązana z wielomianem generującym kod $g(x)$. Wiersze macierzy generującej kod cykliczny można wyznaczyć za pomocą procedury kodowania opisanej w p. 2.4.3. Jeżeli część informacyjną wektora kodowego przedstawimy w postaci $x^{n-k}m(x)$ to

$$x^{n-k}m(x) = q(x)g(x) + r(x). \tag{2.5.1}$$

Sumy $x^{n-k}m(x) + r(x)$ odpowiadają wektorom kodowym, które tworzą macierz generującą $\boldsymbol{G}$.

Macierz generującą kod cykliczny można również obliczyć, wykorzystując właściwość przesunięcia cyklicznego kodu. W tym celu wyznaczamy macierz pomocniczą $\boldsymbol{G}'$ o wymiarach $k \times n$, której wiersze obliczamy z iloczynu $x^i g(x)$ dla $i = 0,1,\ldots,k-1$

$$\boldsymbol{G}'_{k,n} = \begin{bmatrix} 1 \cdot g(x) \\ x \cdot g(x) \\ x^2 \cdot g(x) \\ \cdots \\ x^{k-1} \cdot g(x) \end{bmatrix}. \tag{2.5.2}$$

Tak utworzone wiersze macierzy są przesunięciem cyklicznym pierwszego wiersza, który jest wielomianem generującym kod. Macierz $\boldsymbol{G}$ można wyznaczyć, korzystając z kombinacji liniowej wierszy macierzy $\boldsymbol{G}'$. Metodę tę ilustruje przykład 2.5.1.

P r z y k ł a d 2.5.1.

Wyznaczenie macierzy generującej kod cykliczny Hamminga $(7,4)$.

Kod cykliczny Hamminga opisano w przykładzie 2.4.3. Niech wielomianem generującym kod będzie wielomian

$$g(x) = x^3 + x + 1.$$

Na początku wyznaczamy macierz $\boldsymbol{G}'$. Następnie obliczamy macierz $\boldsymbol{G}$, używając kombinacji liniowych wierszy macierzy $\boldsymbol{G}'$.

$$\boldsymbol{G}' = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Pierwszy wiersz macierzy $\boldsymbol{G}$ jest sumą wierszy: pierwszego, drugiego i czwartego, macierzy $\boldsymbol{G}'$; drugi wiersz jest sumą wiersza: pierwszego i trzeciego; itd.

W przypadku kodów cyklicznych informację można kodować wykorzystując jedną z następujących możliwości:

- wielomian generujący kod $g(x)$,
- macierz generującą $\boldsymbol{G}$,
- równania kontrolne.

Kodowanie informacji z zastosowaniem macierzy generującej opisano w p. 2.3.3. Metody tej używa się również w przypadku kodów cyklicznych. Macierzowy opis kodu cyklicznego pozwala wyznaczyć równania kontrolne kodu cyklicznego. Metodę kodowania i dekodowania informacji za pomocą równań kontrolnych dla kodów liniowych opisano w rozdziale 2.3.

W przypadku zastosowania macierzy generującej $\boldsymbol{G}$ wektor kodowy $\boldsymbol{c_X}$ obliczamy z zależności

$$\boldsymbol{c}_X = \boldsymbol{m}\boldsymbol{G}_{k,n} = [m_1, m_2, \ldots, m_k] \cdot \begin{bmatrix} 1 & 0 & \ldots & 0 & p_{11} & p_{12} & \cdots & p_{1,n-k} \\ 0 & 1 & \ldots & 0 & p_{21} & p_{22} & \cdots & p_{2,n-k} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & p_{k1} & p_{k2} & \cdots & p_{k,n-k} \end{bmatrix}. \tag{2.5.3}$$

Po wykonaniu mnożenia otrzymamy wektor kodowy $\boldsymbol{c_K}$ o następujących współrzędnych:

$$\begin{aligned} c_i &= m_i \quad \text{dla} \quad i = 1,2,\ldots,k\,, \\ c_{k+j} &= m_1 p_{1j} + m_2 p_{2j} + \ldots + m_k p_{k,j} \quad \text{dla} \quad j = 1,2,\ldots,n-k. \end{aligned} \tag{2.5.4}$$

Z wyrażenia tego widać, że pierwszych k elementów wektora kodowego to elementy informacyjne, a pozostałe $n-k$ elementów są liniową funkcją elementów informa-

cyjnych. Równania c_{k+j} nazywamy *równaniami kontrolnymi kodu* (parity-check equations).

Wektory kodowe można również obliczyć za pomocą macierzy kontrolnej $\boldsymbol{H}$ określonej wzorem (2.3.6). Wtedy równania kontrolne c_{k+j} mają taką samą postać jak podane w (2.5.4). Ogólnie można stwierdzić, że kod liniowy jest określony przez macierz generującą lub macierz kontrolną.

P r z y k ł a d 2.5.2.

Wyznaczenie równań kontrolnych kodu cyklicznego.

Przyjmijmy macierz generującą z przykładu 2.5.1. Wektor kodowy obliczamy dla ciągu informacyjnego $m = (m_1, m_2, m_3, m_4)$

$$\boldsymbol{c}_X = \left[c_1, c_2, c_3, c_4, c_5, c_6, c_7\right] = \left[m_1, m_2, m_3, m_4\right] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} =$$

$$= \left[m_1, m_2, m_3, m_4, m_1 + m_2 + m_3, m_2 + m_3 + m_4, m_1 + m_2 + m_4\right].$$

Dla tego kodu równania określone wzorem (2.5.6) będą miały postać:

$$c_1 = m_1, \quad c_2 = m_2, \quad c_3 = m_3, \quad c_4 = m_4,$$
$$c_5 = m_1 + m_2 + m_3, \quad c_6 = m_2 + m_3 + m_4, \quad c_7 = m_1 + m_2 + m_4.$$

2.5.2. Wyznaczenie macierzy kontrolnej na podstawie wielomianu generującego kod dualny

W procesie dekodowania stosuje się macierz kontrolną $\boldsymbol{H}$. Macierz ta ma wymiary $(n-k) \times n$, a wiersze macierzy $\boldsymbol{G}$ są ortogonalne do wierszy macierzy $\boldsymbol{H}$, to jest $\boldsymbol{G} \cdot \boldsymbol{H}^T = 0$, gdzie $\boldsymbol{H}^T$ jest transponowaną macierzą $\boldsymbol{H}$. Postać macierzy kontrolnej $\boldsymbol{H}$ określa wzór (2.3.6), a macierzy $\boldsymbol{H}^T$ – wzór (2.3.7).

Macierz kontrolną $\boldsymbol{H}$ zwykle obliczamy z macierzy generującej $\boldsymbol{G}$. Metodę tę opisano w p. 2.3.2. Istnieje również możliwość obliczenia macierzy kontrolnej $\boldsymbol{H}$, gdy znamy wielomian generujący kod dualny.

Przestrzeń wektorowa generowana przez macierz $\boldsymbol{G}$ oraz przestrzeń wektorowa generowana przez macierz $\boldsymbol{H}$ są podprzestrzeniami przestrzeni wektorowej zawierającej wszystkie wektory n-elementowe i dlatego stanowią one kody liniowe. Jeśli macierz $\boldsymbol{G}$ jest macierzą generującą kod (n,k), to za pomocą macierzy $\boldsymbol{H}$ można wygenerować kod dualny $(n, n-k)$. Gdy kod jest przestrzenią wierszową macierzy, wówczas kod dualny nazywamy *przestrzenią zerową macierzy* i odwrotnie.

Wielomian generujący kod dualny określa wzór (2.4.3). Obliczanie macierzy kodu dualnego pokazuje przykład 2.5.3.

P r z y k ł a d 2.5.3.

Obliczenia macierzy generującej i kontrolnej kodu Hamminga $(7,4)$ oraz jego kodu dualnego $(7,3)$.

Niech wielomianem generującym cykliczny kod Hamminga $(7,4)$ będzie wielomian

$$g(x) = x^3 + x^2 + 1.$$

Macierz generującą kod $\boldsymbol{G}$ dla powyższego wielomianu generującego, obliczamy podobnie jak w przykładzie 2.5.1. Z macierzy tej wyznaczamy macierz kontrolną $\boldsymbol{H}$ oraz kontrolną macierz transponowaną $\boldsymbol{H}^T$:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{H}^T = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Wielomianem generującym kod dualny jest

$$g_D(x) = \frac{x^7 + 1}{x^3 + x^2 + 1} = x^4 + x^3 + x^2 + 1.$$

Podobnie jak wyżej, z wielomianu $g_D(x)$ można obliczyć macierz generującą kod dualny. Macierz tę oznaczamy przez $\boldsymbol{G}_D$. Z macierzy generującej wyznaczamy macierz kontrolną $\boldsymbol{H}_D$ i transponowaną macierz kontrolną $\boldsymbol{H}_D^T$. W wyniku tych obliczeń otrzymamy:

$$\boldsymbol{G}_D = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{H}_D = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{H}_D^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Macierze $\boldsymbol{H}^T$ i $\boldsymbol{G}_D$ różnią się tym, że ich wiersze i kolumny są w odwrotnej kolejności. To samo dotyczy macierzy $\boldsymbol{H}_D^T$ i $\boldsymbol{G}$. Kolejność wierszy nie zmienia ani przestrzeni wierszowej macierzy, ani jej przestrzeni zerowej. Zmiana kolejności kolumn wynika z faktu, że iloczyn dwóch wielomianów równa się zeru wtedy, kiedy zerem jest iloczyn skalarny odpowiadających im wektorów z odwróconą kolejnością współrzędnych w jednym z nich.

2.5.3. Definicja kodu cyklicznego za pomocą pierwiastków wielomianu generującego kod

Kod cykliczny najczęściej definiuje się za pomocą wielomianu generującego. Ta klasyczna definicja kodu może być zapisana w następujący sposób:

$$\begin{aligned} c(x)\,(\bmod g(x)) &= 0, \\ \left(x^n+1\right)(\bmod g(x)) &= 0, \end{aligned} \tag{2.5.5}$$

gdzie $c(x)$ jest ciągiem kodowym, a n długością tego ciągu.

Z algebry wiadomo, że każdy wielomian ma pierwiastki w ciele odpowiedniego rzędu. Jeśli wielomian generujący kod $g(x)$ jest podzielny przez wielomian pierwotny stopnia m, a nie jest podzielny przez żaden wielomian pierwotny stopnia większego od m, to liczba m określa rząd ciała rozkładu wielomianu generującego kod $g(x)$. Wtedy elementy ciała $GF(2^m)$ są pierwiastkami wielomianu generującego. Dla większości kodów cyklicznych z m związana jest również długość ciągu kodowego. Jeśli pierwiastki wielomianu generującego kod są elementami ciała $GF(2^m)$, to długość wektora kodowego jest dzielnikiem liczby $2^m - 1$.

Analizując właściwości kodów cyklicznych stwierdzono, że pierwiastki wielomianu generującego pozwalają zdefiniować kod cykliczny i wyznaczyć macierz kontrolną kodu.

Załóżmy, że wielomianem należącym do przestrzeni kodowej jest wielomian

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_1 x + c_0 .$$

Jeśli pierwiastkami tego wielomianu są $\alpha_1, \alpha_2, \ldots, \alpha_r$, to dla $i = 1, 2, \ldots, r$ będzie on miał wartości zerowe

$$c(\alpha_i) = c_{n-1}\alpha_i^{n-1} + c_{n-2}\alpha_i^{n-2} + \ldots + c_1\alpha_i + c_0 = 0.$$

Wyrażenie to można zapisać jako iloczyn macierzy

$$\left[c_{n-1}, c_{n-2}, \ldots, c_1, c_0\right]\left[\alpha_i^{n-1}, \alpha_i^{n-2}, \ldots, \alpha_i^1, \alpha_i^0\right]^T = 0.$$

Współczynnikami wielomianów kodowych są elementy ciała rozszerzonego $GF(q^m)$.

Druga część tego iloczynu odpowiada transponowanej macierzy kontrolnej. Macierz ta ma wymiary $n \times mr$, gdzie m jest stopniem wielomianu generującego kod. Wiadomo, że macierz $\boldsymbol{H}^{\boldsymbol{T}}$ ma wymiary $n \times r$. Po obliczeniu macierzy okazuje się, że tylko r kolumn macierzy jest liniowo niezależnych, pozostałe zaś są ich kombinacjami liniowymi i mogą być pominięte. Macierz $\boldsymbol{H}^{\boldsymbol{T}}$ ma postać

$$\boldsymbol{H}^{T} = \begin{bmatrix} \alpha_1^{n-1} & \alpha_2^{n-1} & \dots & \alpha_r^{n-1} \\ \alpha_1^{n-2} & \alpha_2^{n-2} & \dots & \alpha_r^{n-2} \\ \dots & \dots & \dots & \dots \\ \alpha_1^{1} & \alpha_2^{1} & \dots & \alpha_r^{1} \\ \alpha_1^{0} & \alpha_2^{0} & \dots & \alpha_r^{0} \end{bmatrix}. \tag{2.5.7}$$

P r z y k ł a d 2.5.4.

Wyznaczenie transponowanej macierzy kontrolnej za pomocą pierwiastków wielomianu generującego.

Rozważmy cykliczny kod Hamminga $(7,4)$ generowany przez wielomian pierwotny z przykładu 2.5.1

$$g(x) = x^3 + x + 1.$$

Ciało rozszerzone $GF(2^3)$ generowane przez ten wielomian ma elementy niezerowe pokazane w tabeli 2.5.1. Drugi wiersz tabeli zawiera elementy ciała w postaci wektorowej. Wartości tych elementów wyznaczono metodą opisaną w p. 1.5.5. Łatwo sprawdzić, że pierwiastkami tego wielomianu generującego są elementy ciała: α, α^2 i α^4.

Tabela 2.5.1. Niezerowe elementy ciała $GF(8)$

Elementy ciała	1	α	α^2	α^3	α^4	α^5	α^6
Wektory	[001]	[010]	[100]	[011]	[110]	[111]	[101]

Ogólną postać macierzy $\boldsymbol{H}^{\boldsymbol{T}}$ otrzymamy po podstawieniu pierwiastków wielomianu generującego kod do (2.5.7)

$$\boldsymbol{H}^{T} = \begin{bmatrix} \alpha^6 & (\alpha^2)^6 & (\alpha^4)^6 \\ \alpha^5 & (\alpha^2)^5 & (\alpha^4)^5 \\ \alpha^4 & (\alpha^2)^4 & (\alpha^4)^4 \\ \alpha^3 & (\alpha^2)^3 & (\alpha^4)^3 \\ \alpha^2 & (\alpha^2)^2 & (\alpha^4)^2 \\ \alpha & \alpha^2 & \alpha^4 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \alpha^6 & \alpha^5 & \alpha^3 \\ \alpha^5 & \alpha^3 & \alpha^6 \\ \alpha^4 & \alpha & \alpha^2 \\ \alpha^3 & \alpha^6 & \alpha^5 \\ \alpha^2 & \alpha^4 & \alpha \\ \alpha & \alpha^2 & \alpha^4 \\ 1 & 1 & 1 \end{bmatrix}.$$

Podstawiając wartości elementów ciała w postaci wektorowej, otrzymamy macierz o wymiarach 7×9

$$\boldsymbol{H}^T = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ k_1 & k_2 & k_3 & k_4 & k_5 & k_6 & k_7 & k_8 & k_9 \end{bmatrix}.$$

Można zauważyć, że kolumny $k_4 \div k_9$ są liniowymi kombinacjami kolumn k_1, k_2 i k_3:

$$k_4 = k_1 + k_2, \quad k_6 = k_3, \quad k_8 = k_1 + k_2,$$
$$k_5 = k_1, \qquad k_7 = k_2, \quad k_9 = k_3.$$

Gdy pominiemy powtarzające się kolumny, wówczas otrzymamy macierz

$$\boldsymbol{H}^T = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

W powyższym przykładzie otrzymaliśmy macierz, która zawiera potęgi elementu α. Spostrzeżenie to można uogólnić. Jeśli wielomian generujący kod jest iloczynem wielomianów nierozkładalnych

$$g(x) = m_i(x) m_j(x) \ldots m_l(x),$$

to macierz kontrolna $\boldsymbol{H}$ będzie

$$\boldsymbol{H} = \begin{bmatrix} \alpha_i^{n-1} & \alpha_i^{n-2} & \ldots & \alpha_i & 1 \\ \alpha_j^{n-1} & \alpha_j^{n-2} & \ldots & \alpha_j & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \alpha_l^{n-1} & \alpha_l^{n-2} & \ldots & \alpha_l & 1 \end{bmatrix}. \tag{2.5.8}$$

2.5.4. Kodowanie i dekodowanie kodów cyklicznych

Podstawowe informacje dotyczące kodowania i dekodowanie kodów liniowych podano w punktach 2.3.3 i 2.3.4. Takie same metody stosuje się również dla kodów cyklicznych.

W realizacji technicznej koderów i dekoderów kodów cyklicznych zwykle ma zastosowanie uproszczony algorytm dekodowania opisany w p. 2.4.4. Realizacja programowa procesu dekodowania umożliwia zastosowanie standardowej tablicy kodu lub metod macierzowych. Zasady kodowania i dekodowania za pomocą macierzy są takie same jak dla kodów liniowych. Podobnie można również skonstruować standardową tablicę kodu, używaną w procesie dekodowania.

W teorii kodów cyklicznych wykorzystuje się pojęcie pierścienia klas reszt modulo $x^n + 1$. Pierścień klas reszt modulo wielomian $x^n + 1$ jest pierścieniem wielomianów stopnia nie większego niż $n-1$, które odpowiadają binarnym ciągom n-elementowym. Pierścień ten wtórnie rozkładamy na warstwy względem ideału generowanego przez wielomian $g(x)$, będący podzielnikiem $x^n + 1$. W wyniku tego rozkładu powstają klasy reszt modulo wielomian $g(x)$. Wielomiany $f_i(x)$, należące do i-tej warstwy, mają tę samą resztę podziału przez wielomian $g(x)$, to znaczy zachodzi

$$f_i(x) \left(\text{mod } g(x)\right) \equiv r_i(x).$$

Zbiór $\{ f_i(x) \}$ o powyższej właściwości nazywamy i-tą klasą reszt.

W przypadku kodów cyklicznych zbiór ciągów kodowych jest podgrupą addytywną zawierającą wszystkie wielokrotności wielomianu generującego kod $g(x)$. W algebrze wielomianów zbiór taki nazywa się jest ideałem, a wielomian $g(x)$ – wielomianem generującym ideał.

Standardową tablicę kodu tworzy się, rozkładając pierścień na klasy reszt względem ideału. Tworzenie tej tablicy i jej właściwości są takie same jak dla kodów liniowych. Aby proces korekcji był poprawny, elementami tworzącymi warstwy muszą być wektory błędów generowane przez kanał transmisyjny. Podczas tworzenia standardowej tablicy kodu należy obliczyć wektory kodowe. Kodowanie informacji z zastosowaniem kodu cyklicznego Hamminga ilustruje przykład 2.5.5.

P r z y k ł a d 2.5.5.
Kodowanie informacji za pomocą cyklicznego kodu Hamminga $(7,4)$.
Niech wielomianem generującym kod cykliczny Hamminga będzie wielomian

$$g(x) = x^3 + x^2 + 1.$$

Kod ten ma odległość minimalną $d = 3$ i może korygować pojedyncze błędy. Macierze $\boldsymbol{G}$, $\boldsymbol{H}$ i $\boldsymbol{H}^T$ tego kodu obliczono w przykładzie 2.5.3. Przyjmijmy, że cią-

giem informacyjnym jest ciąg 1110. Obliczmy wektor kodowy odpowiadający tej informacji. Stosujemy wzór (2.5.3).

$$\boldsymbol{c}_X = \boldsymbol{m}\boldsymbol{G} = [1110]\cdot\begin{bmatrix}1&0&0&0&1&1&0\\0&1&0&0&0&1&1\\0&0&1&0&1&1&1\\0&0&0&1&1&0&1\end{bmatrix} =$$

$$= [1000110] + [0100011] + [0010111] = [1110010].$$

W tabeli 2.5.2 pokazano fragment tablicy rozkładu przestrzeni wektorowej na warstwy względem ideału utworzonego przez wektory kodowe kodu Hamminga (7,4) z przykładu 2.5.3. Drugi wiersz zawiera ciągi kodowe odpowiadające ciągom informacyjnym podanym w wierszu pierwszym. Kod zawiera $2^k = 16$ wektorów kodowych. Elementami tworzącymi warstwy są ciągi zawierające pojedyncze błędy. Elementy warstwy obliczamy, dodając ciągi kodowe i ciągi błędów tworzących warstwę.

Tabela 2.5.2. Tablica dekodowania kodu cyklicznego Hamminga (7,4)

Inform.	0000	0001	0010	0011	. . .	1101	1110	1111	Synd.
Kod	0000000	0001101	0010111	0011010	. . .	1101000	1110010	1111111	000
Klasy	1000000	1001101	1010111	1011010	. . .	0101000	0110010	0111111	110
reszt	0100000	0101101	0110111	0111010	. . .	1001000	1010010	1011111	011
	0010000	0011101	0000111	0001010	. . .	1111000	1100010	1101111	111
	0001000	0000101	0011111	0010010	. . .	1100000	1111010	1110111	101
	0000100	0001001	0010011	0011110	. . .	1101100	1110110	1111011	100
	0000010	0001111	0010101	0011000	. . .	1101010	1110000	1111101	010
	0000001	0001100	0010110	0011011	. . .	1101001	1110011	1111110	001

Wiersz zawierający ciągi kodowe odpowiada klasie reszt $\{0\}$. Pozostałe wiersze tabeli 2.5.2 zawierają ciągi odpowiadające wektorom z błędami. Ciągi te rozłożono na klasy reszt względem ideału. W kolejnych wierszach podano odpowiednio klasy reszt: $\{x^6\}, \{x^5\}, \ldots, \{1\}$, odpowiadające pojedynczym błędom na pozycjach od pierwszej do siódmej.

W ostatniej kolumnie przedstawiono syndromy odpowiadające poszczególnym warstwom. Syndromy te obliczono za pomocą macierzy $\boldsymbol{H}^T$ dla elementów tworzących warstwy. Są one takie same dla wszystkich elementów warstwy.

Jeśli np. wektorem odebranym jest wektor $\boldsymbol{c}_Y = [1\,0\,1\,0\,0\,1\,0]$, to otrzymamy następujący syndrom

$$\boldsymbol{s} = \boldsymbol{c}_Y \boldsymbol{H}^T = [1\,0\,1\,0\,0\,1\,0] \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [1\,1\,0] + [1\,1\,1] + [0\,1\,0] = [0\,1\,1].$$

Z tabeli 2.5.2 wynika, że syndrom ten odpowiada wektorowi błędu $\boldsymbol{e} = [0\,1\,0\,0\,0\,0]$. Po korekcji wektora odebranego otrzymamy następujący wektor wyjściowy dekodera

$$\boldsymbol{c}_D = \boldsymbol{c}_Y + \boldsymbol{e} = [1\,0\,1\,0\,0\,1\,0] + [0\,1\,0\,0\,0\,0\,0] = [1\,1\,1\,0\,0\,1\,0].$$

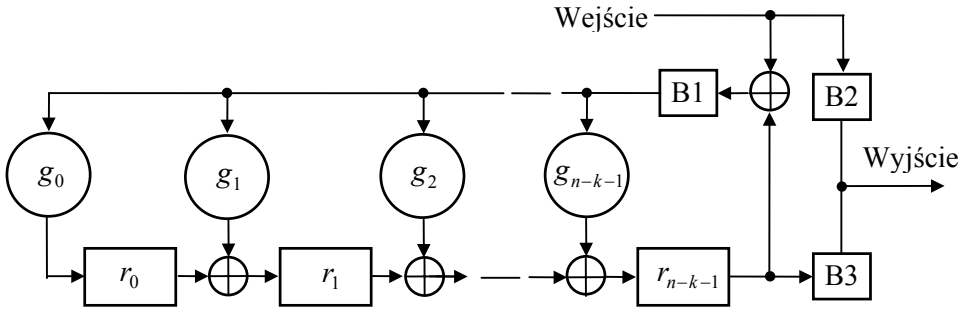
Informacja odebrana jest częścią informacyjną tego wektora, to jest $m^* = 1\,1\,1\,0$.

2.6. Realizacja techniczna koderów i dekoderów kodów cyklicznych

Kodery i dekodery kodów cyklicznych nad ciałami charakterystyki dwa można zbudować za pomocą układów logicznych. Układy kodera i dekodera zawierają rejestry przesuwne oraz sumatory, układy mnożące i bramki. Oznaczenia tych elementów podano na rys. 1.4.1.

2.6.1. Realizacja kodera

Algorytm kodowania kodów cyklicznych opiera się na dzieleniu wielomianów. Układ dzielenia wielomianów opisano w p. 1.7.3. Stanowi on podstawę do konstrukcji kodera.



Rys. 2.6.1. Schemat blokowy kodera

Niech wielomianem generującym kod $g(x)$ będzie wielomian w postaci ogólnej

$$g(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \ldots + g_2x^2 + g_1x + g_0 \quad \text{nad} \quad GF(2).$$

Schemat blokowy kodera odpowiadający temu wielomianowi pokazano na rys. 2.6.1. Zbudowany on jest z rejestru przesuwnego ze sprzężeniem zwrotnym, obliczającego część kontrolną wektora kodowego, oraz układu bramek do sterowania przepływem danych w układzie szeregowej transmisji danych. Na wejście kodera podawana jest informacja ze źródła danych, a jego wyjście łączy się z kanałem transmisyjnym. Rejestr zawiera $n - k$ przerzutników, a konfigurację sprzężenia zwrotnego określa wielomian generujący kod $g(x)$.

W pracy kodera można wyróżnić dwa etapy:

Etap 1. Transmisja części informacyjnej wektora kodowego i obliczanie jego części kontrolnej.

W czasie transmisji elementów informacyjnych wektora kodowego bramki B1 i B2 są otwarte (przewodzą), a bramka B3 jest zamknięta. W ten sposób wejście kodera jest połączone z jego wyjściem. W ciągu k taktów zegarowych elementy informacyjne przechodzą ze źródła danych do kanału transmisyjnego i jednocześnie są podawane na wej-

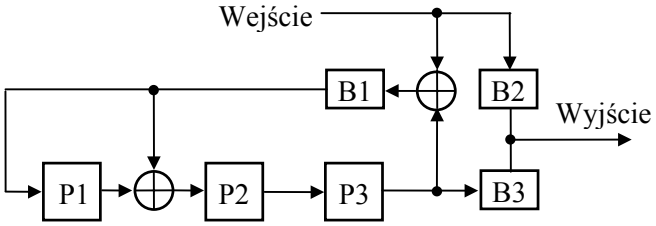
ście układu obliczającego część kontrolną wektora kodowego. Po k taktach zegarowych w rejestrze będzie zapisana obliczona część kontrolna wektora kodowego. Stan ten pokazano na rys. 2.6.1.

Etap 2. Transmisja części kontrolnej wektora kodowego.

Po transmisji elementów ciągu informacyjnego bramki B1 i B2 zostają zamknięte, a bramka B3 – otwarta. W ten sposób wyjście kodera jest odłączone od jego wejścia i przyłączone do wyjścia rejestru, a pętla sprzężenia zwrotnego zostaje przerwana. W następnych $n-k$ taktach zegarowych elementy kontrolne wektora kodowego zostaną przesunięte na wyjście rejestru i przesłane do kanału transmisyjnego.

P r z y k ł a d 2.6.1.

Realizacja kodera kodu Hamminga $(7,4)$ opisanego w przykładzie 2.4.3.



Rys. 2.6.2. Koder kodu Hamminga (7,4)

Schemat blokowy kodera kodu Hamminga $(7,4)$ pokazano na rys. 2.6.2. Wielomianem generującym kod jest wielomian: $g(x) = x^3 + x + 1$. Działanie tego kodera opisano w tabeli 2.6.1. Działanie rejestru ze sprzężeniem zwrotnym jest podobne do działania układu dzielącego pokazanego na rys. 1.7.4.

Tabela.2.6.1. Stany elementów kodera z rys. 2.6.2

Etap	T	We.	P1	P2	P3	Wy.
1	1	1	0	0	0	1
	2	1	1	1	0	1
	3	0	1	0	1	0
	4	1	1	0	0	1
2	5		1	0	0	0
	6		0	1	0	0
	7		0	0	1	1

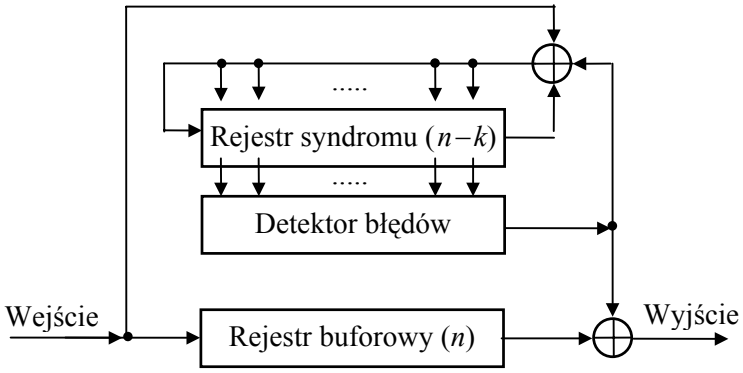
W kolejnych kolumnach tabeli podano: impulsy taktujące T, sygnały wejściowe, stany przerzutników rejestru i sygnały wyjściowe. Wiersze tabeli zawierają stany przerzutników po impulsie taktującym.

2.6.2. Realizacja dekodera

Dekoder kodu cyklicznego wykonuje następujące czynności:

- oblicza syndrom wektora odebranego,
- identyfikuje korygowalne błędy odpowiadające obliczonemu syndromowi,
- koryguje błędy.

Ogólny schemat blokowy dekodera kodu cyklicznego (n,k) pokazano na rys. 2.6.3. Dekoder ten jest dostosowany do układu szeregowego transmisji danych i nazywany dekoderem Meggitt'a.



Rys. 2.6.3. Schemat blokowy dekodera

Dekoder kodu cyklicznego zawiera trzy główne części.

- Układ do obliczania syndromu wektora odebranego.

Układ ten jest zbudowany podobnie do kodera. Rejestr syndromu zawiera $n-k$ przerzutników, a konfigurację sprzężenia zwrotnego określa wielomian generujący kod $g(x)$.

- Detektor błędów.

Detektor błędów wykrywa błędy korygowalne pojawiające się w rejestrze syndromu i generuje sygnały korekcyjne. Detektor jest logicznym układem kombinacyjnym zbudowanym z bramek lub sieci oporników. Projektuje się go tak, że na jego wyjściu pojawi się jedynka wtedy i tylko wtedy, gdy w rejestrze syndromu wystąpi ciąg o wadze $w(s) \le t$, co umożliwia korekcję błędu.

- Rejestr buforowy.

Rejestr buforowy zawiera n przerzutników i powoduje opóźnienie czasowe wektora kodowego niezbędne do obliczenia syndromu i korekcji błędów w szeregowym systemie transmisji.

W pracy dekodera można wyróżnić dwa etapy.

Etap 1. Wprowadzenie wektora kodowego do rejestru buforowego i obliczenie syndromu.

Podczas n cykli zegarowych elementy wektora kodowego są wprowadzane do rejestru buforowego i jednocześnie podawane na wejście układu obliczającego syndrom

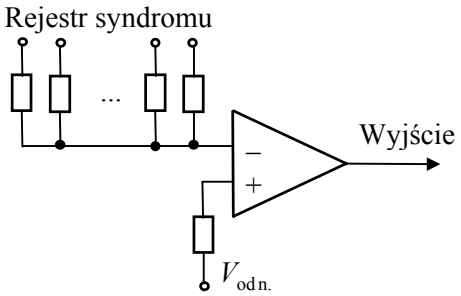
wektora odebranego. W układzie obliczania syndromu następuje dzielenie wektora odebranego przez wielomian generujący kod $g(x)$. W przypadku braku błędów transmisyjnych wektor odebrany dzieli się bez reszty przez wielomian generujący i stan rejestru będzie zerowy. Jeśli wystąpią błędy transmisyjne, to rejestr będzie zawierał niezerowy syndrom wektora odebranego.

Etap 2. Transmisja szeregowa wektora kodowego z rejestru buforowego i korekcja błędów.

Sygnał korekcji jest generowany w detektorze błędów i podawany na sumator wyjściowy oraz do układu obliczania syndromu. W sumatorze wyjściowym sygnał korekcji sumuje się z elementami wektora kodowego, przesyłanego od ujścia danych, i jeśli sygnałem korekcji jest jedynka, następuje korekcja elementu wektora kodowego. Jednocześnie sygnały korekcji są przekazywane do układu obliczania syndromu w celu jego modyfikacji.

Operacje te powtarzają się dla każdego elementu wektora kodowego przesyłanego z dekodera do układu ujścia danych. Po n cyklach zegarowych, kiedy wszystkie elementy wektora kodowego opuszczą rejestr buforowy, rejestr syndromu powinien być wyzerowany. Niezerowy stan rejestru syndromu świadczy o wystąpieniu błędów niekorygowalnych w wektorze odebranym. Proces korekcji można ograniczyć do k elementów informacyjnych wektora kodowego.

Wyżej opisany dekoder można stosować do dowolnych kodów cyklicznych. O jego użyteczności decyduje układ detekcji błędów. Detektor błędów może być wykonany w postaci logicznego układu kombinacyjnego. Dla kodów korygujących błędy wielokrotne liczba kombinacji wykrywanych przez detektor błędów jest duża i taka konstrukcja staje się nieekonomiczna. Stosuje się wtedy dekoder, wykonany z sieci oporników, pokazany na rys. 2.6.4.



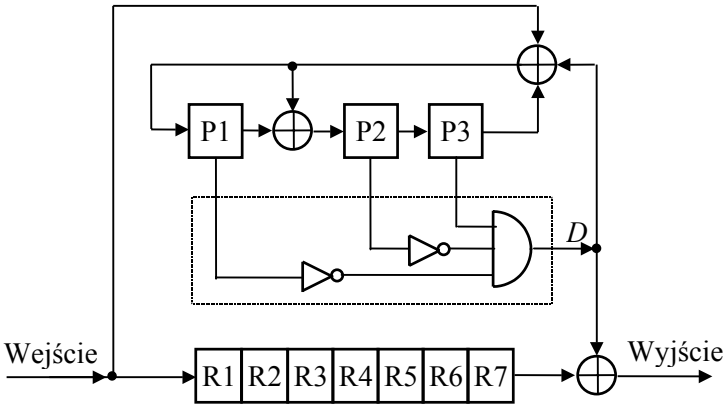
Rys. 2.6.4. Konstrukcja detektora błędów

Gdy waga syndromu $w(s) \le t$, wtedy t lub mniejsza liczba przerzutników rejestru syndromu jest w stanie 1 i wyjście wzmacniacza operacyjnego jest również w stanie 1. W przeciwnym przypadku stan wyjścia jest 0. Wartość t ustala się napięciem $V_{\text{odn.}}$. Gdy kod koryguje więcej niż jeden błąd, konieczna jest modyfikacja dekodera poka-

zanego na rys. 2.6.3. Dekoder, który może być stosowany w urządzeniach rzeczywistych, będzie omówiony w następnym punkcie.

P r z y k ł a d 2.6.2.

Realizacja dekodera cyklicznego kodu Hamminga (7,4) opisanego w przykładzie 2.4.4.



Rys. 2.6.5. Schemat blokowy dekodera kodu Hamminga (7,4)

Tabela 2.6.2. Stany elementów dekodera z rys. 2.6.5

Etap	*T*	We.	R1	R2	R3	R4	R5	R6	R7	P1	P2	P3	*D*	Wy.
1	1	1	0	0	0	0	0	0	0	0	0	0		
	2	1	1	0	0	0	0	0	0	1	1	0		
	3	0	1	1	0	0	0	0	0	1	0	1		
	4	**0**	0	1	1	0	0	0	0	1	0	0		
	5	0	**0**	0	1	1	0	0	0	0	1	0		
	6	0	0	**0**	0	1	1	0	0	0	0	1		
	7	1	0	0	**0**	0	1	1	0	1	1	0		
2	8		1	0	0	**0**	0	1	1	1	0	1	0	1
	9		0	1	0	0	**0**	0	1	1	0	0	0	1
	10		0	0	1	0	0	**0**	0	0	1	0	0	0
	11		0	0	0	1	0	0	**0**	0	0	1	1	1
	12		0	0	0	0	1	0	0	0	0	0	0	0
	13		0	0	0	0	0	1	0	0	0	0	0	0
	14		0	0	0	0	0	0	1	0	0	0	0	1

Schemat blokowy kodera pokazano na rys. 2.6.5, a jego działanie opisano w tabeli 2.6.2. Na rys. 2.6.5 detektor błędów przedstawiono w formie schematu logicznego. Sygnał wyjściowy detektora błędów jest równy $\overline{\mathrm{P1}} \cdot \overline{\mathrm{P2}} \cdot \mathrm{P3}$.

W tabeli 2.6.2 kolejne kolumny pokazują kolejno sygnały: taktujące *T*, wejściowe dekodera, stany przerzutników rejestru buforowego i rejestru syndromu, sygnał wyjściowy układu detektora błędów *D* i sygnał wyjściowy dekodera. Wektor wejściowy zawiera błąd na czwartej pozycji. Bit błędny oznaczono czcionką pogrubioną. W drugim etapie pracy dekodera błąd ten jest skorygowany w takcie jedenastym w wyniku sumowania się sygnałów z rejestru buforowego i detektora błędów.

2.6.3. Dekodowanie z łowieniem błędów

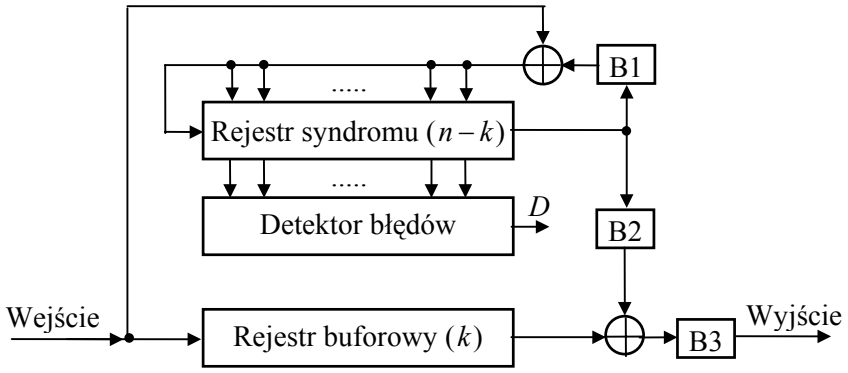
Dekoder pokazany na rys. 2.6.4 w zastosowaniach praktycznych wymaga modyfikacji polepszających jego efektywność. Jedna z praktycznych realizacji dekodera Meggitt'a nosi nazwę *dekodera z łowieniem błędów* (error-trapping decoder). Załóżmy, że stosowany jest kod (n,k), korygujący t błędów w wektorze o strukturze opisanej wzorem (2.4.4)

$$\boldsymbol{c}_X = \left[m_{n-1}, \ldots, m_{n-k}, r_{n-k-1}, \ldots, r_0\right].$$

Z wzoru (2.4.7) wiadomo, że syndrom jest resztą z dzielenia wielomianu $e(x)$, odpowiadającego wektorowi błędów, przez wielomian generujący kod $g(x)$

$$e(x) = q(x)\, g(x) + s(x).$$

Jeśli błąd znajduje się w części kontrolnej wektora kodowego obejmującej $n-k$ pozycji, to wielomian $e(x)$ jest wielomianem stopnia $\leq n-k-1$ i wówczas $q(x)=0$ oraz $e(x)=s(x)$, co oznacza identyczność syndromu z wektorem błędów. W takim przypadku korekcja polega na dodaniu syndromu do części kontrolnej wektora kodowego.



Rys. 2.6.6. Schemat blokowy dekodera z łowieniem błędów

Załóżmy, że błędy nie pokrywają się z częścią kontrolną wektora kodowego, ale zajmują $n-k$ pozycji. Jeśli po i przesunięciach cyklicznych pozycje błędne znajdą się w części kontrolnej wektora kodowego, to możliwa jest ich korekcja wyżej opisaną metodą. Metoda korekcji z łowieniem błędów bazuje na tej procedurze.

Układ dekodera z łowieniem błędów pokazano na rys. 2.6.6. Dekoder ten zawiera dodatkowo układ bramek do sterowania przepływem informacji. Sygnały sterujące bramkami są generowane na podstawie sygnału detektora błędów D. Sygnał D ma wartość 1, gdy $n-k$ wyjść rejestru syndromu ma wartość 1, a w przeciwnym przypadku D ma wartość 0. Ponieważ dla odbiorcy informacji ważna jest tylko część informacyjna wektora kodowego, rejestr buforowy będzie zawierał tylko k przerzutników. Dekoder ma ponadto licznik przesunięć rejestru syndromu, nie pokazany na rysunku 2.6.6.

Tabela 2.6.3. Wymagane kroki korekcji w zależności od położenia sekwencji błędnej

Wymagane kroki korekcji	Część informacja wektora kodowego	Część kontrolna
1+2		
1+2+3		
1+2+3+4		
1+2+3+4+5		

Proces korekcji błędów w dekoderze z łowieniem błędów jest bardziej złożony. Można go opisać za pomocą pięciu kroków korekcji. Jednak liczba kroków korekcji, które należy wykonać w celu skorygowania konkretnego wektora kodowego, zależy od położenia sekwencji błędnej w stosunku do wektora kodowego. Ilustruje to tabela 2.6.3. W tabeli tej wektorom odebranym odpowiadają wiersze tablicy, a obszary zaciemnione oznaczają sekwencje z błędami.

W poszczególnych krokach wykonuje się niżej opisane czynności.

Krok 1. Na początku pracy dekodera bramka B1 jest otwarta (przewodzi), a bramki B2 i B3 zamknięte. Podczas n cykli zegarowych elementy wektora kodowego są podawane na wejście układu obliczającego syndrom wektora odebranego i jednocześnie k elementów informacyjnych zostaje wprowadzonych do rejestru buforowego.

Krok 2. Po wprowadzeniu całego wektora kodowego do rejestru syndromu układ detektora błędów sprawdza wagę syndromu. Mogą wówczas wystąpić dwa przypadki.

- Jeżeli waga syndromu $w(s) \le t$, to błędy znajdują się w części kontrolnej wektora kodowego, a część informacyjna nie zawiera błędów. Bramka B3 zostaje otwarta, a część informacyjna wektora kodowego z rejestru buforowego jest przesłana na wyjście dekodera i w ten sposób proces dekodowania zostanie zakończony. Dekoder wraca do kroku 1.
- Jeżeli waga syndromu $w(s) > t$, oznacza to, że błędy obejmują część informacyjną wektora kodowego. Dekoder przechodzi do kroku 3.

Krok 3. Bramki pozostają w takich samych stanach jak w kroku 1, tzn. że bramka B1 jest otwarta, a B2 i B3 – zamknięte. Zawartość rejestru syndromu zostaje przesunięta w prawo. Po przesunięciu zawartości rejestru syndromu sprawdza się wagę syndromu.

- Jeśli waga syndromu $w(s) \le t$, oznacza to, że prawy bit części informacyjnej wektora kodowego m_{n-k} jest błędny. Lewy skrajny bit rejestru syndromu wskazuje na ten błąd. Stan taki ilustruje wiersz ósmy tabeli 2.6.2. Detektor błędów zamyka bramkę B1, a licznik syndromu ustawia w pozycji 2. Zawartość rejestru syndromu jest przesuwana w takt impulsów zegarowych. Kiedy licznik syndromu osiągnie stan $n-k$, stan rejestru syndromu będzie (0, 0, ..., 0, 1). Wtedy następuje otwarcie B2 i B3 a informacja zawarta w rejestrze buforowym zostaje przesunięta wraz z zawartością rejestru syndromu i transmitowana do urządzeń wyjściowych. Pierwszy element ciągu informacyjnego opuszczającego rejestr buforowy będzie skorygowany. Po k taktach zegarowych transmisja zostanie zakończona. Dekoder wraca do etapu 1.
- Jeśli waga syndromu $w(s) > t$, to dekoder przechodzi do kroku 4.

Krok 4. Zawartość rejestru syndromu jest przesuwana przy otwartej bramce B1, aż zostanie osiągnięta waga syndromu $w(s) \le t$. Jeśli stan ten będzie osiągnięty po i przesunięciach, gdzie $1 \le i \le n-k$, to licznik przesunięć rejestru syndromu startuje od stanu $i+1$ i umożliwia przesuwanie zawartość rejestru syndromu, aż osiągnie on stan $n-k$. Wówczas i bitów, znajdujących się z prawej strony w rejestrze syndromu, wskazuje na błędy zawarte w i bitach prawej części ciągu informacyjnego. Pozostałe elementy informacji są bezbłędne. Wtedy bramka B1 zostaje zamknięta, a bramki B2 i B3 – otwarte. Informacja zawarta w rejestrze buforowym jest transmitowana na zewnątrz i korygowana sygnałami rejestru syndromu.

Krok 5. Jeśli waga syndromu nie osiągnie wartości $w(s) \le t$ po $n-k$ przesunięciach rejestru syndromu przy otwartej bramce B1, to bramka B3 zostaje otwarta, a informacja z rejestru buforowego jest transmitowana na zewnątrz. Jednocześnie jest przesuwana zawartość rejestru syndromu przy otwartej bramce B1. Jeśli waga syndromu osiągnie wartość $w(s) \le t$, to bramka B2 zostanie otwarta a bramka B1 – zamknięta. Rejestry syndromu i buforowy są przesuwane, a elementy opuszczające rejestr buforowy korygowane sygnałami z rejestru syndromu. Kiedy ostatni element informacji opuści rejestr buforowy, bramka B3 zostanie zamknięta. Jeżeli waga syndromu nigdy nie będzie $w(s) \le t$, oznacza to, że wystąpiły błędy niekorygowalne.

Dekodowanie z łowieniem błędów jest algorytmem efektywnym w przypadku błędów pojedynczych, grupowych i krótkich kodów korygujących błędy podwójne, ale nie zdaje egzaminu przy długich kodach i kodach z dużą liczbą błędów korygowalnych. Wówczas stosuje się inne algorytmy opisane w [2.4, 2.5].

2.7. Przegląd binarnych kodów cyklicznych

2.7.1. Cykliczne kody Hamminga

Kod cykliczny nazywamy kodem Hamminga, jeżeli jego wielomian generujący jest wielomianem pierwotnym. Cykliczny kod Hamminga (n,k) generowany przez wielomian stopnia m ma następujące parametry:

- długość wektora kodowego $n = 2^m - 1,$
- liczba pozycji informacyjnych $k = 2^m - m - 1,$
- odległość minimalna $d = 3,$
- zdolność korekcyjna $t = 1.$

Kody Hamminga są kodami doskonałymi, a ich przykłady podano w rozdziałach 2.4÷2.6.

Kody Hamminga można rozszerzyć mnożąc wielomian pierwotny przez czynnik $x+1$

$$g(x) = (x+1)p(x). \tag{2.7.1}$$

Utworzony w ten sposób kod jest cykliczny i charakteryzuje się następującymi parametrami: $n = 2^m - 1,\ k = 2^m - m - 2,\ d = 4,\ t = 1.$ Kod ten może korygować jeden błąd i wykrywać dwa błędy.

Niech $c_Y(x)$ będzie ciągiem odebranym. Dla takiego ciągu można obliczyć dwa syndromy. Dzieląc $c_Y(x)$ przez $p(x)$ i przez $x+1,$ otrzymamy odpowiednio:

$$\begin{aligned} c_Y(x) &= q_1(x)p(x) + s_p(x), \\ c_Y(x) &= q_2(x)(x+1) + s_a, \end{aligned} \tag{2.7.2}$$

gdzie $s_p(x)$ jest wielomianem stopnia $m-1$ lub niższego, a s_a może mieć wartość 0 lub 1.

Gdy $s_p(x) = 0$ i $s_a = 0,$ wtedy odebrany wektor dzieli się przez $(x+1)p(x)$ i jest wektorem kodowym. W przeciwnym przypadku wektor odebrany nie będzie wektorem kodowym.

Procedura dekodowania rozszerzonego kodu Hamminga jest następująca:

1. Dla $s_p(x) = 0$ i $s_a = 0$ dekoder nie wykrywa błędów i nie podejmuje procedury korekcyjnej.

2. Dla $s_p(x) \neq 0$ i $s_a = 1$ dekoder przyjmuje wystąpienie pojedynczego błędu i koryguje wektor odebrany według normalnej procedury stosowanej dla kodu Hamminga.

3. Dla $s_p(x) \neq 0$ i $s_a = 0$ dekoder sygnalizuje wystąpienie błędu podwójnego.

4. Dla $s_p(x)=0$ i $s_a=1$ dekoder sygnalizuje prawdopodobieństwo wystąpienie błędów nieparzystych.

Opisana wyżej procedura rozszerzania kodu może być zastosowana dla każdego kodu cyklicznego. Kody Hamminga mogą być uogólnione dla ciał niebinarnych [2.4].

2.7.2. Kody maksymalnej długości

Kody maksymalnej długości (maximum-length codes) są kodami dualnymi kodów Hamminga. Kody maksymalnej długości istnieją dla każdej liczby całkowitej $m \geq 2$. Mają one następujące parametry:

- długość wektora kodowego $n = 2^m - 1$,
- liczba pozycji informacyjnych $k = m$,
- odległość minimalna $d = 2^{m-1}$.

Wielomiany generujące kody maksymalnej długości obliczamy z zależności

$$g(x) = \frac{x^n + 1}{p(x)}, \tag{2.7.3}$$

gdzie $p(x)$ jest wielomianem pierwotnym stopnia m.

P r z y k ł a d 2.7.1.

Kod maksymalnej długości dla $m=4$ i $p(x) = x^4 + x + 1$.

Dla przyjętych parametrów można skonstruować kod o długości wektora kodowego $n=15$ i $d=8$. Wielomian generujący kod będzie następujący

$$g(x) = \frac{x^{15} + 1}{p(x)} = x^{11} + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1.$$

2.7.3. Kody Bose–Chaudhuri–Hocquenghema

Kody Bose–Chaudhuri–Hocquenghema (BCH) należą do kodów korygujących błędy losowe i mają duże znaczenie praktyczne. Zostały one niezależnie skonstruowane przez Hocquenghema w 1959 r. oraz przez Bose z Chaudhurim w 1960 r. Kody BCH swoją popularność zawdzięczają następującym zaletom:

- Istnieją efektywne metody konstruowania kodów BCH o zadanych właściwościach detekcyjnych i korekcyjnych.
- Konstrukcja koderów i dekoderów kodów BCH jest prostsza niż dla innych kodów.

Kody BCH można konstruować nad ciałem binarnym i ciałami rozszerzonymi. Największe znaczenie mają kody binarne. Udowodniono, że dla każdej liczby całkowitej m i $t < 2^{m-1}$ istnieje kod BCH o długości $n = 2^m - 1$. Może on korygować do t błędów i ma nie więcej niż mt elementów kontrolnych. Kody te mają następujące parametry:

- długość wektora kodowego $n = 2^m - 1$,
- liczba pozycji kontrolnych $n - k \le m\,t$,
- odległość minimalna $d \ge 2\,t + 1$.

Wielomiany generujące kody BCH wyznacza się w następujący sposób. Niech α będzie elementem pierwotnym ciała $GF(2^m)$. Zbiór $\{f(x)\}$ jest zbiorem ciągów kodowych kodu BCH, jeśli pierwiastkami dowolnie wybranego wielomianu $f(x)$ są elementy ciała

$$\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t}.$$

Każdy element ciała o parzystym wykładniku ma w tej sekwencji taką samą funkcję minimalną jak któryś z poprzedzających go elementów o wykładniku nieparzystym. Na przykład α^2 i α^4 są pierwiastkami $m_1(x)$, α^6 jest pierwiastkiem $m_3(x)$ itd. Uwzględniając ten fakt podczas wyznaczania wielomianu generującego kod BCH, wystarczy wziąć pod uwagę elementy ciała z wykładnikami nieparzystymi.

Wielomian generujący kod BCH o zdolności korekcyjnej *t* jest najmniejszą wspólną wielokrotnością funkcji minimalnych $m_1(x), m_3(x), \ldots, m_{2t-1}(x)$

$$g(x) = NWW\big(m_1(x), m_3(x), \ldots, m_{2t-1}(x)\big). \tag{2.7.4}$$

P r z y k ł a d 2.7.2.

Wyznaczanie wielomianów generujących kody BCH.

W p. 1.5.8 pokazano, że dla $m = 4$ dwumian $x^{q^m-1} - 1$ ma następujący rozkład

$$x^{15} + 1 = (x+1)\left(x^4 + x + 1\right)\left(x^4 + x^3 + x^2 + x + 1\right)\left(x^2 + x + 1\right)\left(x^4 + x^3 + 1\right).$$

Na podstawie informacji podanych w tabeli 1.5.5 wiadomo, że wielomiany nierozkładalne z prawej strony znaku równości są wielomianami minimalnymi elementów ciała $GF(2^4)$. Podstawiając symbole wielomianów minimalnych, otrzymamy

$$x^{15} + 1 = m_0(x)\, m_1(x)\, m_3(x)\, m_5(x)\, m_7(x).$$

Korzystając z tego wyrażenia, dla zadanych wartości *t* można wyznaczyć z (2.7.4) wielomiany generujące kody BCH.

$t = 1$, $g(x) = m_1(x)$, kod Hamminga (15,11);

$t = 2$, $g(x) = m_1(x)\, m_3(x)$, kod (15,7);

$t = 3$, $g(x) = m_1(x)\, m_3(x)\, m_5(x)$, kod (15,5).

Po prawej stronie wielomianów generujących podano parametry kodów (n,k). Na przykład dla $t=2,\ (n,k)=(15,7)$. Aby obliczyć liczbę pozycji informacyjnych k wektora kodowego, należy wyznaczyć stopień wielomianu generującego. Dla kodu $(n,k)=(15,7)$ otrzymamy wielomian generujący ósmego stopnia

$$g(x)=\left(x^4+x+1\right)\left(x^4+x^3+x^2+x+1\right)=x^8+x^7+x^6+x^4+1.$$

Wektor kodowy będzie zatem zawierał osiem pozycji kontrolnych i siedem informacyjnych. Odległość minimalna tego kodu jest $d\geq 5$ i może on korygować dwa błędy.

Dla $t=1$ kod BCH ma wielomian generujący

$$g(x)=m_1(x)=x^4+x+1\,.$$

Kod BCH korygujący jeden błąd jest jednocześnie kodem Hamminga. Generalnie kody Hamminga są podzbiorem kodów BCH.

Wyżej opisany sposób konstrukcji kodów BCH umożliwia utworzenie tak zwanych kodów BCH w „wąskim sensie". Istnieją jeszcze inne metody konstrukcji kodów BCH. Parametry kodów BCH dla kilku wartości *m* podano w tabeli 2.7.1. Większe tablice kodów BCH znajdują się w [2.4].

Tabela 2.7.1. Parametry kodów BCH

m	*n*	*k*	*t*	*m*	*n*	*k*	*t*	*m*	*n*	*k*	*t*	*m*	*n*	*k*	*t*
3	7	4	1	6	63	10	13	7	127	15	27	8	255	123	19
4	15	11	1			7	15			8	31			115	21
		7	2	7	127	120	1	8	255	247	1			107	22
		5	3			113	2			239	2			99	23
5	31	26	1			106	3			231	3			91	25
		21	2			99	4			223	4			87	26
		16	3			92	5			215	5			79	27
		11	5			85	6			207	6			71	29
		6	7			78	7			199	7			63	30
6	63	57	1			71	9			191	8			55	31
		51	2			64	10			187	9			47	42
		45	3			57	11			179	10			45	43
		39	4			50	13			171	11			37	45
		36	5			43	14			163	12			29	47
		30	6			36	15			155	13			21	55
		24	7			29	21			147	14			13	59
		18	10			22	23			139	15			9	63
		16	11							131	18				

W praktyce często występuje potrzeba obliczenia wielomianu generującego wybranego kodu BCH. Metodę rozwiązania takiego zadania pokazuje przykład 2.7.3.

P r z y k ł a d 2.7.3.

Obliczanie wielomianów generujących kody BCH nad $GF(2^6)$ na podstawie tablic wielomianów nierozkładalnych.

Z tabeli 1.5.6 mamy następujący zbiór wielomianów minimalnych nad $GF(2^6)$:

$$m_1(x) = x^6 + x + 1, \qquad m_3(x) = x^6 + x^4 + x^2 + x + 1,$$
$$m_5(x) = x^6 + x^5 + x^2 + x + 1, \qquad m_7(x) = x^6 + x^3 + 1,$$
$$m_9(x) = x^3 + x^2 + 1, \qquad m_{11}(x) = x^6 + x^5 + x^3 + x^2 + 1,$$
$$m_{21}(x) = x^2 + x + 1.$$

Dla każdego z powyższych wielomianów wyznaczamy wielomian odwrotny wraz z jego pierwiastkami. Jednym z pierwiastków wielomianu odwrotnego będzie element ciała: α^{q^m-1-i}, gdzie $m = 6$. Pozostałe pierwiastki oblicza się z szeregu (1.5.9). Najmniejszy wykładnik pierwiastka jest indeksem wielomianu minimalnego. Wyniki tych obliczeń podano w tabeli 2.7.2. W pierwszej kolumnie tej tabeli podano wielomiany nierozkładalne, w drugiej – wielomiany odwrotne, w trzeciej pierwiastki wielomianów odwrotnych, a w ostatniej symbole wielomianów minimalnych.

Tabela 2.7.2. Wielomiany odwrotne $GF(2^6)$

Wielomian	Wielomian odwrotny	Pierwiastek wielomianu odwrotnego	Wielomian minimalny
$m_1(x)$	$x^6 + x^5 + 1$	$\alpha^{62}, \alpha^{61}, \alpha^{59}, \alpha^{55}, \alpha^{47}, \alpha^{31}$	$m_{31}(x)$
$m_3(x)$	$x^6 + x^5 + x^4 + x^2 + 1$	$\alpha^{60}, \alpha^{57}, \alpha^{51}, \alpha^{39}, \alpha^{15}, \alpha^{30}$	$m_{15}(x)$
$m_5(x)$	$x^6 + x^5 + x^4 + x + 1$	$\alpha^{58}, \alpha^{53}, \alpha^{43}, \alpha^{23}, \alpha^{46}, \alpha^{29}$	$m_{23}(x)$
$m_7(x)$	$x^6 + x^3 + 1$	wielomian samoodwrotny	
$m_9(x)$	$x^3 + x + 1$	$\alpha^{54}, \alpha^{45}, \alpha^{27}$	$m_{27}(x)$
$m_{11}(x)$	$x^6 + x^4 + x^3 + x + 1$	$\alpha^{52}, \alpha^{41}, \alpha^{19}, \alpha^{38}, \alpha^{13}, \alpha^{26}$	$m_{13}(x)$
$m_{21}(x)$	$x^2 + x + 1$	wielomian samoodwrotny	

Korzystając z powyższych wyników, możemy napisać rozkład dwumianu $x^{q^m-1} - 1$ na wielomiany nierozkładalne

$$x^{63} + 1 = m_0(x)m_1(x)m_3(x)m_5(x)m_7(x)m_9(x)m_{11}(x)m_{13}(x) \cdot$$
$$\cdot m_{15}(x)m_{21}(x)m_{23}(x)m_{27}(x)m_{31}(x).$$

Z wyrażenia tego otrzymujemy wielomiany generujące kody BCH:

$$t=1,\quad g(x)=m_1(x), \qquad (n,k)=(63,57);$$
$$t=2,\quad g(x)=m_1(x)\,m_3(x), \qquad (n,k)=(63,51);$$
$$t=3,\quad g(x)=m_1(x)\,m_3(x)\,m_5(x), \qquad (n,k)=(63,45);$$
$$t=4,\quad g(x)=m_1(x)\,m_3(x)\,m_5(x)\,m_7(x), \qquad (n,k)=(63,39);$$
$$t=5,\quad g(x)=m_1(x)\,m_3(x)\,m_5(x)\,m_7(x)\,m_9(x), \qquad (n,k)=(63,36);$$
$$t=6,\quad g(x)=m_1(x)\,m_3(x)\,m_5(x)\,m_7(x)\,m_9(x)\,m_{11}(x), \qquad (n,k)=(63,30);$$
$$\ldots$$
$$t=15,\quad g(x)=m_1(x)\,m_3(x)\,m_5(x)\ldots m_{27}(x), \qquad (n,k)=(63,7).$$

Współczynniki wielomianów generujących kody podano w tabeli 2.7.3.

Większość kodów BCH ma odległość minimalną $d \geq 2t+1$. Wartość ta może być zwiększona przez dołączenie 1 do pierwiastków wektora kodowego. Funkcją minimalną 1 jest $x+1$, którą dołączamy do wielomianu generującego. Wówczas wzór określający wielomian generujący kodu będzie

$$g(x)=NWW\left(m_0(x)\,m_1(x), m_3(x), \ldots, m_{2t-1}(x)\right). \tag{2.7.5}$$

Wszystkie ciągi kodowe tego kodu mają parzystą liczbę jedynek.

Inną ważną podklasą kodów BCH są kody *Reeda-Solomona*. Kodów tych używa się do korygowania błędów grupowych, co zostanie dokładnie omówione w następnym rozdziale.

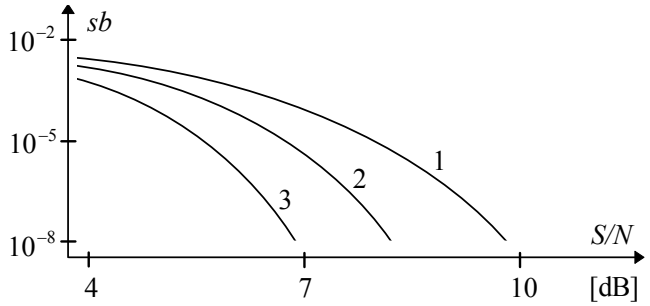
Do kodowania i dekodowania kodów BCH można stosować procedury opisane w poprzednich punktach. Jednak uproszczony algorytm dekodowania, podany w p. 2.4.4, nie zapewnia wykorzystania pełnych zdolności korekcyjnych kodów korygujących błędy wielokrotne. Wynika to z faktu, że w przypadku błędów wielokrotnych nie zawsze jest możliwe przesunięcie błędnych elementów do części kontrolnej wektora kodowego. Kody BCH mają własny algorytm dekodowania, który zapewnia korekcję wszystkich błędów nie przewyższających zdolności korekcyjnej danego kodu. Algorytm ten jest jednak bardzo złożony. Jego opis można znaleźć w [2.4, 2.5].

Kody BCH maja dobre właściwości korekcyjne. Udowodniono, że wszystkie kody z tabeli 2.7.1 o długości nie większej niż 15 oraz kody korygujące błędy podwójne są kodami optymalnymi dla symetrycznych kanałów binarnych. Wśród kodów o długości wektora kodowego 63 znaleziono 8000 kodów lepszych od kodów BCH [2.4]. Należą one do wąskiej grupy kodów mających większą odległość minimalną niż odpowiednie kody BCH. Analiza parametrów kodu pokazuje, że ze wzrostem długości wektora kodowego *n* maleje zdolność korekcyjna kodu *t*, gdy stosunek *n/k* jest stały. Świadczy to o tym, że kody BCH są słabymi kodami dla dużych *n*.

Praktyczne skutki zastosowania kodów cyklicznych pokazano na rys. 2.7.1. Przedstawia on wykresy elementowej stopy błędów *sb* w funkcji *S/N*, gdzie *S* jest mocą sygnału, a *N* mocą szumu. W kanale z zakłóceniami gaussowskimi zastosowano modulację częstotliwości FSK (Frequency-Shift Keying).

Na wykresie pokazano charakterystyki dla trzech kanałów:

1. Kanał bez korekcji.
2. Kanał zabezpieczony kodem Hamminga $(15,11)$ o zdolności korekcyjnej $t = 1$.
3. Kanał zabezpieczony kodem BCH $(127,64)$ o zdolności korekcyjnej $t = 10$.



Rys. 2.7.1. Charakterystyki kanału transmisyjnego

Z wykresów pokazanych na rys. 2.7.1 widać, że dla określonego poziomu zakłóceń największa stopę błędów będzie miał kanał bez zabezpieczenia kodowego. Po zastosowaniu kodów korekcyjnych elementowa stopa błędów zmniejsza się. Efektywniejszą poprawę wierności transmisji osiąga się dla kodów o większej zdolności korekcyjnej. Uzasadnia to potrzebę stosowania kodów korekcyjnych.

2.7.4. Tablica kodów cyklicznych

Problem konstrukcji kodów cyklicznych sprowadza się do syntezy wielomianu generującego kod i wyznaczenia jego odległości minimalnej. Zadania te są dość trudne, dlatego też często posługujemy się tablicami zawierającymi wielomiany generujące kody cykliczne i ich parametry.

Wybrany zestaw wielomianów generujących binarne kody cykliczne podano w tabeli 2.3.7. Kolejne kolumny tej tabeli zawierają: długość wektora kodowego *n*, liczbę pozycji informacyjnych *k*, odległość minimalną *d* i wielomian generujący kod *g*(*x*). Obszerniejszy zbiór wielomianów generujących binarne kody cykliczne podano w [2.4].

Wielomiany generujące kody, podane w tabeli 2.3.7, zawierają współczynniki w systemie ósemkowym. Aby na podstawie tabeli wyznaczyć wielomian generujący, należy liczbę ósemkową zamienić na liczbę dwójkową. Na przykład $2467_8 = 10100110111_2$, gdzie współczynnik zmiennej o najwyższej potędze znajduje się z lewej strony ciągu. Stąd wielomianem generującym kod cykliczny $(15,5)$, który koryguje trzy błędy, będzie wielomian

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1.$$

Tabela 2.7.3. Wielomiany generujące binarnych kodów cyklicznych

n	k	d	$g(x)$	n	k	d	$g(x)$	n	k	d	$g(x)$
7	4	3	13	33	20	6	20741	45	22	8	63335065
	3	4	35		13	10	4172741		21	3	110111011
15	11	3	23		12	10	14217043		20	6	330333033
	10	4	65		11	11	25456465	47	24	11	43073357
	7	5	721		10	12	76563537		23	12	145115461
	6	6	1163	35	25	4	2565	49	28	3	10040001
	5	7	2467		24	4	7637		27	4	30140003
	4	8	7531		23	3	13627		21	4	2010040001
17	9	5	727		20	6	147257	51	43	3	433
	8	6	1171		19	6	251761		35	5	266251
21	16	3	61		18	4	735235		33	6	1403537
	15	4	123		17	6	1532051		32	6	2020213
	12	5	1663		16	7	2433361		27	9	134531443
	11	6	2531		15	8	7455423		26	10	242245105
	10	5	5031		11	5	143676743		24	10	1762776477
	9	8	17053		10	10	244303045		19	14	50112257553
	5	10	214537	39	27	3	13617		18	14	170336760675
23	12	7	5343		26	6	34221		10	18	62066722733023
	11	8	17445		25	3	55263	55	35	5	7164555
25	5	5	4102041		24	6	167725		34	8	11235667
	4	10	14306143		15	10	153651205	57	39	3	1341035
27	9	3	1001001		14	10	274373617		38	6	3443047
	8	6	3003003		13	12	423136633	63	57	3	103
31	26	3	45	41	21	9	6647133		56	4	305
	25	4	157		20	10	13351355		51	5	12471
	21	5	3551	43	29	6	64213		45	7	1701317
	16	7	107657		28	6	134635		39	9	166623567
	15	8	310361		15	13	2607043415		36	11	1033500423
	11	11	5423325		14	14	7211144427		30	13	157464165547
	6	15	313365047	45	35	4	2113		24	15	17323260404441
	5	16	535437151		29	5	230213		18	21	1363026512351725
33	23	3	3043		25	5	7217531		16	23	6331141367235453
	22	6	5145		24	6	11620753		10	27	472622305527250155
	21	3	17537		23	7	21113023		7	31	5231045543503271737

2.8. Kody cykliczne korygujące błędy grupowe

2.8.1. Błędy grupowe

Błędy grupowe (burst-errors), zwane również seryjnymi, powstają w kanałach transmisyjnych i pamięciach masowych. W kanałach transmisyjnych błędy grupowe wynikają z działania zakłóceń impulsowych, a w pamięciach – z uszkodzenia nośnika magnetycznego. Błędy grupowe występują w przypadkowych okresach czasu i są rozdzielone sekwencjami bezbłędnymi.

Błędem grupowym nazywamy sekwencję błędów obejmującą b kolejnych pozycji, której pierwszy i ostatni element są elementami niezerowymi. Na przykład wektor błędu $[000010110101000]$ zawiera błąd grupowy o długości osiem. Ponieważ kody korygujące błędy losowe są nieefektywne podczas korekcji błędów grupowych, do korekcji tych błędów skonstruowano kody specjalne. Konstruując takie kody, dążymy do osiągnięcia jak najmniejszej redundancji, to jest najmniejszej liczby pozycji kontrolnych.

Z obserwacji wynika, że liczba elementów kontrolnych wektora kodowego kodu (n,k) przeznaczonego do korekcji b błędów grupowych powinna wynosić przynajmniej $2b$.

$$n-k \geq 2b.$$

Jeśli zależność ta jest spełniona ze znakiem równości, to zostanie osiągnięta górna granica zdolności korekcyjnej kodu korygującego błędy grupowe. Granica ta nazywa się granicą Reigera. Do oceny efektywności kodu stosuje się zależność

$$z = \frac{2b}{n-k}. \tag{2.8.1}$$

Jeśli $z = 1$, to kod nazywa się kodem optymalnym.

W przypadku, gdy kod jest przeznaczony do korygowania grupy zawierającej do b błędów i jednocześnie do wykrywani l błędów, liczba elementów kontrolnych wektora kodowego powinna być przynajmniej $b+l$, to jest

$$n-k \geq b+l \quad \text{gdzie} \quad l > b.$$

Kody korygujące błędy grupowe można podzielić na kody korygujące pojedyncze grupy błędów (single-burst-errors) oraz kody do korekcji błędów grupowych i losowych (burst-and-random-errors). Do pierwszej grupy należą kody Fire, a do drugiej – kody Reeda-Solomona.

2.8.2. Kody Reeda-Solomona

Kody Reeda-Solomona są podklasą niebinarnych kodów BCH. Kod RS (n,k) nad ciałem $GF(q)$ ma następujące parametry:

- długość wektora kodowego $n = q-1$,

- liczba pozycji kontrolnych $r = n - k$,
- odległość minimalna $d = r + 1$.

Zdolność korygowania błędów grupowych kodu RS oblicza się tak jak dla kodów binarnych, tj. z (2.2.6).

Wielomiany generujące kody RS wyznacza się w następujący sposób. Niech α będzie elementem pierwotnym ciała $GF(q)$. Zbiór $\{f(x)\}$ jest zbiorem ciągów kodowych kodu Reeda-Solomona, jeśli pierwiastkami dowolnie wybranego wielomianu $f(x)$ są elementy ciała $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^r$. Wielomian generujący kodu RS nad ciałem charakterystyki dwa przybiera postać

$$g(x) = \prod_{i=1}^{r} \left(x + \alpha^i\right) = (x + \alpha)\left(x + \alpha^2\right)\ldots\left(x + \alpha^r\right). \tag{2.8.2}$$

Wielomian ten ma stopień *r*.

Niech $q = p^m$; wtedy każdy element zbioru *q* jest ciągiem *m*-elementowym nad $GF(p)$. Kod RS korygujący *t* błędów $(q-1, q-1-2t)$ można traktować jako kod $(m(p^m - 1), m(p^m - 1 - 2t))$ nad $GF(p)$. Kod ten może korygować *t* błędów, które pojawią się w bloku zawierającym *m* elementów. Kody korygujące błędy grupowe opierają się na tej idei.

Kodem dualnym dla kodu RS jest również kod RS. Jednak zasada ta generalnie nie jest prawdziwa dla kodów BCH.

Dla kodu RS stosuje się taki sam algorytm kodowania jak dla innych kodów cyklicznych. Pokazuje to przykład 2.8.1.

P r z y k ł a d 2.8.1.

Kodowanie informacji za pomocą kodu RS nad $GF(8)$.

Niech wielomianem generującym rozszerzenie trzeciego stopnia nad *GF*(2) będzie

$$g(x) = x^3 + x + 1.$$

Tabliczki dodawania i mnożenia ciała $GF(8)$ pokazano w tabeli 1.5.2.

Przyjmijmy, że konstruowany kod będzie miał $d = 4$. Kod ten może korygować jeden błąd i jednocześnie wykrywać jeden błąd. Wielomian generujący kodu obliczamy z (2.8.2)

$$g(x) = (x + \alpha)\left(x + \alpha^2\right)\left(x + \alpha^3\right) = x^3 + \alpha^6 x^2 + \alpha\, x + \alpha^6.$$

Jeśli przyjmiemy wielomian informacyjny $m(x) = x^3 + \alpha\, x^2 + x + \alpha$, to współrzędne wektora kodowego kodu RS $(7,4)$ obliczymy, dzieląc $x^3 m(x)$ przez wielomian generujący kod $g(x)$. Dzielenie zapisujemy w postaci współczynników.

$$
\begin{array}{cccc|cccccccc}
 & & & & & & & 1 & \alpha^5 & \alpha^6 & \alpha^6 \\
\hline
1 & \alpha^6 & \alpha & \alpha^6 & 1 & \alpha & 1 & \alpha & 0 & 0 & 0 \\
 & & & & 1 & \alpha^6 & \alpha & \alpha^6 & & & \\
\cline{5-8}
 & & & & & \alpha^5 & \alpha^3 & \alpha^5 & 0 & & \\
 & & & & & \alpha^5 & \alpha^4 & \alpha^6 & \alpha^4 & & \\
\cline{6-9}
 & & & & & & \alpha^6 & \alpha & \alpha^4 & 0 & \\
 & & & & & & \alpha^6 & \alpha^5 & 1 & \alpha^5 & \\
\cline{7-10}
 & & & & & & & \alpha^6 & \alpha^5 & \alpha^5 & 0 \\
 & & & & & & & \alpha^6 & \alpha^5 & 1 & \alpha^5 \\
\cline{8-11}
 & & & & & & & & 0 & \alpha^4 & \alpha^5
\end{array}
$$

Reszta z dzielenia stanowi część kontrolną wektora kodowego. Wektor kodowy otrzymamy po dodaniu części informacyjnej i kontrolnej

$$\boldsymbol{c}_X = [1\alpha\ 1\alpha\ 000] + [00000\alpha^4\alpha^5] = [1\alpha\ 1\alpha\ 0\alpha^4\alpha^5].$$

Algorytm dekodowania kodów RS wyprowadza się z algorytmu dekodowania kodów BCH. Pełny opis tego algorytmu zamieszczono w [2.4, 2.5].

Dla kodu RS można stosować uproszczony algorytm dekodowania opisany w p. 2.4.4. Dekodowanie wektora z błędem za pomocą uproszczonego algorytmu dekodowania pokazano w przykładzie 2.8.2.

P r z y k ł a d 2.8.2.

Dekodowanie wektora z błędem kodu RS $(7,4)$ nad $GF(8)$.

Niech wektorem odebranym będzie wektor z błędem na pozycji piątej, odpowiadający wektorowi kodowemu obliczonemu w przykładzie 2.8.1

$$\boldsymbol{c}_Y = [1\alpha\ 1\alpha\ 1\alpha^4\alpha^5].$$

Zgodnie z procedurą dekodowania opisaną w p. 2.4.4 obliczamy syndrom oraz jego wagę. Syndrom obliczamy, dzieląc wektor odebrany przez wielomian generujący kod, skąd otrzymujemy

$$1\alpha\ 1\alpha\ 1\alpha^4\alpha^5 = \left(1\alpha^5\alpha^6\alpha^6\right)\left(1\alpha^6\alpha\,\alpha^6\right) + 100.$$

Syndrom jest resztą z tego dzielenia i równa się 100. Ponieważ waga syndromu wynosi jeden, można wykonać korekcję wektora odebranego. Dodajemy w tym celu syndrom do wektora odebranego

$$\boldsymbol{c}_D = \left[1\alpha\ 1\alpha\ 1\alpha^4\alpha^5\right] + \left[0000100\right] = \left[1\alpha\ 1\alpha\ 0\alpha^4\alpha^5\right].$$

Otrzymaliśmy wektor skorygowany $[1\alpha\ 1\alpha\ 0\alpha^4\alpha^5]$.

2.8.3. Realizacja techniczna kodów Reeda-Solomona

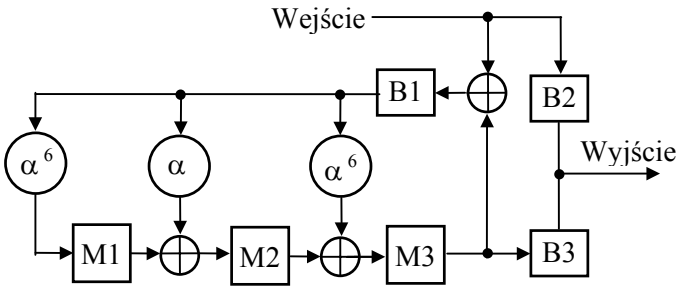
Kody RS są kodami cyklicznymi i zasady konstrukcji koderów i dekoderów tych kodów są takie same jak dla kodów cyklicznych. Realizację techniczną koderów i dekoderów kodów cyklicznych opisano w rozdziale 2.6. Podane tam schematy blokowe mogą być stosowane dla kodów RS.

Ponieważ kody RS są kodami niebinarnymi, operacje wykonywane podczas kodowania czy dekodowania odbywają się na elementach wielostanowych. Konstrukcję układów do realizacji takich operacji opisano w p. 1.7.4. Konstrukcję kodera kodu RS pokazano w przykładzie 2.8.3.

P r z y k ł a d 2.8.3.

Konstrukcja kodera kodu RS $(7,4)$ nad $GF(8)$.

Koder kodu RS $(7,4)$ nad $GF(8)$ przedstawiono na rys. 2.8.1. Opiera się on na schemacie blokowym kodera kodu cyklicznego pokazanym na rys. 2.6.1.



Rys. 2.8.1. Schemat blokowy kodera kodu RS $(7,4)$ nad $GF(8)$

Koder przedstawiony na rys. 2.8.1 realizuje kod RS o wielomianie generującym

$$g(x) = x^3 + \alpha^6 x^2 + \alpha\, x + \alpha^6.$$

Tabela 2.8.1. Stany elementów kodera z rys. 2.8.1

Etap	T	We.	M1	M2	M3	Wy.
1	1	1	0	0	0	1
	2	α	α^6	α	α^6	α
	3	1	α^4	0	α^2	1
	4	α	α^5	α^5	α^5	α
2	5		α^5	α^4	0	0
	6		0	α^5	α^4	α^4
	7		0	0	α^5	α^5

Elementy kodera są elementami ośmiostanowymi. Działanie kodera opisano w tabeli 2.8.1. Kolejne kolumny tabeli zawierają: impulsy taktujące, sygnały wejściowe,

stany elementów rejestru i sygnały wyjściowe. Praca kodera jest podobna do pracy kodera kodów binarnych opisanego w p. 2.6.1.

W pracy kodera można wyróżnić dwa etapy. W etapie pierwszym następuje transmisja części informacyjnej wektora kodowego i obliczanie części kontrolnej. W etapie drugim jest transmitowana na wyjście część kontrolna wektora kodowego.

Dekodery kodów korygujących błędy grupowe można budować, opierając się na dekoderze z łowieniem błędów opisanym w p. 2.6.3. Podczas projektowania dekoderów nad ciałami wieloelementowymi należy uwzględnić zasady konstrukcji elementów wielostanowych podane w p. 1.7.4.

2.8.4. Rozszerzone kody Reeda-Solomona

Cykliczne kody RS można rozszerzyć przez dodanie dwóch elementów do wektora kodowego [2.3]. Powstaje wtedy kod o długości wektora kodowego q+1, gdzie q jest liczbą elementów ciała, nad którym jest konstruowany kod. Te dodatkowe elementy można wykorzystać jako elementy informacyjne lub kontrolne. Istnieje wówczas szansa poprawienia współczynnika sprawności kodu R określonego wzorem (2.2.1).

Rozszerzony kod Reeda-Solomona o zwiększonej liczbie elementów informacyjnych jest cyklicznym kodem rewersyjnym i ma następujące parametry:

- długość wektora kodowego $n = q + 1,$
- liczba pozycji kontrolnych $r = q - d + 2.$

Wielomiany generujące *kody rewersyjne* są wielomianami samoodwrotnymi. Zdolność korygowania błędów grupowych rozszerzonego kodu RS oblicza się tak samo jak dla kodu RS.

Wielomiany generujące rozszerzone kody RS wyprowadza się, wykorzystując rozkład dwumianu $x^{q+1} - 1$ na wielomiany nierozkładalne. Wielomiany generujące rozszerzone kody RS nad $GF(q)$ o odległości Hamminga d mają następującą postać:

- dla ciał charakterystyki dwa $p = 2$:

$$g(x) = \prod_{i=0}^{(d-2)/2} m_{(q-1)i}(x) \quad \text{dla} \quad d = 2,4,6,\ldots,q, \tag{2.8.3}$$

$$g(x) = \prod_{i=(q+3-d)/2}^{q/2} m_{(q-1)i}(x) \quad \text{dla} \quad d = 3,5,7,\ldots,q+1, \tag{2.8.4}$$

- dla innych ciał $p > 2$

$$g(x) = \prod_{i=(q+3-d)/2}^{q+1)/2} m_{(q-1)i}(x) \quad \text{dla} \quad d = 2,4,6,\ldots,q+1, \tag{2.8.5}$$

gdzie $m_{(q-1)i}(x)$ są wielomianami minimalnymi elementów ciała $GF(q^2)$.

Stopień wielomianów generujących, jak również liczba elementów kontrolnych kodu wynosi $d-1$. Nie jest znana metoda obliczania wielomianu generującego o nieparzystej odległości minimalnej d dla ciał, których $p>2$.

Aby wyznaczyć wielomian generujący rozszerzony kod RS, należy:

- skonstruować ciało $GF(q^2)$,
- znaleźć wielomiany minimalne ciała $GF(q^2)$,
- obliczyć wielomian generujący kodu z (2.8.3) lub (2.8.4).

Obliczanie wielomianu generującego rozszerzony kod RS pokazano w przykładzie 2.8.3.

P r z y k ł a d 2.8.3.

Obliczenie wielomianu generującego rozszerzony kod RS nad $GF(4)$ korygujący jeden błąd.

Odległość minimalna kodu korygującego jeden błąd z (2.2.7) wynosi $d=2t+1=3.$ Wektor kodowy rozszerzonego kodu RS będzie miał długość $n=q+1=5,$ a liczba pozycji informacyjnych będzie $k=q-d+2=3.$

Wielomian generujący $g(x)$ rozszerzonego kodu RS $(5,3)$ obliczamy z (2.8.4). Wielomiany nierozkładalne nad $GF(4)$ podano w tabeli 1.6.1.

$$g(x)=\prod_{i=(q+3-d)/2}^{q/2} m_{(q-1)i}(x)=\prod_{i=2}^{2} m_{3i}(x)=m_6(x)=x^2+\alpha^2 x+1.$$

Część 3
Kryptografia

W przeszłości kryptografia była stosowana w dyplomacji i wojsku. Jej podstawy teoretyczne opracował Shannon w 1945 r. [3.11]. Powszechne zainteresowanie tą dziedziną pojawiło się od połowy lat siedemdziesiątych i było związane z rozwojem systemów teleinformatycznych. Prace z zakresu kryptografii, prowadzone w laboratorium IBM, doprowadziły w 1977 r. do powstania pierwszej normy kryptograficznej DES. Monografią, zawierającą pełny przegląd problemów kryptograficznych z bogatym spisem literatury, jest książka [3.9].

3.1. Elementy kryptologii

3.1.1. Ochrona danych

Ochrona danych jest dziedziną wiedzy zajmującą się metodami zabezpieczenia danych przed nielegalnym dostępem i modyfikacją danych w systemach komputerowych i teleinformatycznych. Dane ochrania się, stosując kilka metod. *Kryptografia* jest metodą utajniania danych w wyniku ich przekształcenia metodami matematycznymi w szyfry i właśnie ona będzie głównym tematem tego rozdziału. Sztuka łamania szyfrów nazywa się *kryptoanalizą*, a dziedzinę matematyki obejmująca kryptografię i kryptoanalizę określa się mianem *kryptologii*.

Kiedy dane są gromadzone w pamięciach komputerów i przesyłane w sieciach komputerowych, mogą one być usuwane, modyfikowane i kopiowane. Ochrona danych ma na celu zapobieganie przypadkowemu lub umyślnemu ich zniszczeniu, ujawnieniu lub modyfikacji.

Rozróżnia się dwa zasadnicze przedmioty ochrony:

- poufność lub prywatność chroni dane przed nielegalnym dostępem do nich,
- autentyczność lub integralność jest zabezpieczeniem przed bezprawną modyfikacją danych.

Do ochrony danych stosuje się następujące metody:

- organizacyjne,
- techniczne,
- kryptograficzne.

Do metod organizacyjnych zalicza się metody administracyjne takie jak podział kompetencji personelu, kontrola dostępu i rozliczanie czasu pracy. W głównej mierze to właśnie techniki administracyjne zabezpieczają systemy komputerowe. Wprowadzenie programu ochrony danych powinna poprzedzać analiza zagrożeń danych i określenie środków im przeciwdziałających. Podczas wprowadzania programów ochrony należy zarówno uwzględnić nowe procedury i metody organizacyjne, szkole-

nie i orientowanie się pracowników w zakresie bezpieczeństwa, jak i zapewnić możliwość kontroli.

Zabezpieczenia techniczne obejmują środki fizyczne oraz identyfikację, uwierzytelnienie i upoważnienie personelu. Zabezpieczenia fizyczne takie jak bramy, kraty itp. umożliwiają odgrodzenie obszarów o ograniczonym dostępie oraz kontrolę użytkowników wchodzących do tych obszarów lub wychodzących z nich. Jednak żaden system bezpieczeństwa nie będzie skuteczny, jeśli człowiek zawiedzie zaufanie.

Dobre rozwiązanie problemów bezpieczeństwa systemów komputerowych wymaga zwykle połączenia środków organizacyjnych, technicznych i kryptograficznych. Udział poszczególnych elementów zależy od znaczenia chronionych informacji, rodzaju zagrożenia i dostępnych środków.

Bezpieczeństwo systemu informatycznego powinno być okresowo badane i poprawiane. Chociaż nie można zapewnić absolutnego bezpieczeństwa w dzisiejszych systemach informatycznych, to jednak jest ono tym większe, im wiedza projektantów systemu bezpieczeństwa jest głębsza od wiedzy kryptoanalityka.

3.1.2. Systemy i algorytmy kryptograficzne

Kryptografia jest dziedziną wiedzy zajmującą się *szyfrowaniem*, czyli metodami utajniania treści wiadomości. W wyniku szyfrowania (encryption) *tekst jawny* (plaintext), nazywany też tekstem otwartym (cleartext), zostaje przekształcony w tekst zaszyfrowany zwany *kryptogramem* lub *szyfrogramem* (ciphertext). Proces odtwarzania treści kryptogramu nazywamy *deszyfrowaniem* (decryption). Relacje między tymi pojęciami pokazano na rys. 3.1.1.

Rys. 3.1.1. Szyfrowanie i deszyfrowanie danych

Sztukę łamania szyfrów określamy mianem *kryptoanalizy*, a osoby uprawiające kryptoanalizę nazywa się *kryptoanalitykami*.

Algorytm kryptograficzny, zwany też szyfrem (cipher), jest funkcją matematyczną służącą do szyfrowania i deszyfrowania wiadomości. Do szyfrowania tekstu jawnego stosuje się algorytm szyfrujący (encryption algorithm), a do deszyfrowania kryptogramów algorytm deszyfrujący (decryption algorithm). Zarówno szyfrowanie, jak i deszyfrowanie odbywa się z udziałem *klucza kryptograficznego* (key). Klucze przyjmują wiele wartości, a zbiór tych wartości nazywamy przestrzenią klucza (keyspace).

Algorytmy kryptograficzne wraz z metodą ich implementacji nazywamy *systemem kryptograficznym*. System kryptograficzny zawiera dwa podstawowe elementy:

- algorytm kryptograficzny,
- klucz kryptograficzny.

Przed erą komputerową kryptografia zajmowała się szyframi działającymi na pojedynczych znakach. Były to algorytmy dokonujące podstawienia lub przestawienia znaków. Kryptografię tę określa się obecnie jako *kryptografię klasyczną*.

W zależności od stosowanego algorytmu kryptograficznego systemy kryptograficzne dzieli się na:

- systemy kryptograficzne z kluczem tajnym (secret-key systems),
- systemy kryptograficzne z kluczem jawnym lub publicznym (public-key systems).

W systemach z kluczem tajnym ten sam klucz służy do szyfrowania i deszyfrowania informacji. Systemy takie nazywane są *systemami symetrycznymi*, a stosowane w nich algorytmy *algorytmami symetrycznymi*.

W systemach z kluczem publicznym używa się dwóch oddzielnych kluczy do szyfrowania i deszyfrowania. System taki nosi nazwę *systemu asymetrycznego*. Klucz szyfrujący jest często nazywany *kluczem jawnym* lub publicznym (public key), a klucz deszyfrujący *kluczem tajnym* lub prywatnym (private key).

3.1.3. Właściwości informacyjne języka

Podstawy teoretyczne kryptografii opracowane przez Shannona [3.10] opierają się na stworzonej przez niego teorii informacji. W teorii informacji ilości informacji w wiadomości mierzy się za pomocą entropii (equivocation)

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i),$$

gdzie $p(x_i)$ oznacza prawdopodobieństwo wystąpienia wiadomości x_i.

Entropia jest przeciętną liczbą bitów niezbędną do optymalnego zakodowania wszystkich możliwych wiadomości. Entropia jest również miarą nieokreśloności. Gdy sygnały występują z jednakowym prawdopodobieństwem $p(x_i) = 1/n$, wtedy entropia osiąga maksymalną wartość, która wynosi

$$H(X) = \log_2 n.$$

Zawartość informacyjną języka lub wskaźnik języka (rate of language) r określa się za pomocą przeciętnej liczby bitów informacji na jeden znak.

$$r = H(X) / N,$$

gdzie N jest długością wiadomości. Dla języka angielskiego r wynosi od 1 do 1,5 bita/literę.

Wskaźnik bezwzględny R określa maksymalną ilość informacji, która mogłaby być zakodowana w jednym znaku

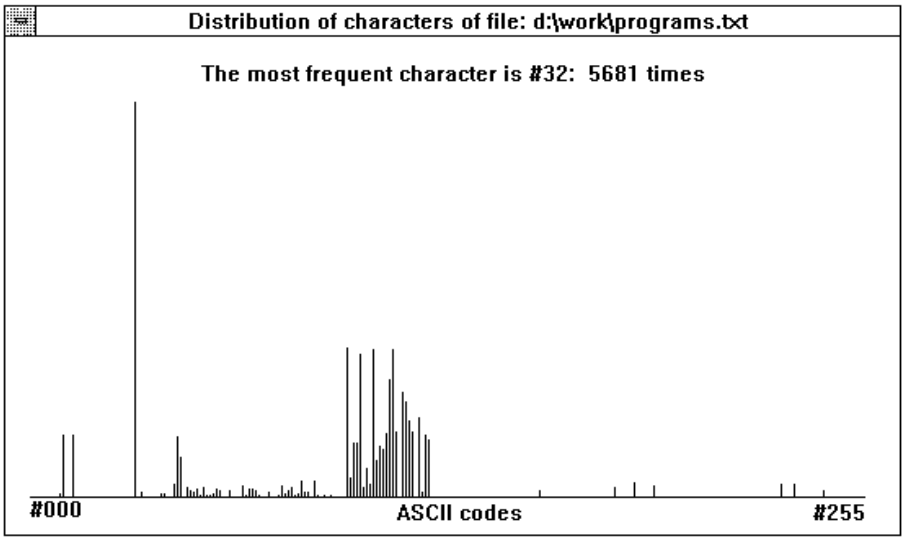
$$R = \log_2 26 = 4{,}7 \text{ bita/literę}.$$

Faktyczny wskaźnik języka jest więc znacznie mniejszy niż jego wskaźnik bezwzględny. Świadczy to o tym, że język angielski ma dużą nadmiarowość lub redundancję. Podobną redundancję mają wszystkie języki naturalne. Tę redundancję D określa się wzorem

$$D = R - r.$$

Jeśli przyjmiemy wartość średnią $r = 1{,}2$ bita/literę, oznacza to, że z ośmiu bitów kodu ASCII tylko 1,2 bita przenosi informację, a reszta jest nadmiarem.

Nadmiarowość języka uwidacznia się we właściwościach statystycznych języka, to jest: w rozkładzie częstotliwości liter, rozkładzie częstotliwości diagramów (par liter), trigramów i *n*-gramów. W różnych językach programowania też można zaobserwować podobieństwo struktury składniowej. Typowy histogram rozkładu znaków w pliku tekstowym, napisanym w języku angielskim, pokazano na rys. 3.1.2. Względną częstość występowania liter, liczb i kilku innych znaków w literackim języku angielskim, polskim i programach w Pascalu pokazano w tabeli 3.3.3.

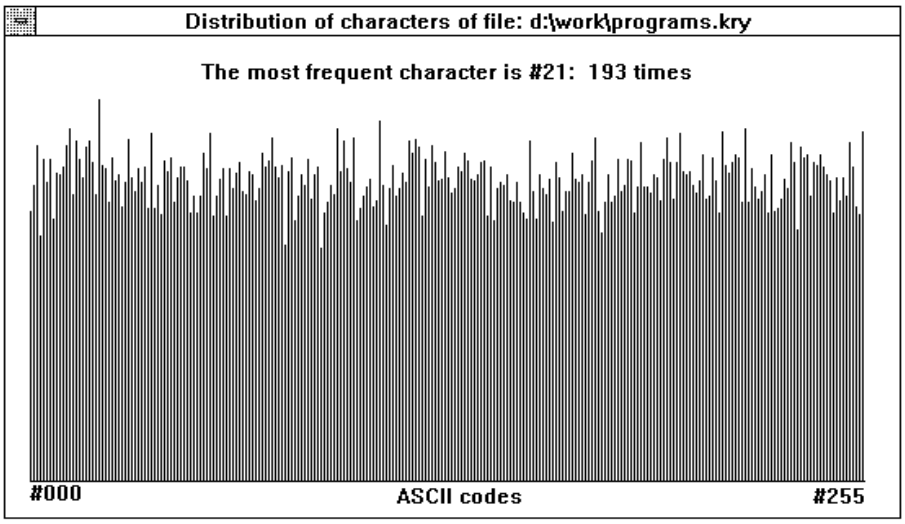


Rys. 3.1.2. Histogram rozkładu znaków w pliku tekstowym

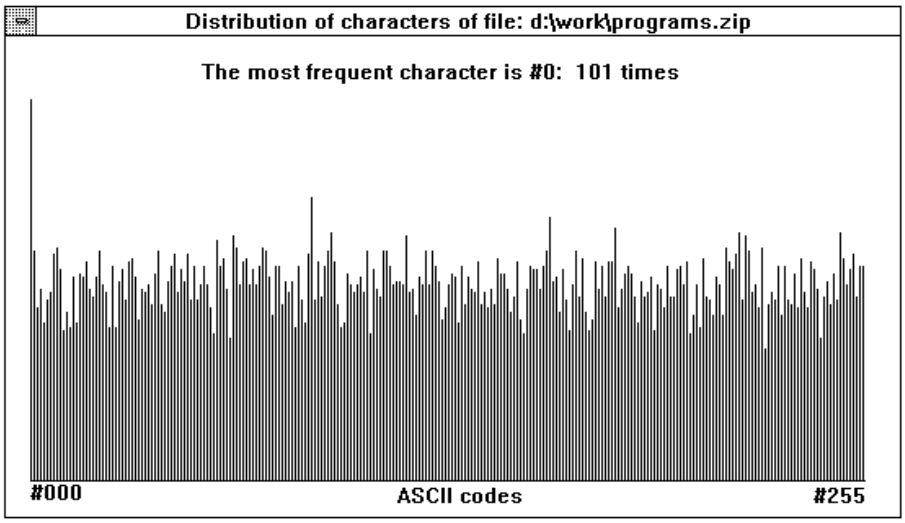
Kryptoanalitycy wykorzystują naturalną nadmiarowość języka do zredukowania liczby możliwych tekstów jawnych lub bezpośrednio do łamania szyfrów za pomocą analizy statystycznej rozkładu znaków i ciągów znaków. Im bardziej nadmiarowy jest język, tym łatwiejsza jest kryptoanaliza szyfrów.

Szyfrowanie tekstów ma na celu, między innymi, usunięcie naturalnych korelacji między znakami tekstu, aby uniemożliwić kryptoanalitykowi zastosowanie analizy

częstotliwości znaków i ciągów znaków. Typowy histogram rozkładu częstości znaków kryptogramu pokazano na rys. 3.1.3.



Rys. 3.1.3. Histogram rozkładu znaków zaszyfrowanego pliku tekstowego



Rys. 3.1.3. Histogram rozkładu znaków pliku tekstowego po kompresji

W celu wzmocnienia szyfrów w wielu rzeczywistych implementacjach systemów kryptograficznych korzysta się z programów kompresji (archiwizatorów), aby zredukować objętość tekstu przed jego zaszyfrowaniem. Kompresja zmniejsza nadmiarowość wiadomości. Dla plików tekstowych współczynnik kompresji ma wartość około

65÷70%. Histogram rozkładu znaków w pliku tekstowym po kompresji za pomocą programu PKZIP.EXE pokazano na rys. 3.1.3.

Niektóre archiwizatory, np. ARJ, PKZIP i RAR, mogą zabezpieczać skompresowane pliki przed nielegalnym dostępem za pomocą hasła. Podczas rozpakowywania archiwum należy bezwzględnie poddać hasło, gdyż w przeciwnym razie pliki nie będą rozpakowane.

3.1.4. Kryptoanaliza

Kryptoanaliza jest nauką o odtwarzaniu tekstu jawnego, gdy nie znamy klucza, albo o odtwarzaniu samego klucza. Kryptoanaliza zajmuje się również wyszukiwaniem słabych punktów kryptogramu, które umożliwią poznanie tekstu jawnego. Działanie kryptoanalityka nazywa się łamaniem szyfru (attack). W literaturze można znaleźć tylko ogólne wskazówki dotyczące metod kryptoanalizy.

Podczas łamania szyfrów można założyć, że kryptoanalityk zna algorytm kryptograficzny, chociaż w rzeczywistości założenie takie nie zawsze jest prawdziwe. Ponieważ szyfry bazujące na tajnych algorytmach są mało bezpieczne, więc odtworzenie algorytmu jest tylko kwestią czasu i pieniędzy. Kryptoanalitycy przeważnie opierają się w swojej pracy na fragmentach kryptogramów i fragmentach tekstów jawnych, które zdobywają mniej lub bardziej legalnymi metodami.

Podczas łamania szyfrów kryptoanalitycy stosują następujące techniki:

- Metoda prób i błędów.

Metoda prób i błędów polega na podstawianiu różnych kombinacji klucza i poszukiwaniu sensownego tekstu jawnego. Metoda łamania klucza w wyniku wypróbowania wszystkich możliwych jego kombinacji nazywa się atakiem brutalnym.

- Analiza statystyczna.

Analiza statystyczna sprowadza się do określenia prawdopodobieństwa rozkładu liter w kryptogramie i tekście jawnym. Ponieważ rozkład liter jest charakterystyczną cechą każdego języka, informacje takie można wykorzystać do łamania niektórych szyfrów, np. szyfrów podstawieniowych.

- Wyszukiwanie prawdopodobnych słów.

Każdy dokument lub program zawiera pewne słowa i zwroty pojawiające się w określonych miejscach. W dokumentach takimi słowami są np. nazwy miejscowości i zwroty grzecznościowe, a w programach słowa kluczowe. Za pomocą prawdopodobnych słów można znaleźć fragmenty klucza.

- Analiza matematyczna.

Metoda ta polega na napisaniu układu równań na podstawie znanych algorytmów, rozwiązanie których da wartości zmiennych, reprezentujących fragmenty wiadomości lub klucza. W ten sposób można również uzyskać wyrażenia generujące klucze kryptograficzne.

W zależności od informacji, jakie posiada kryptoanalityk, wybiera on jedną z metod postępowania. Metody łamania szyfrów można podzielić na trzy grupy.

1. Łamanie szyfru ze znanym kryptogramem (cipher-only attack).

Na podstawie kilku kryptogramów kryptoanalityk stara się znaleźć fragmenty tekstu jawnego i klucza, które pozwolą odszyfrować inne kryptogramy. W tym przypadku kryptoanalityk może wykorzystać metodę prób i błędów, analizę statystyczną lub poszukiwać prawdopodobnego słowa.

2. Łamanie szyfru ze znanym tekstem jawnym (known-plaintext attack).

Kryptoanalityk dysponuje fragmentami kryptogramów i odpowiadających im tekstów jawnych. Stara się on określić klucz i algorytm do deszyfrowania kolejnych wiadomości zaszyfrowanych tym samym kluczem.

3. Łamanie szyfru z wybranym tekstem jawnym (chosen-plaintext attack).

Podczas łamania szyfru z wybranym tekstem jawnym kryptoanalityk może zdobyć kryptogram odpowiadający wybranemu fragmentowi tekstu jawnego. Sytuacja taka jest dogodniejsza niż w poprzednim przypadku, gdyż kryptoanalityk może zdobyć więcej informacji o kluczu. Ilość informacji pozwala zwykle zastosować analizą matematyczną. Na taki atak są szczególnie narażone bazy danych wtedy, gdy użytkownik wprowadza do nich określone informacje. Kryptoanalityk może wówczas obserwować zmiany w ich tekście zaszyfrowanym.

Systemy kryptograficzne charakteryzują się różnymi poziomami bezpieczeństwa. Teoretycznie wszystkie szyfry można złamać, jeśli do dyspozycji będzie dostatecznie dużo czasu i mocy obliczeniowej. Ponieważ złamanie niektórych szyfrów wymaga zbyt dużych nakładów, praktycznie nie można ich złamać i są one uważane za bezpieczne.

Do określenia szyfrów bezpiecznych stosuje się dwa pojęcia: bezpieczeństwa bezwarunkowego i bezpieczeństwa obliczeniowego. Szyfr jest *bezwarunkowo bezpieczny*, jeśli niezależnie od liczby przechwyconych kryptogramów nie ma w nich wystarczających informacji, aby jednoznacznie określić tekst jawny. Szyfr jest *bezpieczny obliczeniowo*, gdy nie można go złamać analitycznie za pomocą dostępnych środków.

Ilość czasu i mocy obliczeniowej potrzebnej do złamania szyfru nazywamy nakładem pracy i określamy za pomocą liczby operacji komputerowych. Ponieważ moc obliczeniowa komputerów ciągle rośnie, więc utrzymanie odpowiedniego poziomu bezpieczeństwa szyfrów wymaga konstruowania coraz lepszych algorytmów kryptograficznych.

3.2. Systemy kryptograficzne

3.2.1. System kryptograficzny z kluczem tajnym

Systemy kryptograficzne z kluczem tajnym są znane od dawna i często nazywa się je systemami konwencjonalnymi lub klasycznymi. W skład systemu kryptograficznego wchodzi pięć elementów:

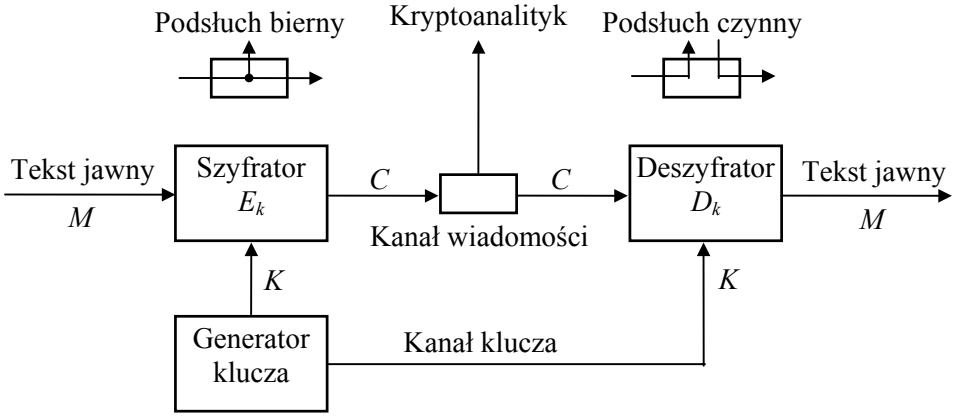
- przestrzeń wiadomości jawnych M,
- przestrzeń kryptogramów C,
- przestrzeń kluczy K,
- algorytm szyfrowania E,
- algorytm deszyfrowania D.

Schemat blokowy systemu kryptograficznego z kluczem tajnym, zastosowanego do ochrony kanału transmisji danych, pokazano na rys. 3.2.1. Elementami tego systemu kryptograficznego są: szyfrator, deszyfrator i generator klucza. Szyfrator realizuje algorytm kryptograficzny i generuje zbiór kryptogramów C z udziałem klucza kryptograficznego k

$$C = E_k(M). \tag{3.2.1}$$

Kryptogram jest transmitowany kanałem transmisyjnym, a następnie deszyfrowany przez deszyfrator. Na wyjściu deszyfratora otrzymujemy odtworzony zbiór tekstów jawnych M

$$M = D_k(C). \tag{3.2.2}$$



Rys. 3.2.1. System kryptograficzny z kluczem tajnym

Jeśli założymy, że kanał transmisyjny nie generuje błędów transmisyjnych, to odtworzony tekst jawny na wyjściu deszyfratora jest równy tekstowi jawnemu na wej-

ściu szyfratora. Operacje szyfrowania i deszyfrowania muszą być zdefiniowanymi jednoznacznie operacjami odwracalnymi

$$M = D_k\left(C_k\left(M\right)\right). \tag{3.2.3}$$

W czasie transmisji kryptogramów może nastąpić ich przechwycenie przez kryptoanalityka. Jeśli kryptoanalityk nie zmieni kryptogramu, to podsłuch taki nazywamy biernym, a jeśli nastąpi modyfikacja kryptogramu, podsłuch nazywamy czynnym. Te dwie metody podsłuchu pokazano schematycznie na rys. 3.2.1. Podsłuch bierny narusza poufność danych a podsłuch aktywny ich autentyczność.

W nowoczesnych systemach kryptograficznych algorytmy kryptograficzne nie są zazwyczaj tajne. Tajne algorytmy okazały się niepraktyczne w dużych systemach i dlatego stosuje się je tylko w systemach o małym stopniu zabezpieczenia. Przykładem stosowania tajnego algorytmu może być technika kodowania satelitarnych sygnałów telewizyjnych. W powszechnych systemach kryptograficznych bezpieczeństwo systemu zapewnia tajny klucz.

System kryptograficzny z kluczem tajnym ma wspólny klucz szyfrujący i deszyfrujący i dlatego nazywa się *systemem symetrycznym* lub systemem z jednym kluczem. Klucze kryptograficzne mogą być generowane za pomocą jednego generatora. Gdy klucze są przesyłane kanałami transmisyjnymi, jak to pokazano na rys. 3.2.1, wtedy kanał klucza musi być kanałem bezpiecznym.

Aby zachować poufność i autentyczność danych w systemach z tajnym kluczem, należy spełnić pewne wymagania. Poufność wymaga, aby kryptoanalityk nie mógł określić jawnej postaci danych na podstawie przechwyconego kryptogramu. Warunek poufności będzie spełniony, jeśli kryptoanalityk po przechwyceniu kryptogramu nie będzie mógł metodami obliczeniowymi określić przekształcenia deszyfrującego D_k ani tekstu jawnego M. Ażeby zachować poufności kryptogramu, należy więc chronić tajność przekształcenia deszyfrującego.

W celu zachowania autentyczności danych każda próba zastąpienia szyfru oryginalnego szyfrem fałszywym powinna być wykryta. Warunek autentyczności będzie spełniony, kiedy kryptoanalityk po przechwyceniu kryptogramu nie będzie mógł metodami obliczeniowymi określić przekształcenia szyfrującego E_k. Również nie powinno być możliwe doprowadzenie do ustalenia takiego kryptogramu C', że $D_k(C')$ będzie jednym z elementów zbioru M. W celu zachowania autentyczności kryptogramu należy więc chronić tajność przekształcenia szyfrującego.

Ponieważ w systemach symetrycznych klucze szyfrujące i deszyfrujące są takie same, a algorytmy kryptograficzne nie są tajne, zatem przekształcenia E_k i D_k łatwo można wyprowadzić z siebie. Dlatego też w takich systemach poufność i autentyczność nie mogą być traktowane rozdzielnie, a system musi spełniać jednocześnie wszystkie wymagania zarówno w stosunku do poufności, jak i autentyczności. Systemy symetryczne zabezpieczają tajność przekształceń E_k i D_k dzięki stosowaniu tajnego klucza, zapewniając w ten sposób poufność i autentyczność danych.

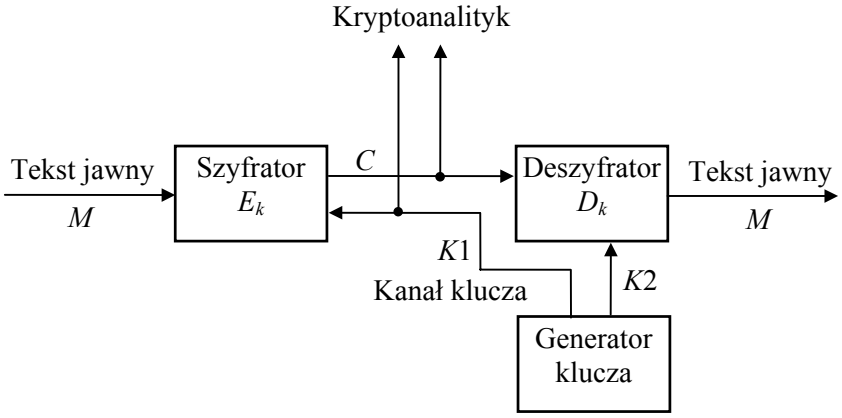
Szyfry symetryczne pod względem techniki szyfrowania dzielą się na:

- szyfry strumieniowe (stream ciphers),
- szyfry blokowe (block ciphers).

W szyfrowaniu strumieniowym lub potokowym przetwarzaną jednostką jest bit lub znak, a w szyfrowaniu blokowym blok, zawierający najczęściej osiem znaków. Obie te metody szyfrowania mogą być używane zarówno w transmisji danych, jak i do przechowywania danych w pamięciach.

3.2.2. System kryptograficzny z kluczem jawnym

Koncepcja systemu kryptograficznego z kluczem jawnym lub publicznym została przedstawiona pierwszy raz przez Diffie i Helmana w 1976 r. [3.3]. W systemie takim zastosowano dwa różne klucze: szyfrujący oraz deszyfrujący. Jest to system asymetryczny albo system z dwoma kluczami.



Rys. 3.2.2. System kryptograficzny z kluczem publicznym

Schemat blokowy systemu kryptograficznego z kluczem jawnym pokazano na rys. 3.2.2. W systemie tym generator klucza generuje dwa klucze: szyfrujący $K1$ i deszyfrujący $K2$. Klucz szyfrujący jest kluczem jawnym i przesyła się go do nadawcy informacji zwykłym kanałem bez zabezpieczeń. Każdy nadawca informacji może użyć klucza szyfrującego i obliczyć kryptogram

$$C = E_{k1}(M). \tag{3.2.4}$$

Algorytmy kryptograficzne z kluczem publicznym są tak skonstruowane, że odwrócenie operacji szyfrującej jest niemożliwe w rozsądnym czasie. Aby odczytać wiadomość zaszyfrowaną, należy znać klucz deszyfrujący $K2$. Klucz deszyfrujący jest tajny i nie może być wyznaczony z klucza szyfrującego. Proces deszyfrowania określa zależność

$$M = D_{k2}(C). \tag{3.2.5}$$

Oba powyższe przekształcenia muszą być operacjami jednoznacznie odwracalnymi

$$M = D_{k2}\left(C_{k1}\left(M\right)\right). \tag{3.2.6}$$

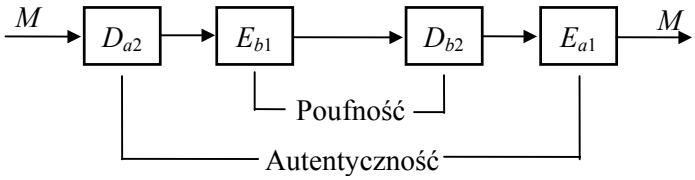
Systemy kryptograficzne z kluczem jawnym opierają się na tzw. nieodwracalnych funkcjach zapadkowych (trapdoor one-way functions). Obliczenie wartości takiej funkcji jest proste, ale odwrócenie operacji okazuje się praktycznie niemożliwe, gdy brak dodatkowych informacji. Algorytmy z kluczem jawnym opisano w rozdziale 3.5. Algorytmy te są algorytmami blokowymi.

W systemach asymetrycznych, podobnie jak w systemach symetrycznych, algorytmy kryptograficzne nie są tajne. Ponieważ w systemach z kluczem jawnym przekształcenia deszyfrującego D_{k2} nie można wyznaczyć z przekształcenia szyfrującego E_{k1}, poufność i autentyczność zapewniają oddzielne przekształcenia.

Jeśli system kryptograficzny ma konfigurację taką, jak przedstawiono na rys. 3.2.2, to poufność informacji jest zachowana, gdyż tylko odbiorca znający klucz $K2$ może odczytać szyfr. Autentyczność nie zostanie zachowana, ponieważ dowolny nadawca, znający przekształcenie szyfrujące, może podstawić fałszywy szyfr.

Aby zachować autentyczności informacji bez poufności, należy odwrócić kolejność operacji. Nadawca szyfruje wówczas wiadomość, używając tajnego przekształcenia D_{k2}, a odbiorca deszyfruje kryptogram, używając jawnego przekształcenia E_{k1}. Autentyczność w takim systemie jest zapewniona, gdyż tylko określony nadawca może stosować przekształcenia D_{k2}. Poufność natomiast nie jest zagwarantowana, ponieważ dowolny odbiorca znający przekształcenie E_{k1} może odtworzyć zaszyfrowaną informację.

W celu osiągnięcia poufności i autentyczności informacji jednocześnie nadawca i odbiorca muszą stosować podwójne przekształcenia pokazane na rys. 3.2.3. Nadawca i odbiorca wykorzystują dwie pary kluczy *a* i *b*. Nadawca, który chce przesłać wiadomość określonemu odbiorcy, najpierw szyfruje wiadomość *M*, stosując przekształcenie tajne D_{a2}, a następnie wynik poddaje przekształceniu jawnemu E_{b1}. Odbiorca w pierwszej kolejności stosuje przekształcenie tajne D_{b2}, a następnie przekształcenie jawne E_{a1}.



Rys. 3.2.3. Poufność i autentyczność w systemie z kluczem jawnym

Nie wszystkie algorytmy z kluczem publicznym mogą być używane w systemach zapewniających jednocześnie poufność i autentyczność. Algorytmy kryptograficzne z kluczem publicznym opisano w rozdziale 3.6. Opisany tam algorytm RSA może zapewnić poufność i autentyczność danych, a algorytm Merklego-Hellmana gwaran-

tuje tylko jedną z tych cech – albo poufność, albo autentyczność.

3.2.3. Ocena systemów kryptograficznych

Istnieje wiele kryteriów służących do oceny użytkowych właściwości systemu kryptograficznego. Pięć najważniejszych to [3.11]:

1. Stopień tajności.

Stopień tajności ocenia się za pomocą złożoności obliczeniowej. Podstaw do analizowania złożoności obliczeniowej szyfrów dostarcza teoria złożoności. Złożoność obliczeniowa algorytmu zależy od ilości pracy i ilości przechwyconego materiału potrzebnego do złamania szyfru. W ilości pracy uwzględnia się zasoby komputerowe, a ilość zgromadzonego materiału musi dać rozwiązanie jednoznaczne. System kryptograficzny jest bezpieczny wówczas, gdy wzrost ilości przechwyconego materiału nie ułatwia złamania szyfru.

2. Rozmiar klucza.

Klucze kryptograficzne są transmitowane kanałami transmisyjnymi lub przechowywane w pamięciach. Aby nie obciążać kanałów i pamięci, rozmiar klucza powinien być jak najmniejszy. Z drugiej strony dłuższy klucz zapewnia większy stopień tajności.

3. Złożoność algorytmów szyfrującego i deszyfrującego.

Złożoność algorytmów kryptograficznych powoduje przedłużenie czasu wykonywania operacji kryptograficznych i z tego względu powinny one być możliwie proste. Jednak bardziej złożone algorytmy kryptograficzne zapewniają wyższy stopień tajności.

4. Propagacja błędów.

W niektórych typach szyfrów nawet pojedyncze błędy transmisyjne powodują dużą liczbę błędów na wyjściu deszyfratora. Wywołuje to stratę informacji i konieczność jej retransmisji. Dlatego też dąży się od ograniczenia propagacji błędów.

5. Zwiększenie objętości danych.

W niektórych systemach kryptograficznych wzrasta objętość danych na wyjściu szyfratora w porównaniu z jego wejściem. Ten niepożądany efekt powoduje wzrost objętości przesyłanych danych oraz jest przyczyną wielu problemów na wyjściu systemu kryptograficznego i dlatego powinien być ograniczony.

Wartości parametrów użytkowych systemu kryptograficznego powinny wynikać z założonego stopnia bezpieczeństwa systemu. Stopień bezpieczeństwa systemu kryptograficznego decyduje o tym, jak trudno jest złamać szyfr.

3.3. Szyfry podstawieniowe i przestawieniowe

3.3.1. Podział szyfrów podstawieniowych

W szyfrze podstawieniowym (substitution cipher) każda litera lub grupa liter tekstu jawnego jest zastąpiona inną literą lub inną grupą liter. Szyfry podstawieniowe są najstarszą grupą szyfrów. Do konstrukcji takich szyfrów stosuje się permutacje, które są elementami kombinatoryki. Szyfry podstawieniowe należą do szyfrów z tajnym kluczem.

W kryptografii klasycznej rozróżnia się cztery grupy szyfrów podstawieniowych:

- monoalfabetowe,
- homofoniczne,
- wieloalfabetowe,
- poligramowe.

Szyfry monoalfabetowe zawierają takie same znaki w alfabecie jawnym i tajnym. Szyfry takie nazywamy również prostymi szyframi podstawieniowym lub endomorficznymi. W szyfrach homofonicznych każdemu znakowi tekstu jawnego przyporządkowuje się po kilka znaków kryptogramu.

Szyfry wieloalfabetowe stanowią kombinacje wielu prostych szyfrów podstawieniowych. Zmiana alfabetu może na przykład następować wraz ze zmianą pozycji znaku tekstu jawnego. W szyfrach poligramowych szyfruje się grupy znaków.

3.3.2. Proste szyfry podstawieniowe

W prostych szyfrach podstawieniowych lub monoalfabetowych stosuje się jednoznaczne odwzorowanie

$$f\colon\ m_i \to f(m_i),$$

które zastępuje każdy znak tekstu jawnego m_i odpowiadającym mu znakiem kryptogramu $f(m_i)$. W celu zaszyfrowania tekstu jawnego $m = m_1 m_2 \ldots$ przekształcenie f stosujemy do każdego znaku, skąd otrzymujemy

$$E_f(m) = f(m_1) f(m_2) \ldots$$

Prosty szyfr podstawieniowy można zdefiniować za pomocą tabeli 3.3.1.

Tabela 3.3.1. Szyfr podstawieniowy

m_i	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
$f(m_i)$	F	E	K	J	N	P	O	C	D	Y	U	H	V	M	B	Z	L	W	G	X	T	A	R	I	S	Q

W przypadku tak zdefiniowanego algorytmu podstawieniowego tekstowi jawnemu KRYPTOGRAFIA będzie odpowiadał szyfr UWSZXBOWFPDF.

W prostym szyfrze podstawieniowym odwzorowaniu n znaków odpowiada permutacja liczb całkowitych: $0,1,2,\ldots,n-1$. Liczba możliwych podstawień wynosi

$$n! = n \cdot (n-1) \ldots 2 \cdot 1. \tag{3.3.1}$$

Kluczem podstawieniowym jest permutacja elementów stosowanego alfabetu. Dla alfabetu angielskiego, zawierającego 26 liter, istnieje $26! = 4 \cdot 10^{26}$ różnych podstawień. Jednak szyfry monoalfabetowe są dość łatwe do złamania za pomocą analizy częstotliwości występowania znaków.

Odmianą szyfrów monoalfabetowych są *szyfry przesunięte*. W szyfrach tych alfabetem tajnym $f(m)$ jest alfabet przesunięty cyklicznie o pewną liczbę pozycji k. Przykładem takiego szyfru jest szyfr Cezara, w którym $k = 3$.

Jeśli korzystamy z alfabetu 26 literowego, to jego znaki możemy zakodować za pomocą liczb w sposób pokazany w tabeli 3.3.2.

Tabela 3.3.2. Przypisanie liczb literom alfabetu

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

Liczby odpowiadające znakom informacji m w alfabecie przesuniętym oblicza się z zależności

$$f(m) = (m + k) \bmod n, \tag{3.3.2}$$

gdzie n jest liczbą znaków.

Jeśli znakiem kryptogramu jest znak c, to przekształcenie deszyfrujące ma postać

$$f^{-1}(c) = (c - k) \bmod n. \tag{3.3.3}$$

Gdy stosujemy szyfr Cezara, dla którego $k = 3$, tekstowi jawnemu KRYPTOGRAFIA odpowiada kryptogram NUBSWRJUDILD.

Można skonstruować bardziej złożony algorytm szyfrowania, w którym zamiast dodawania stosuje się mnożenie. Znaki tekstu jawnego a mnoży się przez klucz k

$$f(m) = m\,k \bmod n. \tag{3.3.4}$$

Ze względu na warunek jednoznacznego deszyfrowania, klucz k wybiera się tak, aby k i n były względnie pierwsze. Dla liczb względnie pierwszych największy wspólny dzielnik jest równy jeden $NWD(n,k) = 1$. Do deszyfrowania stosuje się liczbę odwrotną k^{-1} liczoną modulo n, którą oblicza się z zależności

$$k\,k^{-1} (\bmod n) \equiv 1. \tag{3.3.5}$$

Funkcja deszyfrująca dla szyfru, w którym zastosowano iloczyn, będzie

$$f^{-1}(c) = c\,k^{-1} \bmod n. \tag{3.3.6}$$

P r z y k ł a d 3.3.1.

Szyfrowanie i deszyfrowanie za pomocą szyfru podstawieniowego z iloczynem.

Zaszyfrujmy znak tekstu jawnego S dla $k=7$ i $n=26$. Znak kryptogramu obliczamy z (3.3.4), podstawiając za m liczbę 18 odpowiadającą literze S z tabeli 3.3.2.

$$f(m) = mk \bmod n = 18 \cdot 7 \bmod 26 = 22.$$

Literze S tekstu jawnego odpowiada litera W kryptogramu.

Liczba $k^{-1}=15$, ponieważ $7 \cdot 15 (\text{mod } 26) = 1$. Znak tekstu jawnego odpowiadający znakowi kryptogramu W obliczamy z (3.3.6)

$$f^{-1}(c) = ck^{-1} \bmod n = 22 \cdot 15 \bmod 26 = 18.$$

W wyniku deszyfrowania otrzymaliśmy literę S.

Liczbę odwrotną k^{-1} modulo n obliczamy, rozwiązując kongruencję (3.3.5). Każda kongruencja ma nieskończenie wiele rozwiązań. Dla celów kryptograficznych potrzebny jest pierwiastek będący najmniejszą liczbą dodatnią. Rozwiązanie takie można znaleźć, rozwiązując następujące równanie w dziedzinie liczb całkowitych

$$k\,k^{-1} - t\,n = 1, \tag{3.3.7}$$

gdzie k i n są wiadome, a szukamy k^{-1}. Równanie to można rozwiązać, odwracając algorytm Euklidesa.

Algorytm Euklidesa stosuje się do wyznaczania największego wspólnego dzielnika NWD. Jeśli $d = NWD(a,b)$, to NWD można przedstawić w postaci kombinacji liniowej liczb a i b ze współczynnikami całkowitymi, to jest $d = u\,a + v\,b$. Opierając się na tym twierdzeniu, można rozwiązać równanie (3.3.7). Ilustruje to przykład 3.3.2.

P r z y k ł a d 3.3.2.

Wyznaczenie liczby odwrotnej modulo n.

Przyjmijmy dane z przykładu 3.3.1: $k=7$ i $n=26$. Szukamy k^{-1}, stosując algorytm Euklidesa. W tym celu należy rozwiązać równanie (3.3.7)

$$k^{-1} \cdot 7 - t \cdot 26 = 1.$$

Najpierw wyznaczamy $NWD(7,26)$:

$$26 = 3 \cdot 7 + 5,$$
$$7 = 1 \cdot 5 + 2,$$
$$5 = 2 \cdot 2 + 1.$$

Aby przedstawić liczbę 1 jako kombinację liniową liczb 7 i 26, wykorzystujemy ciąg równości od ostatniej do pierwszej w algorytmie Euklidesa. W każdym kroku zapisujemy liczbą 1 wykorzystując wcześniejsze reszty, aż dojdziemy do samych liczb 7 i 26:

$$1 = 5 - 2 \cdot 2 =$$
$$= 5 - 2(7 - 5) = 3 \cdot 5 - 2 \cdot 7 =$$
$$= 3(26 - 3 \cdot 7) - 2 \cdot 7 = 3 \cdot 26 - 11 \cdot 7.$$

W niektórych przypadkach występuje potrzeba zmiany znaków liczb w końcowym wyrażeniu. Aby zmienić znaki liczb, dodajemy i odejmujemy liczbę równą $7 \cdot 26$

$$1 = 3 \cdot 26 - 7 \cdot 26 + 26 \cdot 7 - 11 \cdot 7 = 15 \cdot 7 - 4 \cdot 26.$$

Porównując pierwsze równanie tego przykładu z otrzymanym wynikiem, widzimy, że k^{-1}=15.

Metodę przesunięcia i mnożenia w szyfrach prostych można łączyć, w wyniku czego otrzymujemy przekształcenie afiniczne

$$f(m) = (m k_1 + k_2) \bmod n, \tag{3.3.8}$$

gdzie k_1 i n są liczbami względnie pierwszymi. Kluczem jest tu para liczb k_1 i k_2.

Tabela 3.3.3. Częstość występowania wybranych znaków w tekstach

Znak	Polski	Ang.	Pascal	Znak	Polski	Ang.	Pascal	Znak	Polski	Ang.	Pascal
A	0,080	0,067	0,037	N	0,047	0,053	0,050	Spacja	0,172	0,197	0,192
B	0,013	0,013	0,013	O	0,071	0,063	0,046	0	–	–	0,003
C	0,038	0,019	0,032	P	0,024	0,012	0,022	1	–	–	0,004
D	0,030	0,031	0,028	Q	–	0,001	–	2	–	–	0,002
E	0,069	0,089	0,081	R	0,035	0,042	0,057	3	–	–	0,001
F	0,001	0,021	0,014	S	0,038	0,043	0,034	4	–	–	0,001
G	0,010	0,017	0,017	T	0,024	0,070	0,060	5	–	–	0,002
H	0,010	0,043	0,015	U	0,018	0,021	0,019	6	–	–	0,001
I	0,070	0,054	0,050	V	–	0,006	0,008	7	–	–	0,001
J	0,019	0,002	0,002	W	0,036	0,018	0,007	8	–	–	0,001
K	0,027	0,009	0,003	X	–	0,001	0,008	9	–	–	0,002
L	0,031	0,033	0,031	Y	0,032	0,023	0,008	.	0,009	0,008	0,012
M	0,024	0,022	0,014	Z	0,056	0,001	0,001	,	0,009	0,002	0,010

Szyfry monoalfabetyczne można łatwo złamać, stosując analizę statystyczną. W tym celu badamy rozkład częstości występowania znaków w tekście jawnym i kryptogramie. Następnie kojarzymy znaki o zbliżonej częstości. Względną częstość występowania liter, spacji, liczb oraz kropki i przecinka w języku literackim angielskim i polskim oraz programach w Pascalu pokazano w tabeli 3.3.3 [3.12]. W tabeli

nie uwzględniono znaków diakrytycznych języka polskiego. Kreska w tabeli oznacza, że częstość występowania znaku jest mniejsza od 0,0005.

W pracy kryptonalityka pomocna jest znajomość występowania ciągów dwuznakowych (digramów) i trzyznakowych (trigramów). Najłatwiejsze do złamania są szyfry oparte na alfabetach przesuniętych, ponieważ każda litera kryptogramu znajduje się w stałej odległości od odpowiedniej litery tekstu jawnego. W przypadku szyfrów, w których korzysta się z przekształceń afinicznych (3.3.8), współczynniki k_1 i k_2 można wyznaczyć, rozwiązując układ równań:

$$\begin{aligned} &(m_1 k_1 + k_2) \bmod n = c_1, \\ &\ldots \\ &(m_t k_1 + k_2) \bmod n = c_t, \end{aligned} \tag{3.3.9}$$

gdzie m_i są liczbami odpowiadającymi znakom tekstu jawnego, a c_i – kryptogramu.

P r z y k ł a d 3.3.3.

Łamanie szyfru z przekształceniem afinicznym.

Załóżmy, że w tekście jawnym i kryptogramie wykryto zgodne prawdopodobieństwa występowania następujących par liter: F i R oraz J i T. Podstawiając do (3.3.9) liczby z tabeli 3.3.2 odpowiadające literom, otrzymamy:

$$\begin{aligned} &(5k_1 + k_2) \bmod 26 = 17, \\ &(9k_1 + k_2) \bmod 26 = 19. \end{aligned}$$

Po odjęciu równania pierwszego od drugiego można obliczyć k_1

$$4k_1 \bmod 26 = 2, \quad k_1 = 7.$$

Liczbę k_2 obliczamy z pierwszego równania

$$(35 + k_2) \bmod 26 = 17, \quad k_2 = 8.$$

W praktyce, aby znaleźć k_1 i k_2, musimy korzystać z więcej niż dwóch równań, ponieważ powyższe równania mają rozwiązania wielokrotne. Rozwiązania wielokrotne występują w przypadku, gdy m_i jest podzielnikiem n oraz jeśli prawdopodobieństwa występowania kilku liter w tekście są w przybliżeniu równe.

3.3.3. Szyfry podstawieniowe homofoniczne

Szyfr homofoniczny odwzorowuje każdy znak tekstu jawnego m_i na jeden znak ze zbioru $f(m_i)$ tekstu zaszyfrowanego, gdzie zbiory $f(m_i)$ są rozłączne. Znaki zbioru $f(m_i)$ nazywa się homofonami. Tekst jawny $m = m_1 m_2 \ldots$ jest zaszyfrowany jako $c = c_1 c_2 \ldots$, przy czym c_i wybiera się dowolnie ze zbioru homofonów $f(m_i)$.

P r z y k ł a d 3.3.4.

Szyfr homofoniczny.

Przyjmijmy, że litery alfabetu angielskiego są szyfrowane jako liczby dwucyfrowe. Liczba znaków przydzielonych każdej literze jest proporcjonalna do względnej częstości jej występowania w tekście i każda z liczb jest przydzielona tylko jednej literze. Niżej podano fragment tabeli z możliwym przyporządkowaniem liczb.

Litera	Homofony
A	19 34 41 56 60 73 83 96
B	31
C	27 59 62 81
D	11 28 77
E	10 23 42 49 61 88 99
F	76
G	23
...	...

Na przykład tekst jawny m = A D A może być zaszyfrowany jako c = 73 11 34.

Szyfry homofoniczne mogą być znacznie trudniejsze do złamania niż zwykłe szyfry podstawieniowe. Szyfry homofoniczne ukrywają rozkład znaków dzięki przypisaniu literom tekstu jawnego wielu symboli kryptogramu. Im więcej symboli zostanie przydzielonych literom, tym silniejszy będzie szyfr. Szyfr ten nie zamazuje jednak statystycznych właściwości języka, co jest widoczne np. w analizie rozkładu digramów.

3.3.4. Szyfry podstawieniowe wieloalfabetowe

Szyfry wieloalfabetowe lub polialfabetyczne zostały wprowadzone w XVI wieku przez Battistę. Wieloalfabetowe szyfry podstawieniowe mają wiele jednoznakowych kluczy, które zmieniają się w procesie szyfrowania. Każdy z kluczy szyfruje jeden znak tekstu jawnego. Po wyczerpaniu się wszystkich kluczy wraca się do klucza pierwszego. Liczba kluczy jest nazywana *okresem szyfru*. Najczęściej wieloalfabetowe szyfry podstawieniowe są szyframi okresowymi. Szyfr taki ukrywa rozkład znaków tekstu jawnego dzięki użyciu wielu podstawień. Do popularnych szyfrów wieloalfabetowych należą szyfry Vigenère'a i Beauforta.

Szyfry Vigenère'a

Szyfr Vigenère'a pochodzi z XVI wieku. Wymaga on przypisania znakom tekstu jawnego liczb, np. według tabeli 3.3.2. Jeśli kluczem szyfru jest sekwencja znaków $k = k_1 k_2 \ldots k_d$, to szyfrowanie znaku m_i, należącego do jawnego alfabetu n-literowego, określa zależność

$$c_i = (m_i + k_i) \bmod n. \tag{3.3.10}$$

Wyrażenie deszyfrujące ma postać

$$m_i = (c_i - k_i) \bmod n. \tag{3.3.11}$$

Jeśli okres szyfru Vigenère'a d=1, szyfr staje się prostym szyfrem podstawieniowym.

Tabela 3.3.4. Tablica Vigenère'a

		Tekst jawny																								
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
	C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
	D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
	E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
	F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
	G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
	H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
	I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
	J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
l	L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
u	M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
c	N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
z	O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
	R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
	S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
	T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
	V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
	Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
	Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Szyfr Beauforta jest szczególnym przypadkiem szyfru Vigenère'a, w którym

$$c_i = (m_i - k_i) \bmod n. \tag{3.3.12}$$

Ponieważ $n \bmod n = 0$, możemy napisać

$$c_i = (m_i - k_i) \bmod n = (m_i + (n - k_i) \bmod n.$$

Szyfr Beauforta odpowiada szyfrowi Vigenère'a z kluczem $n - k_i$. Z porównania wzorów (3.3.11) i (3.3.12) widać, że szyfr Beauforta jest odwrotnością szyfru Vigenère'a.

Szyfry Vigenère'a można stosować w postaci złożonej z wieloma kluczami $k_i, l_i, \ldots, s_i$

$$c_i = (m_i + k_i + l_i + \ldots + s_i) \bmod n.$$

Klucze $k_i, l_i, \ldots, s_i$ powinny mieć różne okresy. Okres sumy $k_i + l_i + \ldots + s_i$ jest równy najmniejszej wspólnej wielokrotności okresów poszczególnych kluczy.

Dużym ułatwieniem podczas szyfrowaniu i deszyfrowaniu tekstu jest tablica Vigenère'a pokazana w tabeli 3.3.4.

P r z y k ł a d 3.3.5.

Szyfrowanie tekstu szyfrem Vigenère'a.

Załóżmy, że alfabet zawiera 26 liter, a kluczem szyfru Vigenère'a jest ciąg E N T D O o okresie $d = 5$. Do szyfrowania stosujemy tabelę 3.3.4. Tekst jawny KRYPTOGRAFIA będzie zaszyfrowany następująco

m = K R Y P T O G R A F I A
k = E N T D O E N T D O E N
c = O E R S H S T K D T M N

W przykładzie pierwszą literę każdej grupy przesunięto o 4 pozycje, drugą – o 13, trzecią – o 19, czwartą – o 3, a piątą – o 14 pozycji.

Tablica Vigenère'a może służyć zarówno do szyfrowania, jak i do deszyfrowania. W procesie deszyfrowania szukamy wiersza odpowiadającego znakowi klucza, a następnie literę tekstu jawnego odczytujemy w nagłówku tej kolumny, w której wystąpi litera kryptogramu.

Aby złamać okresowy szyfr podstawieniowy, należy najpierw wyznaczyć okres tego szyfru, a następnie wyznaczyć znaki klucza. Okres wyznacza się, stosując wskaźnik zgodności lub metodę Kasiskiego. Algorytmy te opisano w [3.2, 3.12].

Szyfrowanie ciągów binarnych

Bezpieczeństwo szyfrów podstawieniowych wzrasta wraz ze wzrostem długości klucza. Jeśli długość klucza jest równa długości informacji, a klucz jest losową sekwencją znaków, to takiego szyfru nie można złamać. Szyfr ten nazywamy szyfrem z *kluczem jednokrotnym* (one time pad). Gdy klucz ma redundancję, wówczas może ona być wykorzystana do złamania szyfru. Szyfr z kluczem jednokrotnym został wynaleziony w 1917 r. przez Vernama do łączności telegraficznej. Może on być stoso-

wany dla ciągów binarnych.

W przypadku ciągów binarnych można zdefiniować dwa klucze: klucz 0 i klucz 1 pokazane w tabeli 3.3.5. Szyfrowanie dla tak zdefiniowanych kluczy realizuje się, dodając modulo 2 kolejne bity tekstu jawnego i klucza

$$c_i = (m_i + k_i) \bmod 2 \quad \text{dla} \quad i = 1,2,\ldots \tag{3.3.13}$$

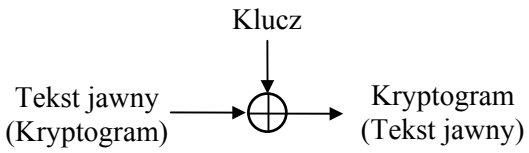
Ponieważ $(k_i + k_i) \bmod 2 = 0,$ deszyfrowanie można wykonać za pomocą takiej samej operacji

$$m_i = (c_i + k_i) \bmod 2 \quad \text{dla} \quad i = 1,2,\ldots \tag{3.3.14}$$

Tabela 3.3.5. Klucze dla alfabetu binarnego

Klucz 0		Klucz 1	
Tekst jawny	Szyfr	Tekst jawny	Szyfr
0	0	0	1
1	1	1	0

W praktyce szyfrator szeregowy dla ciągów binarnych można zrealizować za pomocą bramki Ex-OR. Schemat takiego układu pokazano na rys 3.3.1. Układ ten może również służyć do deszfrowania kryptogramów.



Rys. 3.3.1. Szyfrator dla ciągów binarnych

Operacja szyfrowania i deszyfrowania ciągu binarnego przebiega następująco:

Szyfrowanie								
Tekst jawny	1	0	0	1	1	1	0	1
Klucz	1	1	0	1	0	0	1	0
Kryptogram	0	1	0	0	1	1	1	1

Deszyfrowanie								
Kryptogram	0	1	0	0	1	1	1	1
Klucz	1	1	0	1	0	0	1	0
Tekst jawny	1	0	0	1	1	1	0	1

Jak wspomniano wyżej, szyfr taki jest mocny, jeśli klucz jest jednokrotny. Wtedy szyfr można złamać w wyniku przechwycenia szyfrogramu oraz odpowiadającego mu tekstu jawnego i odtworzenia na tej podstawie klucza kryptograficznego.

3.3.5. Szyfry podstawieniowe poligramowe

Szyfry poligramowe lub wieloliterowe szyfrują jednocześnie większe bloki liter.

Złamanie takiego szyfru jest znacznie trudniejsze ze względu na ukrycie częstości występowania liter. Typowymi szyframi poligramowymi są szyfr Playfaira i szyfr Hilla.

Szyfr Playfaira został wynaleziony w połowie XIX wieku i był stosowany w okresie I wojny światowej. Kluczem w tym szyfrze jest przypadkowa tablica 5×5 znakowa, w której pominięto nieużywaną literę J.

W	A	R	F	S
I	C	O	D	B
E	P	G	K	Y
M	N	Q	T	U
V	H	X	L	Z

Szyfrowanie pary liter tekstu jawnego $m_1 m_2$ przeprowadza się następująco:

1. Jeśli m_1 i m_2 znajdują się w tym samym wierszu, to znakami kryptogramu c_1 i c_2 są litery leżące po prawej stronie m_1 i m_2, przy czym pierwszą kolumnę traktuje się jako położoną na prawo od ostatniej.

2. Jeśli m_1 i m_2 znajdują się w tej samej kolumnie, to znakami kryptogramu c_1 i c_2 są litery leżące poniżej m_1 i m_2, przy czym pierwszy wiersz traktuje się jako wiersz leżący pod ostatnim wierszem.

3. Jeśli m_1 i m_2 znajdują się w różnych wierszach i kolumnach, to znakami kryptogramu c_1 i c_2 są litery leżące w narożnikach prostokąta wyznaczonego przez m_1 i m_2, przy czym c_1 pochodzi z wiersza zawierającego m_1, a c_2 – z wiersza zawierającego m_2.

4. Jeśli $m_1 = m_2$, to do tekstu jawnego między te litery wstawia się nieznaczącą literę, np. X, co eliminuje powtórzenia. Podobnie dopisuje się nieznaczącą literę na końcu tekstu, jeśli ostatnia litera nie ma pary.

Stosując powyższe zasady, po zaszyfrowaniu tekstu jawnego KRYPTOGRAFIA otrzymamy kryptogram GFEGQDQORSWC.

Szyfr Hilla dokonuje przekształcenia liniowego d znaków tekstu jawnego $m_1 m_2 \ldots m_d$ w d znaków kryptogramu $c_1 c_2 \ldots c_d$. Kluczem jest macierz współczynników. Dla $d = 2$, wyrażenie szyfrujące ma postać:

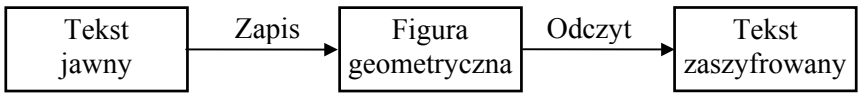
$$\boldsymbol{C} = \boldsymbol{K}\,\boldsymbol{M} \bmod n,$$
$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} \bmod n. \tag{3.3.15}$$

Deszyfrowanie wykonuje się, używając macierzy odwrotnej $\boldsymbol{K}^{-1}$, przy czym $\boldsymbol{K}^{-1}\boldsymbol{K} \bmod n = \boldsymbol{I}$, gdzie $\boldsymbol{I}$ jest macierzą jednostkową.

3.3.6. Szyfry przestawieniowe

Szyfr przestawieniowy lub permutacyjny (transposition cipher) zmienia kolejność znaków w tekście. Przekształcenia to można wykonać za pomocą figur geometrycznych w sposób pokazany na rys. 3.3.2.

Figurą geometryczną tradycyjnie jest macierz dwuwymiarowa. Wtedy tekst jawny zapisujemy np. w wierszach o określonej liczbie pozycji, a następnie odczytujemy go w kolumnach. Deszyfrowanie odbywa się dzięki zapisaniu kryptogramu w kolumnach i odczytaniu go w wierszach. Klucz tego szyfru składa się z macierzy oraz ścieżek zapisu i odczytu.



Rys. 3.3.2. Realizacja przestawienia

P r z y k ł a d 3.3.6.

Szyfrowanie informacji za pomocą szyfru przestawieniowego.

Niech tekstem jawnym będzie słowo KRYPTOGRAFIA. Tekst ten wpisujemy do macierzy o rozmiarach 4×3.

1	2	3
K	R	Y
P	T	O
G	R	A
F	I	A

Odczytując kolumny, np. w kolejności: 2, 1, 3, otrzymamy kryptogram RTRI-KPGFYOAA.

W szyfrach przestawieniowych można stosować różne figury geometryczne oraz przestawiać kolumny. Szyfry te nie zmieniają częstości występowania liter. Szyfry przestawieniowe mogą być łamane metodą anagramową opartą na badaniu częstości występowania digramów i trigramów w tekście zaszyfrowanym.

Szyfry przestawieniowe można opisać za pomocą permutacji. Wiele szyfrów przestawieniowych permutuje znaki tekstu jawnego z pewnym okresem p. Szyfry takie nazywamy *permutacyjnymi*. Szyfry przestawieniowe są najczęściej wykorzystywane jako elementy składowe szyfrów złożonych.

3.4. Szyfry kaskadowe

3.4.1. Charakterystyka szyfrów kaskadowych

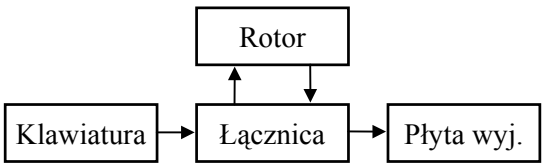
Połączenie podstawowych metod szyfrowania, jakimi są pojedyncze podstawienie lub przestawienie, daje szyfr złożony nazywany szyfrem kaskadowym lub produktowym (product cipher). Szyfry kaskadowe są połączeniem szyfrów podstawieniowych lub podstawieniowych i przestawieniowych. Szyfry takie mają lepsze właściwości kryptograficzne niż samo podstawienie lub przestawienie. Szyfry kaskadowe są realizowane w szyfrujących maszynach rotorowych i algorytmach komputerowych. Omówimy je na przykładzie maszyny rotorowej Enigmy i algorytmów komputerowych Lucifer i DES. Opis innych algorytmów można znaleźć w [3.2, 3.9].

Charakterystykę teoretyczną szyfrów kaskadowych podał Shannon [3.11]. Szyfry kaskadowe umożliwiają zmniejszenie nadmiarowości tekstu jawnego i zwiększenie bezpieczeństwa szyfru. W tym celu stosuje się dwie podstawowe techniki: mieszania (confusion) i rozproszenia (diffusion). Mieszanie zmniejsza związek między tekstem jawnym a kryptogramem i jest realizowane w wyniku podstawienia. Podstawienia, nawet złożone, nie są dostatecznie skutecznym zabezpieczeniem. Przykładem jest algorytm Enigmy, który został złamany bez zastosowania komputerów.

Rozproszenie rozprzestrzenia nadmiarowość tekstu jawnego po całym kryptogramie. Rozproszenie realizuje się za pomocą przestawienia. W szyfrach strumieniowych używa się wyłącznie techniki mieszania. W szyfrach blokowych są stosowane obie techniki: mieszania i rozproszenia. Użycie wyłącznie rozproszenia jest mało skuteczne, gdyż szyfr taki można łatwo złamać.

3.4.2. Maszyny rotorowe

W latach dwudziestych i trzydziestych XX wieku wynaleziono wiele urządzeń mechanicznych i elektromechanicznych do szyfrowania tekstów. Głównym elementem tych maszyn był wirnik, zwany rotorem, do wykonywania podstawień. Maszyna rotorowa miała kilka rotorów i realizowała szyfr Vigenère'a z bardzo długim okresem. Klucz szyfrujący był określony sposobem okablowania rotorów i ich położeniem. Ze zmianą położenia rotorów następowała zmiana klucza.



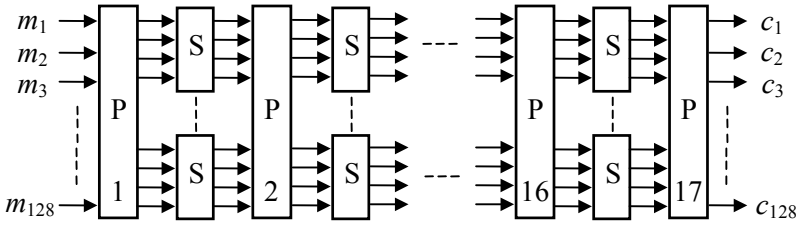
Rys. 3.4.1. Schemat blokowy Enigmy

Maszyny rotorowe stosowano w okresie II wojny światowej. Najbardziej znaną maszyną elektromechaniczną z tego okresu jest niemiecka Enigma. Schemat blokowy Enigmy przedstawia rys. 3.4.1. Litery tekstu jawnego wprowadzano z klawiatury, a znaki kryptogramu podświetlano na płycie wyjściowej. Struktura połączeń maszyny umożliwiała szyfrowanie i deszyfrowanie dokumentów. Elementami szyfrującymi w maszynie były obracający się rotor i łącznica. W Enigmie rotor miał od czterech do ośmiu wirników. Łącznica umożliwiała zamianę 12 liter spośród 26 liter alfabetu, co zwiększało ogólną liczbę kluczy. Każdy z wirników miał 26 pozycji, a okres dla maszyny z n rotorami jest równy 26^n. Na przykład maszyna z czterema wirnikami ma okres $26^4 = 456976$. Całkowita liczba możliwych do uzyskania kluczy wynosiła około $2 \cdot 10^{20}$.

Pomimo złożoności Enigmy jej metody szyfrowania zostały złamane przez polskich kryptografów w końcu 1932 r. W czasie II wojny światowej wiedza ta została przekazana Brytyjczykom i Francuzom. Metody szyfrowania stosowane w Enigmie można zrealizować programowo. Rozwiązanie takie zaimplementowano w systemie operacyjnym Unix do szyfrowania plików.

3.4.3. Algorytm Lucifer

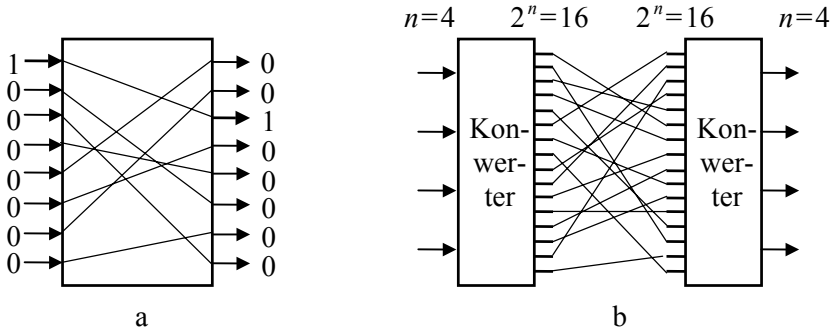
Algorytm Lucifer zaprojektowano na początku lat siedemdziesiątych w IBM. Szyfr składa się z wykonywanych na przemian podstawień i permutacji. Urządzenia realizujące te operacje nazwano skrzynkami podstawień S i skrzynkami permutacji P.



Rys. 3.4.2. Schemat blokowy algorytmu Lucifer

Schemat blokowy algorytmu Lucifer pokazano na rys 3.4.2. Skrzynka permutacji ma 128 wejść i tyle samo wyjść. Zmienia ona kolejność bitów. Na rys. 3.4.3a pokazano uproszczoną skrzynkę permutacji o ośmiu wejściach.

Skrzynka podstawień zawiera dwa układy zamieniające liczbę n-bitową na liczbę 2^n-bitową i odwrotnie. Ilustruje to rys. 3.4.3b. Między tymi układami zmienia się połączenia. Układ taki wykonuje przekształcenia nieliniowe, w wyniku czego liczby jedynek i zer są różne na wejściu i wyjściu skrzynki podstawień. W algorytmie ustalono dwa sposoby połączeń, oznaczone przez 0 i 1, które mogą być wybierane kluczem zewnętrznym. Całkowita liczba skrzynek S w układzie wynosi 512. Skrzynkami



Rys. 3.4.3. Skrzynka permutacji P (a) i skrzynka podstawień S (b)

tymi steruje się za pomocą klucza 128 bitowego. Do zamiany ciągu klucza 128-bitowego na ciąg sterujący 512-bitowy służy specjalny algorytm. Metody stosowane w szyfrze Lucifer umożliwiły stworzenie algorytmu DES.

3.4.4. Standard szyfrowania danych DES

W 1977 r. Narodowe Biuro Normalizacji w USA (National Bureau of Standards) wprowadziło oficjalnie normę kryptograficzną Data Encryption Standard [3.1] do szyfrowania informacji nieutajnianych przez agendy rządowe. DES szyfruje 64-bitowe bloki danych przy użyciu klucza o długości 56 bitów. Do szyfrowania i deszyfrowania używa się takiego samego algorytmu.

Opis algorytmu

Schemat blokowy algorytmu DES pokazano na rys. 3.4.4. 64-bitowy blok wejściowy jest poddany permutacji początkowej *IP* zgodnie z tabelą 3.4.1.

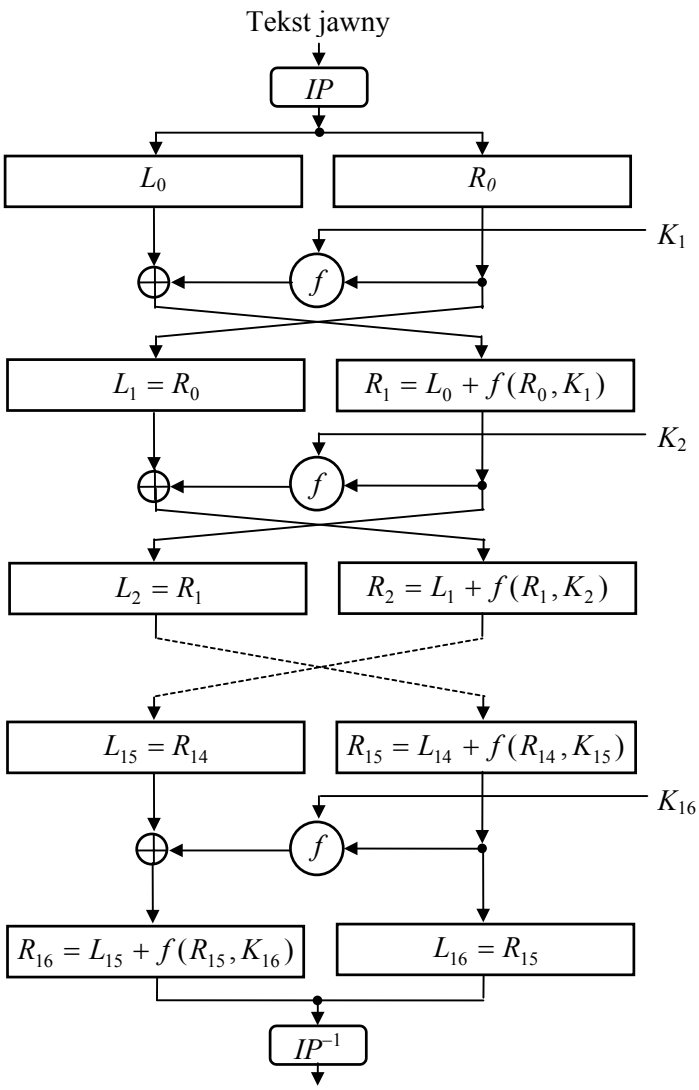
Tabela 3.4.1. Permutacja początkowa *IP*

58	50	42	34	26	18	10	02	60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06	64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01	59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05	63	55	47	39	31	23	15	07

Permutacja *IP* z ciągu bitów $t_1, t_2, t_3, \ldots, t_{64}$ tworzy ciąg $t_{58}, t_{50}, t_{42}, \ldots, t_7$. Podobnie będą zapisane pozostałe permutacje algorytmu DES. Po wykonaniu permutacji początkowej blok wejściowy jest dzielony na dwa bloki: lewy L_0 i prawy R_0. Dalsze operacje są wykonywane oddzielnie na każdym z tych bloków. Blok R_0 jest przesuwany na lewo, a blok L_0 przekształcany z zastosowaniem funkcji szyfrującej $f(x)$ i umieszczany z prawej strony.

Funkcja szyfrująca składa się z podstawień i permutacji, a jej parametrami są blok R_0 i klucz K_1. Operacje te powtarza się szesnaście razy. Każdy z tych cykli można opisać zależnościami:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} + f\left(R_{i-1}, K_i\right).$$



Rys. 3.4.4. Schemat blokowy algorytmu DES

Po ostatniej iteracji nie zmienia się pozycji części lewej i prawej, lecz łączy się je razem i wykonuje permutację końcową IP^{-1}.

Tabela 3.4.2. Permutacja końcowa IP^{-1}

40	08	48	16	56	24	64	32	39	07	47	15	55	23	63	31
38	06	46	14	54	22	62	30	37	05	45	13	53	21	61	29
36	04	44	12	52	20	60	28	35	03	43	11	51	19	59	27
34	02	42	10	50	18	58	26	33	01	41	09	49	17	57	25

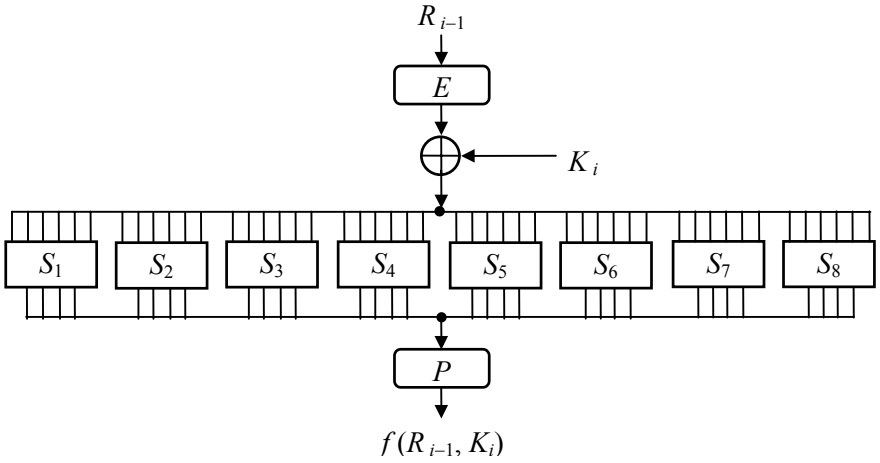
Obliczanie funkcji szyfrującej *f*

Funkcja szyfrująca $f(R_{i-1}, K_i)$ zawiera szereg operacji pokazanych na rys. 3.4.5. Danymi wejściowymi funkcji f są: 32-bitowy blok R_{i-1} i 48-bitowy klucz K_i. Na bloku 32-bitowym R_{i-1} wykonuje się permutację rozszerzającą *E*.

Tabela 3.4.3. Permutacja rozszerzająca *E*

32	01	02	03	04	05	04	05	06	07	08	09	08	09	10	11
12	13	12	13	14	15	16	17	16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27	28	29	28	29	30	31	32	01

W wyniku tej permutacji tworzy się ciąg 48-bitowy. Następnie ciąg ten jest dodawany modulo 2 do klucza K_i za pomocą bramki Ex-OR, a blok wyjściowy dzieli się na osiem ciągów 6-bitowych. Każdy z tych ciągów zostaje poddany operacji podstawienia w następujący sposób. Pierwszy i ostatni bit ciągu 6-bitowego określa numer wiersza tablicy podstawienia S_i od 0 do 3, a cztery bity środkowe – numer jej kolumny od 0 do 15.



Rys. 3.4.5. Schemat blokowy obliczania funkcji szyfrującej

Tablice podstawień dla kolejnych iteracji $S_1 \div S_8$ pokazano w tabeli 3.4.4. Na przykład, jeśli na wejściu elementu S_1 będzie ciąg binarny 110010, to liczba dziesięt-

na odpowiadająca ciągowi wyjściowemu będzie w 2 wierszu i 9 kolumnie. Liczba ta wynosi 12, co odpowiada ciągowi binarnemu 1100.

Tabela 3.4.4. Tablice podstawień funkcji f

S_i	Wiersz	Kolumna															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1	0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
	1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
	2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
	3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13
S_2	0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
	1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
	2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
	3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09
S_3	0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
	1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
	2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
	3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12
S_4	0	07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
	1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
	2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
	3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14
S_5	0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
	1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
	2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
	3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03
S_6	0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
	1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
	2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
	3	04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13
S_7	0	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
	1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
	2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
	3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12
S_8	0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
	1	01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
	2	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
	3	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

Opisany wyżej proces podstawień jest procesem nieliniowym i stanowi krytyczny etap w algorytmie. Zapewnia on w znacznym stopniu bezpieczeństwo szyfru, gdyż jest trudny do przeanalizowania.

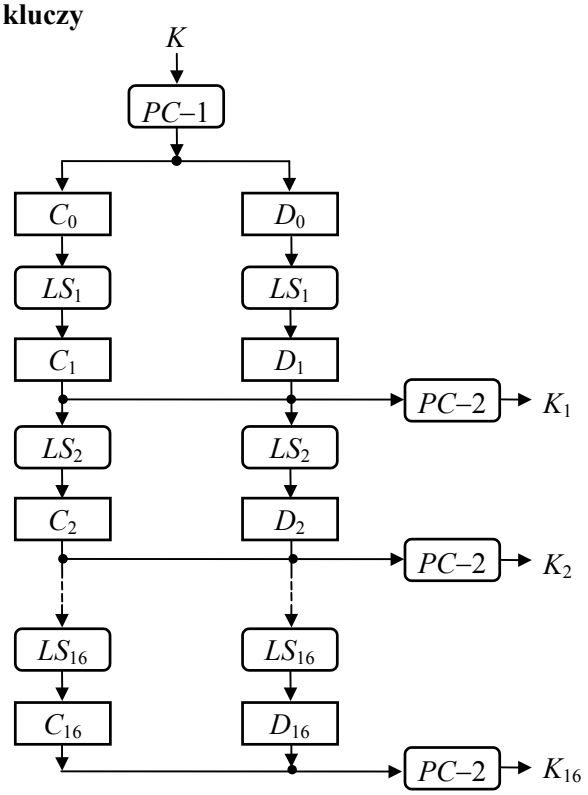
Tablice podstawień na rys 3.4.5 są reprezentowane przez bloki S_i. Ciągi wyjściowe tych bloków łączy się razem w wyniku konkatenacji w jeden ciąg 32-bitowy. Na ciągu tym wykonuje się permutację P według tabeli 3.4.5.

Tabela 3.4.5. Permutacja P

16	07	20	21	29	12	28	17	01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09	19	13	30	06	22	11	04	25

Uzyskany w ten sposób ciąg binarny jest dodawany modulo 2 do L_{i-1} za pomocą bramki Ex-OR, w rezultacie czego powstaje blok R_i, jak to pokazano na rys. 3.4.4.

Generowanie kluczy



Rys. 3.4.6. Schemat blokowy generatora kluczy

Kolejnym zagadnieniem jest sposób tworzenia kluczy K_i. Klucz pierwotny 64-bitowy K wprowadzony do systemu kryptograficznego służy do generowania 16 kluczy wtórnych zawierających po 48 bitów i używanych w procesie szyfrowania i deszyfrowania.

Schemat blokowy algorytmu generowania kluczy pokazano na rys. 3.4.6. Klucz pierwotny K po odrzuceniu ośmiu bitów parzystości jest poddany permutacji PC-1,

pokazanej w tabeli 3.4.6. Otrzymany 56-bitowy blok klucza dzieli się na dwa bloki 28-bitowe C_0 i D_0. Bloki te są oddzielnie przesuwane cyklicznie w lewo, w wyniku czego powstają ciągi C_1 i D_1. Liczbę przesunięć w lewo bloków klucza dla kolejnych cykli podano w tabeli 3.4.8. Na rysunku 3.4.6 przesunięcia są realizowane w blokach LS_i. Po przesunięciu bloków kluczy w lewo łączy się je razem za pomocą operatora konkatenacji i poddaje permutacji PC-2 pokazanej w tabeli 3.4.7.

Tabela 3.4.6. Permutacja klucza PC-1

57	49	41	33	25	17	09	01	58	50	42	34	26	18
10	02	59	51	43	35	27	19	11	03	60	52	44	36
63	55	47	39	31	23	15	07	62	54	46	38	30	22
14	06	61	53	45	37	29	21	13	05	28	20	12	04

Tabela 3.4.7. Permutacja PC-2

14	17	11	24	01	05	03	28	15	06	21	10
23	19	12	04	26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Tabela 3.4.8. Tablica przesunięć kluczy w lewo

Cykl i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	5	16
Przesunięcia	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Permutacja PC-2 jest permutacją z kompresją. Wybiera ona 48 bitów z ciągu 56-bitowego. Otrzymany w ten sposób 48-bitowy klucz jest wykorzystywany w procesie szyfrowania lub deszyfrowania. Do deszyfrowania używa się kluczy w odwrotnej kolejności.

Proces deszyfrowania przebiega według tego samego algorytmu, ale w pierwszej iteracji używa się klucza K_{16}, w drugiej K_{15} itd. Wynika to z faktu, że permutacja końcowa IP^{-1} jest odwróceniem permutacji początkowej IP, a proces szyfrowania przebiega w kolejności odwrotnej:

$$R_{i-1} = L_i,$$

$$L_{i-1} = R_i + f\left(L_i, K_i\right).$$

W algorytmie DES stosuje się klucz 56-bitowy. Liczba kluczy możliwych do wygenerowania wynosi 2^{56}. Jednak połowę tych kluczy stanowią klucze komplementarne, które zawierają zera w miejscu jedynek i odwrotnie. Ponieważ w wyniku szyfrowania komplementarnego tekstu jawnego za pomocą klucza komplementarnego otrzyma się również szyfrogram komplementarny, więc podczas brutalnego ataku na szyfr wystarczy przetestować tylko $2^{55} = 3{,}6 \cdot 10^{16}$ kluczy.

Przyjęty w algorytmie DES sposób generowania kluczy wtórnych dla każdego cyklu algorytmu powoduje, że niektóre klucze są słabe. Klucze takie generują szesnaście jednakowych kluczy wtórnych. Kluczy słabych jest cztery i pokazano je w zapisie szesnastkowym w tabeli 3.4.9. W drugiej kolumnie tabeli znajdują się ciągi kluczy po odrzuceniu bitów parzystości. Istnieją również klucze półsłabe i częściowo słabe. Wszystkie takie klucze są mało bezpieczne, a wykaz ich podano w [3.9].

Tabela 3.4.9. Klucze słabe algorytmu DES

Pierwotny klucz słaby	Ciąg klucza słabego
01010101 01010101	0000000 0000000
FEFEFEFE FEFEFEFE	FFFFFFF FFFFFFF
1F1F1F1F 0E0E0E0E	0000000 FFFFFFF
E0E0E0E0 F1F1F1F1	FFFFFFF 0000000

Najprostszym sposobem zwiększenia rozmiaru klucza oraz mocy szyfru jest dwukrotne szyfrowanie z dwoma niezależnymi kluczami. W [3.9] podano kilka innych sposobów modyfikacji algorytmu DES.

Szyfr DES zaimplementowano zarówno w wersji programowej, jak i sprzętowej. W realizacji sprzętowej uzyskuje się większe szybkości przetwarzania. W implementacji programowej szybkość szyfrowania na procesorze 80386 dla częstotliwości zegara 25 MHz wynosi 320 Kbit/sek, a na procesorze 80486 dla częstotliwości zegara 33 MHz – około 2,6 Mbit/sek.

Jako przykład realizacji algorytmu DES w postaci układów scalonych mogą posłużyć układy WD2001 i WD2002 produkowane przez firmę Western Digital w technologii NMOS. Dla częstotliwości zegara 3 MHz uzyskuje się szybkość przetwarzania 1,8 Mbit/sek. W szybkości tej uwzględniono czas: wprowadzania danych i klucza oraz wyprowadzania danych. Łącznie czynności te zajmują około 60% czasu przetwarzania. W 1993 r. doniesiono, że firma Digital Equipment Corporation opracowała prototypowy układ scalony realizujący algorytm DES, który szyfruje i deszyfruje dane z szybkością 1 Gbit/sek.

3.5. Klucze kryptograficzne

3.5.1. Charakterystyka kluczy kryptograficznych

Generacja klucza jest procesem, który dostarcza klucze dla systemu kryptograficznego. Sposób generowania klucza wiąże się z typem systemu kryptograficznego. W rozdziale tym będą omówione metody generowania kluczy dla systemów z kluczem tajnym, przeznaczonych głównie dla szyfrów strumieniowych. Generowanie kluczy w systemach z kluczem publicznym jest ściśle związane z zastosowanym algorytmem.

Bezpieczeństwo systemu kryptograficznego zależy od algorytmu kryptograficznego i klucza. Ponieważ algorytmy kryptograficzne nie są utajniane, nietrudno złamać szyfry, jeśli łatwo można odtworzyć klucz kryptograficzny. Słaby klucz kryptograficzny powoduje słabość całego systemu kryptograficznego. W bezpiecznych szyfrach stosuje się klucze losowe tak długie jak wiadomość.

W systemie kryptograficznym z tajnym kluczem używa się dwóch rodzajów kluczy:

- kluczy nadrzędnych do szyfrowania innych kluczy,
- kluczy do szyfrowania danych.

Klucze nadrzędne generuje się za pomocą stochastycznych procesów stacjonarnych takich jak np. rzut monetą. Inne klucze są generowane za pomocą algorytmów jako ciągi zbliżone do losowych. Klucze takie mogą być odtwarzane w komputerach, ale algorytm ich generowania należy wybierać tak, aby złamanie klucza nie było łatwe.

W systemach kryptograficznych generalnie używa się generatorów sekwencji okresowych wykorzystujących rejestry przesuwne z liniowym sprzężeniem zwrotnym, zwanych generatorami liniowymi. Konstrukcję takich generatorów opisano w p. 1.4.2. Sekwencje okresowe można scharakteryzować dwoma parametrami: długością okresu i współczynnikiem złożoności. Generowanie i właściwości sekwencji okresowych opisano w rozdziale 1.4.

Złożoność (complexity) sekwencji okresowej definiuje się jako stosunek liczby wektorów niezależnych m do całkowitej liczby wektorów w sekwencji. Ponieważ liczba wektorów w sekwencji jest równa jej okresowi, współczynnik złożoności C określa zależność

$$C = \frac{m}{\text{okres sekwencji}}, \tag{3.5.1}$$

gdzie m jest stopniem wielomianu generującego sekwencję okresową.

Okres binarnej sekwencji pseudolosowej określa wzór (1.4.2). Dla takiej sekwencji otrzymamy

$$C = \frac{m}{2^m - 1}. \tag{3.5.2}$$

Ze wzrostem m złożoność sekwencji pseudolosowej maleje, dlatego też sekwencje pseudolosowe nie nadają się bezpośrednio do konstrukcji kluczy kryptograficznych.

Kryptoanalityk, który chce złamać klucz, zwykle dąży do znalezienia wielomianu

charakterystycznego generującego klucz. W przypadku sekwencji pseudolosowych, generowanych za pomocą liniowych równań rekurencyjnych, wystarczy znać $2m$ bitów, aby znaleźć równanie charakterystyczne sekwencji. Liczba $2m$ jest bardzo mała w porównaniu z okresem sekwencji pseudolosowej, który wynosi $2^m - 1$. Dlatego też dąży się do powiększenia liczby wektorów niezależnych w sekwencji i tym samym zwiększenia jej złożoności.

Liczbę wektorów niezależnych w sekwencji okresowej można zwiększyć przez zastosowaniu generatorów nieliniowych, które będą omówione w następnym punkcie. Ponadto do generowania kluczy binarnych można wykorzystywać niektóre algorytmy kryptograficzne, np. algorytm DES. Generator oparty na algorytmie DES zastosowano w normie amerykańskiej ANSI X9.17 określającej metodę generowania kluczy.

Okresowe ciągi niebinarne da się generować również za pomocą rejestrów przesuwnych z liniowym sprzężeniem zwrotnym. Generatory takich sekwencji i ich właściwości opisano w rozdziale 1.6.

Do generowania niebinarnych ciągów okresowych używa się także generatorów kongruencyjnych (Linear Congruential Generator – LCG) pracujących według wzoru

$$X_n = \left(a\, X_{n-1} + b\right) \bmod m, \tag{3.5.3}$$

gdzie a, b i m są stałymi, a X_{n-1} i X_n – kolejnymi liczbami ciągu.

Gdy wartości stałe są odpowiednio dobrane, generator kongruencyjny osiąga okres maksymalny równy m. Klucze generowane za pomocą tych generatorów nie są kryptograficznie bezpieczne. Lepsze właściwości mają generatory będące kombinacją liniowych generatorów kongruencyjnych.

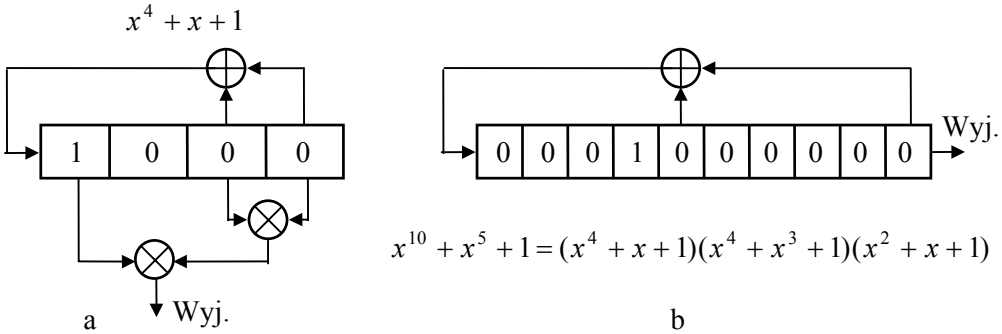
3.5.2. Generatory nieliniowe kluczy binarnych

Generatory nieliniowe kluczy binarnych konstruuje się na bazie generatorów liniowych, dodając elementy nieliniowe. Elementem nieliniowym jest np. bramka AND. Pod względem konstrukcyjnym rozróżnia się dwa typy generatorów nieliniowych.

1. Generatory z nieliniowym sprzężeniem do przodu (feed-forward nonlinear logic).
2. Generator złożony z generatorów liniowych połączonych za pomocą elementów nieliniowych.

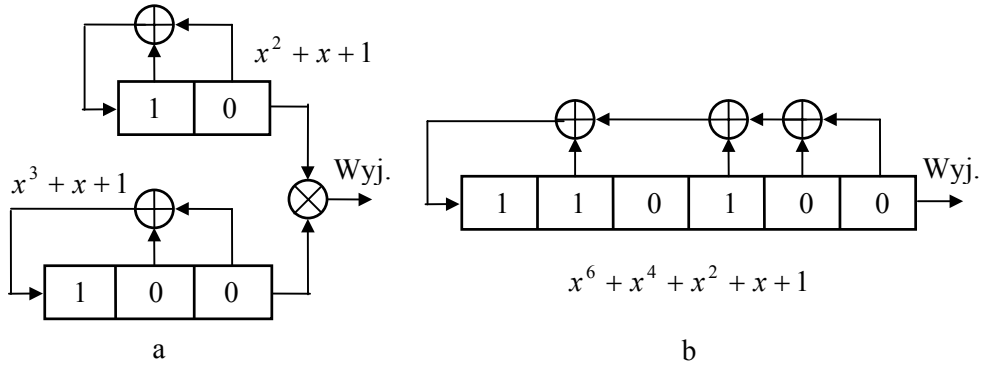
Przykłady generatorów nieliniowych pokazano na rysunkach 3.5.1 i 3.5.2. Na rysunkach obok generatorów nieliniowych znajdują się schematy równoważnych generatorów liniowych, które generują takie same sekwencje okresowe jak generatory nieliniowe. Metodę konstrukcji generatorów liniowych równoważnych układom nieliniowym opisano w p. 3.5.3.

Generator z nieliniowym sprzężeniem do przodu, pokazany na rys. 3.5.1, generuje sekwencję o okresie 15 zawierającą 10 wektorów niezależnych. Generator skonstruowany z generatorów liniowych połączonych elementem mnożącym, pokazany na rys.3.5.2, generuje sekwencję o okresie 21 zawierającą 6 wektorów niezależnych.



Rys. 3.5.1. Generator nieliniowy typ 1 (a) i jego odpowiednik liniowy (b)

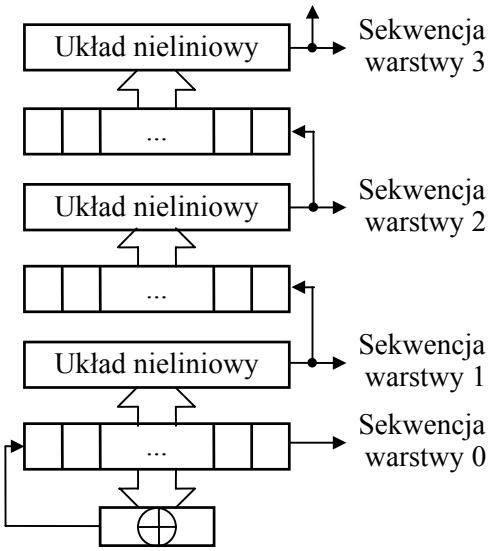
Dzięki zastosowaniu generatorów nieliniowych można zwiększyć złożoność sekwencji i polepszyć właściwości kluczy kryptograficznych. Generatory nieliniowe zawierają mniejszą liczbę elementów niż ich odpowiedniki liniowe i są stosowane ze względów ekonomicznych.



Rys. 3.5.2. Generator nieliniowy typ 2 (a) i jego odpowiednik liniowy (b)

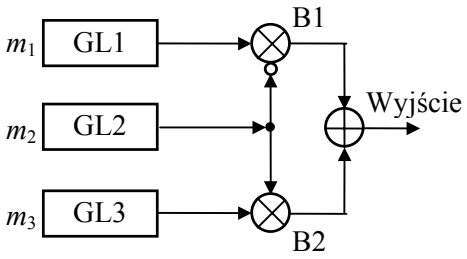
Opisane wyżej zasady konstrukcji generatorów nieliniowych wykorzystuje się do konstrukcji generatorów bardziej złożonych. Do generatora ze sprzężeniem nieliniowym do przodu można dołączyć kilka warstw zawierających elementy nieliniowe. Generator z trzema warstwami pokazano na rys. 3.5.3. W każdej warstwie jest rejestr przesuwny i układ nieliniowy. Ze wzrostem liczby warstw wzrasta liczba wektorów niezależnych w generowanej sekwencji.

Projektowanie generatorów nieliniowych opiera się na badaniach eksperymentalnych, gdyż nie ma ogólnych metod umożliwiających zaprojektowanie generatora nieliniowego o założonych właściwościach. Na podstawie wyników eksperymentalnych wiadomo, że generowane sekwencje posiadają więcej wektorów niezależnych, jeśli warstwy mają różne połączenia.



Rys. 3.5.3. Generator wielowarstwowy

Można również budować złożone generatory nieliniowe, łącząc generatory liniowe za pomocą elementów nieliniowych. Przykładem takiego generatora jest generator Gaffego pokazany na rys. 3.5.4.



Rys. 3.5.4. Generator Gaffego

Generator Gaffego zawiera trzy generatory liniowe: GL1, GL2 i GL3 połączone za pomocą sieci zawierającej elementy nieliniowe. Generator GL2 steruje przepływem sygnałów z generatorów GL1 i GL2 na wyjście układu za pomocą bramek iloczynowych B1 i B2. Na przykład, jeśli na wyjściu generatora GL2 jest jedynka, to bramka B2 jest otwarta, a B1 zamknięta. Jeśli a_1, a_2 i a_3 są bitami wyjściowymi odpowiednich generatorów liniowych, to bit wyjściowy b będzie określony wzorem

$$b = a_1 \overline{a}_2 + a_2 a_3.$$

Z powyższego wzoru widać, że układ bramek wyjściowych spełnia rolę multipleksera, a generator GL2 steruje jego pracą.

Dla generatora Gaffego można obliczyć liczbę wektorów niezależnych ciągu wyj-

ściowego. Jeśli wielomiany generatorów liniowych mają odpowiednio stopnie m_1, m_2 i m_3, to liczba wektorów niezależnych sekwencji wyjściowej wynosi

$$m_1 m_2 + (m_2 + 1) m_3 .$$

Okres sekwencji wyjściowej jest równy najmniejszej wspólnej wielokrotności okresów składowych generatorów liniowych. Gdy okresy generatorów składowych są względnie pierwsze, wtedy okres ciągu wyjściowego będzie równy iloczynowi okresów generatorów składowych.

Generator Gaffego można rozbudować w ten sposób, że w miejsce generatorów GL1, GL2 i GL3 zostaną wstawione trzy generatory Gaffego. I tak otrzymamy generator o dużym współczynniku złożoności. Do tej samej grupy co generator Gaffego należy generator V. Pless i generator multiplekserowy [3.2, 3.9].

3.5.3. Łamanie kluczy kryptograficznych

Podstawowe metody łamania szyfrów omówiono w p. 3.1.4. Głównym elementem procesu łamania szyfru jest odtworzenie klucza kryptograficznego. Klucze losowe można złamać metodą wypróbowywania każdego możliwego klucza, co określa się *łamaniem brutalnym*.

W przypadku kluczy generowanych za pomocą algorytmów komputerowych dąży się do znalezienia algorytmu i jego parametrów. Podczas łamania klucza wykorzystuje się jego okresowość i redundancję. Kryptoanalitycy w swojej pracy przeważnie opierają się na fragmentach kryptogramów i fragmentach tekstów jawnych, które umożliwiają odtworzenie fragmentów kluczy kryptograficznych, a następnie wyznaczenie algorytmów ich generowania.

Problem łamania klucza kryptograficznego, generowanego komputerowo, sprowadza się do znalezienia wielomianu charakterystycznego sekwencji klucza. Rozważmy klucz kryptograficzny generowany przez generator liniowy. Jak podano w p. 3.5.1, aby znaleźć wielomian charakterystyczny sekwencji okresowej, wystarczy znać co najmniej 2m bitów tej sekwencji. Mając fragment sekwencji okresowej $s_0 s_1 s_2 \ldots s_{2m-1}$, można napisać układ m równań zawierający m niewiadomych:

$$\begin{aligned} a_0 s_0 + a_1 s_1 + \ldots + a_m s_m &= 0, \\ a_0 s_1 + a_1 s_2 + \ldots + a_m s_{m+1} &= 0, \\ &\ldots \\ a_0 s_{m-1} + a_1 s_m + \ldots + a_m s_{2m-1} &= 0. \end{aligned} \tag{3.5.4}$$

Rozwiązanie powyższego układu równań pozwoli znaleźć wielomian generujący sekwencję okresową stopnia m, lecz wymaga ono zastosowania m^3 operacji i m^2 komórek pamięci. Metodę tę ilustruje przykład 3.5.1.

P r z y k ł a d 3.5.1.

Wyznaczanie wielomianu charakterystycznego binarnej sekwencji okresowej.

Zakładamy, że mamy fragment sekwencji okresowej 1000100110… . Wyznaczmy współczynniki wielomianu stopnia piątego generującego tą sekwencję

$$a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0.$$

Układ równań (3.5.4) napiszemy, podstawiając elementy sekwencji okresowej. Stosujemy w tym celu poniższą tabelę.

1	0	0	0	1	0	0	1	1	0	Równania
a_0	a_1	a_2	a_3	a_4	a_5					$a_0 + a_4 = 0$
	a_0	a_1	a_2	a_3	a_4	a_5				$a_3 = 0$
		a_0	a_1	a_2	a_3	a_4	a_5			$a_2 + a_5 = 0$
			a_0	a_1	a_2	a_3	a_4	a_5		$a_1 + a_4 + a_5 = 0$
				a_0	a_1	a_2	a_3	a_4	a_5	$a_0 + a_3 + a_4 = 0$

W pierwszym wierszu tabeli podano elementy sekwencji okresowej $s_0\, s_1\, s_2 \ldots s_9$, a w ostatniej kolumnie – równania (3.5.4). Jeśli założymy, że $a_5 = 1$ i $a_0 = 1$, to po rozwiązaniu układu równań otrzymamy: $a_4 = 1$, $a_3 = 0$, $a_2 = 1$ i $a_1 = 0$. Wielomian generujący sekwencję ma postać

$$x^5 + x^4 + x^2 + 1 = \left(x^4 + x + 1\right)\left(x + 1\right).$$

Otrzymany wielomian piątego stopnia można rozłożyć na dwa wielomiany: stopnia czwartego i pierwszego. Łatwo sprawdzić, że wielomiany stopnia piątego i czwartego generują taką samą sekwencję okresową. Wielomian stopnia czwartego można również otrzymać bezpośrednio, zakładając $a_5 = 0$.

Kryptoanalityk poszukujący równania charakterystycznego klucza zwykle nie zna jego stopnia. Może on założyć stopień równania, a następnie, jeśli uzyska rozwiązanie, zredukować jego stopień, rozkładając je na wielomiany nierozkładalne.

Znany jest również algorytm Berlekampa-Masseya [1.2], który umożliwia znalezienie bezpośrednio równania charakterystycznego klucza najniższego stopnia.

3.5.4. Zarządzanie kluczami

Zarządzanie kluczami kryptograficznymi jest najbardziej kłopotliwym problemem kryptografii komputerowej. W systemie komputerowym, pracującym w środowisku wielozadaniowym, może nastąpić wiele zdarzeń przypadkowych prowadzących do ujawnienia lub utraty kluczy. Łatwiej jest wykonać kryptograficzne implementacje sprzętowe. Prostsze jest również zarządzanie kluczami jawnymi.

Zastosowanie kryptografii z kluczami tajnymi w systemie komputerowym wymaga opracowania reguł bezpiecznej obsługi i sterowania kluczami. Zbiór takich reguł na-

zywa się *protokołem zarządzania kluczami*.

Najłatwiejszym do rozwiązania problemem jest przechowywanie kluczy do szyfrowania plików należących do pojedynczych użytkowników. Najprościej jest wówczas przechowywać klucz poza systemem komputerowym. Klucze są wprowadzane do systemu komputerowego tylko w czasie szyfrowania i deszyfrowania plików. Klucze mogą być przechowywane w układzie scalonym typu ROM lub na karcie magnetycznej, z której będą odczytywane za pomocą czytnika dołączonego do komputera. Wtedy do rozwiązania pozostaje tylko problem bezpiecznego kasowania kluczy po ich użyciu.

Specjalnej ochrony wymagają klucze umożliwiające dostęp do informacji wielu użytkownikom, jak np. dostęp do baz danych. Wtedy lepiej jest przechowywać klucze w systemie, pod ochroną osoby zarządzającej kluczami. Klucze przechowywane u wielu użytkowników są narażone na zgubienie lub ujawnienie. W tym wypadku dobrym rozwiązaniem jest przechowywanie kluczy w oddzielnym pliku w postaci zaszyfrowanej.

Innego rozwiązania wymaga problem zarządzania kluczami w sieci komputerowej. W tradycyjnej kryptografii klucze kryptograficzne były generowane w jednym miejscu i rozsyłane za pośrednictwem kurierów. Rozwiązania takiego nie da się zaakceptować w nowoczesnych systemach informatycznych zawierających wiele węzłów i terminali. Najwygodniejszą metodą jest rozsyłanie kluczy za pomocą samej sieci.

Koncepcję zarządzania kluczami przeznaczonymi do ochrony transmisji danych i plików w systemach stosujących algorytm DES opracowano w IBM i opisano w [3.2, 3.4 i 3.8]. Według tej koncepcji stosuje się hierarchiczną strukturę kluczy. Ogólnie dla zabezpieczenia transmisji i zbiorów danych korzysta się z następujących kluczy:

- klucz główny komputera $KM0$,
- pierwszy wariant klucza głównego komputera $KM1$,
- drugi wariant klucza głównego komputera $KM2$,
- klucz główny terminala KMT,
- pierwotny klucz komunikacyjny lub klucz sesji KS,
- wtórny klucz komunikacyjny KNS,
- pierwotny klucz pliku lub klucz pliku KF,
- wtórny klucz pliku KNF.

Komputer główny ma klucz główny $KM0$ przechowywany w zabezpieczonym obszarze pamięci i przesyłany przez kuriera. Klucz ten służy do generowania kluczy $KM1$ i $KM2$, potrzebnych do szyfrowania innych kluczy.

Każdy terminal ma swój klucz główny terminala *KMT* przechowywany w chronionym obszarze pamięci i umożliwiający utworzenie zabezpieczonego kanału między terminalem a komputerem, który służy do transmisji kluczy szyfrujących informację, kluczy sesji *KS*. W systemie komputera głównego kopie tych kluczy są przechowywane w postaci zaszyfrowanej kluczem $KM1$. Klucz $KM2$ służy do szyfrowania kluczy plików *KF*.

Implementacja protokółu wymaga określenia operacji kryptograficznych w komputerze i terminalach. W komputerze stosuje się następujące operacje:

1. Instalacja klucza głównego (set master key)

$$smk\{KM0\}.$$

2. Szyfrowanie klucza K kluczem głównym $KM0$ (encipher under master key)

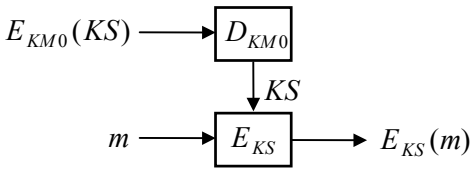
$$emk\{K\} \rightarrow E_{KM0}(K).$$

W nawiasie podano dane wejściowe, a po strzałce wynik operacji kryptograficznej.

3. Szyfrowanie danych (encipher)

$$ecph\{E_{KM0}(KS), m\} \rightarrow E_{KS}(m).$$

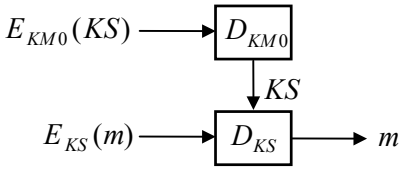
Wykonanie operacji szyfrowania wymaga przekazania do szyfratora klucza sesji KS zaszyfrowanego pod kluczem $KM0$. Tam następuje deszyfracja klucza sesji i szyfrowanie danych m. Schemat operacji pokazano na rys. 3.5.5.



Rys. 3.5.5. Operacja szyfrowania danych

4. Deszyfrowanie danych (decipher)

$$dcph\{E_{KM0}(KS), E_{KS}(m)\} \rightarrow m.$$

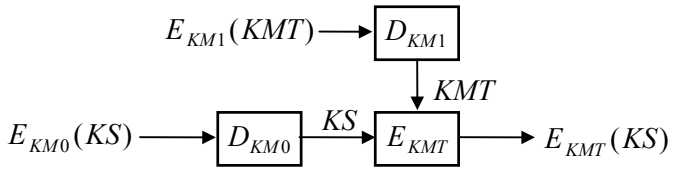


Rys. 3.5.6. Operacja deszyfrowania danych

5. Przeszyfrowanie z klucza głównego (reencipher from mastery key)

$$rfmk\{E_{MK1}(KMT), E_{KM0}(KS)\} \rightarrow E_{KMT}(KS).$$

Główny klucz terminala KMT jest przechowywany w postaci zaszyfrowanej kluczem $KM1$. Operacja $rfmk$ umożliwia pobranie klucza sesji KS zaszyfrowanego kluczem $KM0$ i zapisania go w postaci zaszyfrowanej kluczem KMT. Klucz KS w takiej postaci jest przesyłany do terminala. Schemat operacji pokazano na rys. 3.5.7.

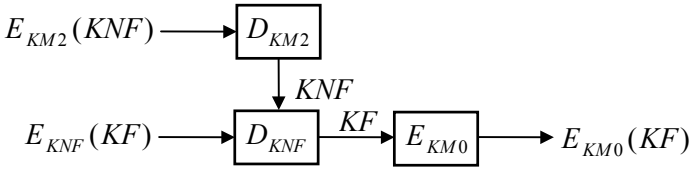


Rys. 3.5.7. Operacja przeszyfrowania z klucza głównego

5. Przeszyfrowanie na klucz główny (reencipher to master key)

$$rtmk\{E_{KM2}(KNF), E_{KNF}(KF)\} \rightarrow E_{KM0}(KF).$$

Klucz szyfrujący plik KF jest szyfrowany kluczem $KM0$. Aby ograniczyć dostęp do tego klucza, przechowuje się go w postaci zaszyfrowanej kluczem $KM2$. Użycie klucza KF wymaga przeszyfrowania go na klucz główny komputera. Schemat operacji pokazano na rys 3.5.8.



Rys. 3.5.8. Operacja przeszyfrowania na klucz główny

W terminalach stosowane są następujące operacje:

1. Deszyfrowanie klucza sesji z klucza głównego (decipher from master key)

$$dmk\{E_{KMT}(KS) \rightarrow KS.$$

2. Szyfrowanie danych

$$ecph\{m\} \rightarrow E_{KS}(m).$$

3. Deszyfrowanie danych

$$dcph\{E_{KS}(m)\} \rightarrow m.$$

Filozofia tego protokółu opiera się na zasadzie, że każdy klucz w systemie kryptograficznym jest używany krótko, co uniemożliwia kryptoanalitykowi zgromadzenie dostatecznie dużej ilości materiału potrzebnego do złamania szyfru.

W zwykłych kanałach telekomunikacyjnych podsłuch bierny nie powoduje mierzalnych zmian sygnału i nie da się go wykryć. W systemach informatycznych można wykorzystywać kanały optyczne, a informacje przesyłać w postaci spolaryzowanych strumieni fotonów. Wówczas realne jest zaprojektowanie kanałów komunikacyjnych w taki sposób, że każda próba podsłuchu może być wykryta. Takie kanały kwantowe

mogą być stosowane do bezpiecznej dystrybucji kluczy kryptograficznych. Kryptografia kwantowa znajduje się jeszcze na etapie badań i dopiero w przyszłości można się spodziewać praktycznego zastosowania rozwiązań tego typu.

3.6. Szyfry z kluczem jawnym

3.6.1. Charakterystyka algorytmów z kluczem jawnym

System kryptograficzny z kluczem jawnym opisano w p. 3.2.2. Algorytmy z kluczem jawnym lub publicznym są nazywane algorytmami asymetrycznymi w odróżnieniu od algorytmów symetrycznych z kluczem tajnym. Pierwsze algorytmy z kluczem jawnym opublikowano w tym samym czasie, kiedy dyskutowano o algorytmie DES jako o proponowanym standardzie. Od 1976 r. zaproponowano wiele algorytmów z kluczem jawnym. Tylko kilka z nich jest bezpiecznych i praktycznych.

Większość algorytmów z kluczem jawnym bazuje na jednym z trzech problemów trudnych, którymi są:

- problem plecakowy,
- problem logarytmu dyskretnego,
- problem faktoryzacji.

Siła dowolnego algorytmu z kluczem jawnym zależy od złożoności obliczeniowej problemu, na którym on bazuje.

Pierwszy algorytm kryptograficzny z kluczem jawnym lub publicznym opracowali Merkle i Hellman [3.5] w 1978 r. Zaproponowali oni szyfr, którego bezpieczeństwo opiera się na problemie plecaka, będącego zagadnieniem **NP**-zupełnym. Pojęcie **NP** pochodzi z teorii złożoności. Teoria ta umożliwia analizowanie złożoności obliczeniowej algorytmów kryptograficznych. Zwykle algorytmy klasyfikuje się w zależności od ich złożoności czasowej lub przestrzennej wyrażonej jako funkcja argumentu n. Algorytm jest liniowy, jeśli jego złożoność rośnie liniowo ze wzrostem n. Podobnie algorytmy mogą być kwadratowe, sześcienne, wykładnicze itd. Algorytmy z kluczem jawnym zwykle mają złożoność wykładniczą.

Trudność obliczenia logarytmu dyskretnego wykorzystano, między innymi, w algorytmie ElGamala, a problem faktoryzacji w algorytmie RSA. Z bezpiecznych i praktycznych algorytmów z kluczem jawnym niektóre nadają się do szyfrowania danych, inne zaś do podpisów cyfrowych. Tylko dwa algorytmy, tj. RSA i ElGamala, są odpowiednie zarówno do szyfrowania danych, jak i podpisów cyfrowych. Do podpisów cyfrowych preferowany jest algorytm DSA (Digital Signature Algorithm). Wszystkie algorytmy z kluczem jawnym są w realizacji algorytmami wolnymi w porównaniu z algorytmami symetrycznymi.

3.6.2. Algorytm Merklego–Hellmana

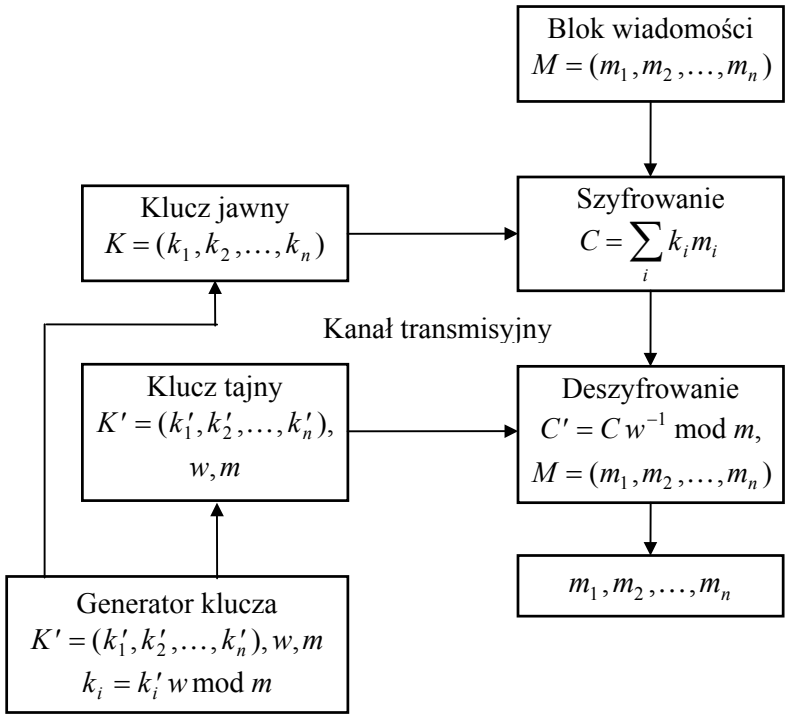
W algorytmie z kluczem jawnym Merklego–Hellmana wykorzystano problem plecaka. Problem plecakowy, zwany też problemem upakowania, jest następujący. Mamy nieuporządkowany zbiór przedmiotów, każdy o innej wadze. Z tego zbioru należy wybrać podzbiór o zadanej z góry łącznej wadze.

Problem plecakowy można sformułować w dziedzinie liczb. Mamy dodatnią liczbę całkowitą s oraz zbiór dodatnich liczb całkowitych $a=(a_1,a_2,\ldots,a_n)$. Szukamy takiego podzbioru zbioru a, którego suma wynosi s. Inaczej, dla danego ciągu a szukamy wektora dwójkowego $m=(m_1,m_2,\ldots,m_n)$ spełniającego równanie

$$s=\sum_{i=1}^{n} a_i\, m_i .$$

Wartości m_i mogą być 0 lub1. Wartość 1 wskazuje, że dany element jest składnikiem sumy, a wartość 0, że nie jest składnikiem sumy. Na przykład jeśli a=(1191, 171, 459, 4517, 2410, 197), a s=3798, to metodą prób i błędów można znaleźć składniki sumy s=1191+2410+197. Wtedy m=(1, 0, 0, 0, 1, 1).

Znalezienie właściwego podzbioru jest zagadnieniem trudnym, gdyż należy przeszukać 2^n możliwych podzbiorów i nie ma lepszego algorytmu. Złożoność takiego problemu opisuje funkcja wykładnicza. Dla $n=200$ zadanie staje się nierozwiązalne metodą ataku brutalnego. Jeśli natomiast liczba elementów będzie 1000, to liczba możliwych podzbiorów wynosi 2^{1000}, co jest większą liczbą niż liczba atomów we wszechświecie szacowaną na 2^{256}.



Rys. 3.6.1. Algorytm Merklego–Hellmana

Merkle i Hellman pokazali, że problem plecakowy o złożoności wykładniczej można przekształcić w problem o złożoności liniowej. W tym celu generuje się dwa klucze: jawny służący do szyfrowania i tajny używany w procesie deszyfrowania. Deszyfrowanie z kluczem jawnym jest problemem o złożoności wykładniczej i złamanie szyfru staje się bardzo trudne. Gdy do deszyfrowania używamy klucza tajnego, wtedy problem ma złożoność liniową i łatwo go rozwiązać.

Schemat blokowy algorytmu Merklego–Hellmana przedstawiono na rys. 3.6.1. Klucz tajny jest generowany za pomocą generatora liczb losowych i zawiera ciąg liczb losowych $K' = (k_1', k_2', \ldots, k_n')$ oraz liczbę pierwszą m i liczbę losową $w < m$.

Elementy ciągu K' są tak wybrane, aby każdy element ciągu był większy od sumy wszystkich poprzedzających go elementów. Ciąg taki nazywamy ciągiem superrosnącym. Liczba pierwsza m i liczba losowa w spełniają zależności:

$$\begin{gathered} m > \sum_{i=1}^{n} k_i', \\ 1 < w < m. \end{gathered} \tag{3.6.1}$$

Dla liczby w oblicza się odwrotność w^{-1} modulo m za pomocą wzoru

$$w\, w^{-1} \left(\mathrm{mod}\ m\right) \equiv 1. \tag{3.6.2}$$

Sposób obliczania tej odwrotności pokazano w przykładzie 3.3.2.

Elementy klucza jawnego są obliczane z ciągu K' z zależności

$$k_i = k_i'\, w \bmod m. \tag{3.6.3}$$

Klucz jawny przesyła się do nadawcy kryptogramów kanałem niezabezpieczonym.

Szyfr Merklego–Hellmana jest szyfrem blokowym, szyfrującym ciągi binarne. Ciąg informacyjny $m_1, m_2, \ldots$ dzieli się na bloki $M = (m_1, m_2, \ldots, m_n)$, zawierające po n bitów. Szyfrowanie bloku odbywa się z udziałem klucza jawnego K według wzoru

$$C = \sum_i k_i m_i . \tag{3.6.4}$$

W procesie deszyfrowania wykorzystuje się klucz tajny. Podczas deszyfracji najpierw oblicza się C'

$$C' = C\, w^{-1} \bmod m. \tag{3.6.5}$$

Znając C' i klucz tajny K', możemy wyznaczyć elementy wiadomości. Proces szyfrowania i deszyfrowania ilustruje przykład 3.6.1.

Poprawność algorytmu Merklego–Hellmana sprawdzamy podstawiając (3.6.4) i (3.6.3) do (3.6.5)

$$C' = w^{-1} \sum_{i=1}^{n} k_i m_i \bmod m = \sum_{i=1}^{n} \left(k_i' m_i w w^{-1} \bmod m\right) \bmod m.$$

Wykorzystując (3.6.2) otrzymamy

$$C' = \sum_{i=1}^{n} k_i' m_i \bmod m.$$

Ponieważ $m > \sum_{i=1}^{n} k_i'$, możemy napisać

$$C' = \sum_{i=1}^{n} k_i' m_i .$$

P r z y k ł a d 3.6.1.

Szyfr Merklego–Hellmana.

Stosując algorytm Merklego–Hellmana, wykonać szyfrowanie i deszyfrowanie ciągu binarnego $M = (1,0,0,1,1,1,0,1)$. Przyjąć, że wygenerowane elementy klucza tajnego wynoszą: $K' = (3, 5, 11, 21, 39, 77, 153, 311)$, $w = 167$, $m = 673$.

Obliczamy w^{-1}, korzystając z algorytmu Euklidesa. Najpierw wyznaczamy $NWD(167,673)$.

$$673 = 4 \cdot 167 + 5,$$

$$167 = 33 \cdot 5 + 2,$$

$$5 = 2 \cdot 2 + 1.$$

Aby znaleźć w^{-1}, należy rozwiązać równanie (3.3.7)

$$w^{-1} \cdot 167 - t \cdot 673 = 1.$$

W celu przedstawienia liczby 1 jako kombinacji liniowej liczb 167 i 673 wykorzystujemy ciąg równości od ostatniej do pierwszej w algorytmie Euklidesa. W każdym kroku zapisujemy liczbę 1 za pomocą wcześniejszych reszt, aż dojdziemy do samych liczb 167 i 673

$$1 = 5 - 2 \cdot 2 =$$

$$= 5 - 2(167 - 33 \cdot 5) = 67 \cdot 5 - 2 \cdot 167 =$$

$$= 67(673 - 4 \cdot 167) - 2 \cdot 167 = 67 \cdot 673 - 270 \cdot 167.$$

Zmiana znaków liczb w ostatnim wyrażeniu wymaga dodania i odjęcia od tego wyrażenia liczby równej $167 \cdot 673$.

$$1 = 67 \cdot 673 - 167 \cdot 673 + 673 \cdot 167 - 270 \cdot 167 = 403 \cdot 167 - 100 \cdot 673.$$

Szukana odwrotność wynosi $w^{-1} = 403$.

Elementy klucza jawnego obliczamy, wykorzystując klucz tajny i zależność (3.6.3). Kluczem jawnym będzie ciąg $K = (501,162,491,142,456,72,650,116)$.

Kryptogram obliczamy z (3.6.4)

$$C = \sum_{i=1}^{8} k_i m_i = 501 + 142 + 456 + 72 + 116 = 1287.$$

W procesie deszyfrowania najpierw obliczamy C' z (3.6.5)

$$C' = C w^{-1} \bmod m = 1287 \cdot 403 \bmod 673 = 451.$$

Wykorzystując klucz tajny K', możemy wyrazić liczbę C' jako sumę elementów tego klucza. Biorąc elementy klucza od ostatniego do pierwszego, stwierdzamy, że liczba 451 jest sumą liczb: 311, 77, 39, 21 i 3, co zapisujemy w postaci

$$451 = 1 \cdot 3 + 0 \cdot 5 + 0 \cdot 11 + 1 \cdot 21 + 1 \cdot 39 + 1 \cdot 77 + 0 \cdot 153 + 1 \cdot 311.$$

Informacja otrzymana w wyniku deszyfracji będzie zatem równa $M = (1,0,0,1,1,1,0,1)$.

Szyfr Merklego–Hellmana został złamany cztery lata po jego opublikowaniu. Okazało się, że na podstawie klucza jawnego można znaleźć klucz zbliżony do tajnego i umożliwiający zdeszyfrowanie zaszyfrowanych wiadomości. Oprócz szyfru Merklego–Hellmana opracowano inne szyfry plecakowe, które zostały również złamane. Obecnie bezpieczną odmianą szyfru plecakowego jest algorytm Chora–Rivesta. Jednak realizacja tego algorytmu wymaga tylu obliczeń, że jest on mało przydatny w praktyce.

3.6.3. Algorytm ElGamala

Algorytm ElGamala może być używany zarówno do szyfrowania danych, jak i podpisów cyfrowych [3.9]. Moc algorytmu wynika z trudności obliczania logarytmów dyskretnych. Problem ten można sformułować w następujący sposób. Jeśli p jest liczbą pierwszą, a g i m są liczbami całkowitymi, to należy znaleźć takie x, że $g^x = m \bmod p$.

Schemat blokowy algorytmu ElGamala, zastosowanego do podpisywania dokumentów, pokazano na rys 3.6.2. Aby wygenerować parę kluczy, jawny i tajny, losujemy liczbę pierwszą p oraz dwie liczby g i x mniejsze od p. Następnie obliczamy

$$y = g^x \bmod p. \tag{3.6.6}$$

Kluczem publicznym są liczby: p, g i y, a kluczem tajnym jest x. Liczby p i g mogą być wykorzystywane przez grupę użytkowników.

W celu podpisania wiadomości $m \in \{0, p-1\}$ nadawca losuje liczbę k względnie pierwszą z liczbą $p-1$ i oblicza

$$a = g^k \bmod p. \tag{3.6.7}$$

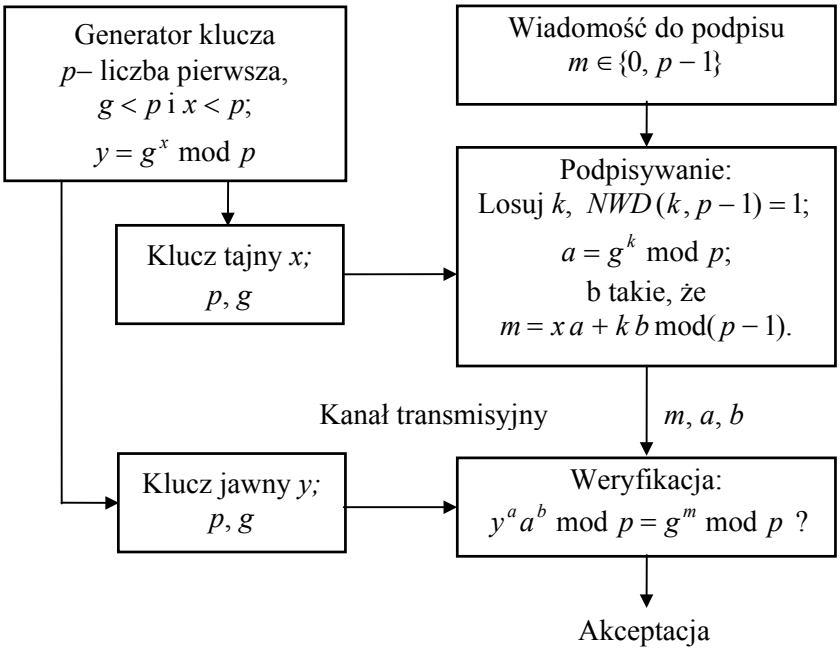
Następnie oblicza on liczbę b z następującego równania

$$m = (x\,a + k\,b) \bmod (p-1). \tag{3.6.8}$$

Do rozwiązana równania stosuje się rozszerzony algorytm Euklidesa opisany w [3.2]. Podpis stanowi para liczb a i b. Liczba k musi być utrzymywana w tajemnicy, gdyż jest ona prywatnym kluczem podpisującego dokument.

Odbiorca dokumentu weryfikuje podpis, sprawdzając równość

$$y^a a^b \bmod p = g^m \bmod p. \tag{3.6.9}$$



Rys. 3.6.2. Algorytm ElGamala do podpisów cyfrowych

P r z y k ł a d 3.6.2.
Algorytm ElGamala do podpisów cyfrowych.
Zakładamy $p = 11,\ g = 2$ oraz $x = 8.$ Obliczamy y z (3.6.6)

$$y = g^x \bmod p = 2^8 \bmod 11 = 3.$$

Kluczem jawnym są liczby: $p = 11,\ g = 2$ i $y = 3.$

Aby podpisać wiadomość $m = 5,$ wybieramy liczbę $k = 9$ i sprawdzamy, że

$$NWD(k, p-1) = NWD(9,10) = 1.$$

Obliczamy a z (3.6.7)

$$a = g^k \bmod p = 2^9 \bmod 11 = 6$$

oraz liczbę b z równania (3.6.8)

$$m = (a x + b k) \bmod (p-1);$$

$$5 = (8 \cdot 6 + 9b) \bmod 10, \quad b = 3.$$

Liczbę b można znaleźć numerycznie, rozwiązując równanie $8 \cdot 6 + 9b - 10x = 5$ w dziedzinie liczb całkowitych.

Aby sprawdzić podpis, potwierdzamy, że jest spełnione równanie (3.6.9)

$$y^a a^b \bmod p = g^m \bmod p,$$

$$3^6 6^3 \bmod 11 = 2^5 \bmod 11.$$

Algorytm ElGamala można stosować również do szyfrowania wiadomości m wyrażonych w postaci liczb należących do zbioru $\{0, p-1\}$. Algorytm szyfrowania jest podobny do algorytmu podpisywania. Różnica polega na tym, że zamiast podpisywania i weryfikacji podpisu stosuje się operacje szyfrowania i deszyfrowania, a generator klucza jest umieszczony po stronie odbiorczej. Kryptogram złożony z liczb a i b obliczamy z zależności:

$$\begin{aligned} a &= g^k \bmod p, \\ b &= m y^k \bmod p. \end{aligned} \tag{3.6.10}$$

Do wyznaczenia postaci jawnej wiadomości m z kryptogramu stosuje się wyrażenie

$$m = \frac{b}{a^x} \bmod p. \tag{3.6.11}$$

Odwrotność liczby a^x modulo p obliczamy z (3.3.7). Poprawność algorytmu wynika z przekształcenia

$$\frac{b}{a^x} = \frac{m y^k}{a^x} = \frac{m g^{xk}}{g^{kx}} = m \bmod p.$$

3.6.4. Algorytm RSA

Algorytm Rivesta, Shamira i Adelmana, zwany algorytmem RSA, pojawił się wkrótce po opublikowaniu algorytmu Merklego–Hellmana [3.5]. Algorytm ten nadaje się do szyfrowania danych i podpisów cyfrowych. Jest on algorytmem wykładniczym z kluczem publicznym. Bezpieczeństwo szyfru RSA opiera się na trudności faktoryzacji dużych liczb.

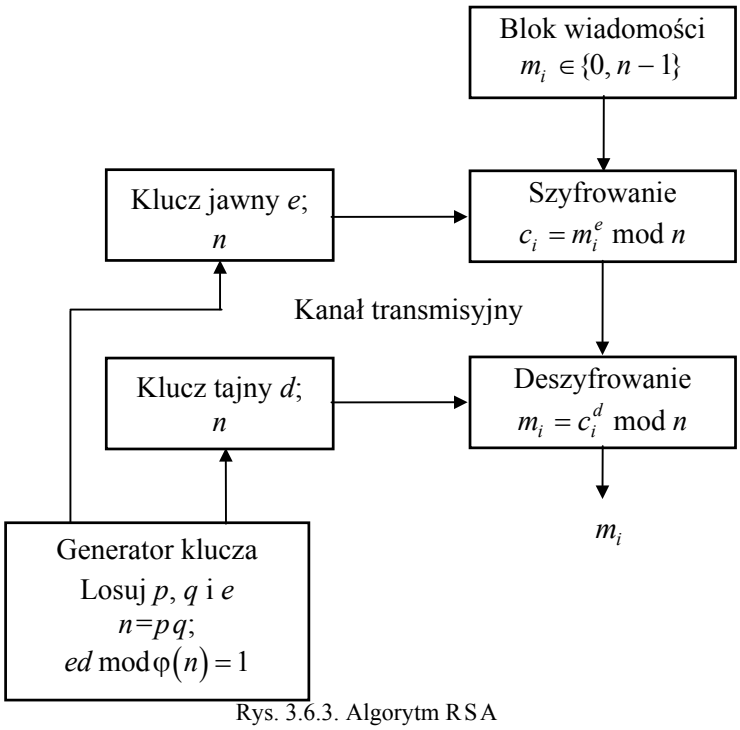
Schemat blokowy algorytmu pokazano na rys. 3.6.3. Klucze tajny i jawny szyfru RSA są funkcjami pary dużych liczb pierwszych. W celu wygenerowania kluczy odbiorca kryptogramu losuje dwie liczby pierwsze p i q oraz liczbę e. Iloczyn $n = p\,q$ oraz e stanowią publiczny klucz szyfrujący. Liczba e musi być względnie pierwsza z funkcją Eulera $\varphi(n)$. Spełnione są więc zależności:

$$n = pq,$$
$$\varphi(n) = (p-1)(q-1), \tag{3.6.12}$$
$$NWD(e, \varphi(n)) = 1.$$

Kluczem tajnym, służącym do deszyfracji kryptogramu, jest liczba d. Liczba d spełnia zależność

$$e\,d \bmod \varphi(n) = 1. \tag{3.6.13}$$

Po obliczeniu kluczy liczby p i q należy wymazać z systemu, aby nie zostały ujawnione.



Rys. 3.6.3. Algorytm RSA

Aby przeprowadzić szyfrowanie, bloki tekstu jawnego przedstawia się w postaci liczb z zakresu od 0 do $n-1$. Elementy kryptogramu c_i odpowiadające wiadomości m_i oblicza się z zależności

$$c_i = m_i^e \bmod n. \tag{3.6.14}$$

Mając kryptogram c_i, możemy obliczyć wiadomość m_i z równania

$$m_i = c_i^d \bmod n. \tag{3.6.15}$$

Wiadomość można również zaszyfrować za pomocą liczby d, a rozszyfrować za pomocą liczby e.

Funkcję szyfrującą realizuje następujący algorytm numeryczny:

1. Liczbę *e* przedstawiamy w postaci binarnej $e = e_k, e_{k-1}, \ldots, e_1, e_0$.
2. Podstawiamy $c_i = 1$.
3. Powtarzamy w pętli od 0 do *k* następujące operacje:
   - $c_i = c_i^2 \bmod n$,
   - jeżeli $e_i = 1$, to $c_i = c_i m \bmod n$.

Funkcje szyfrująca i deszyfrująca są wzajemnie odwrotne. Poprawność algorytmu można pokazać podstawiając (3.6.14) do (3.6.15).

$$m_i = m_i^{ed} \bmod n.$$

Z (3.6.13) wynika, że $ed = t\varphi(n) + 1$ dla pewnej liczby całkowitej *t*, tak więc

$$m_i^{t\varphi(n)+1} \bmod n = m_i m_i^{t\varphi(n)} \bmod n = m_i \left(m_i^{\varphi(n)} \bmod n\right)^t \bmod n.$$

Wykorzystując uogólnienie Eulera małego twierdzenia Fermata (1.1.3), otrzymamy

$$m_i = m_i^{ed} \bmod n = m_i \bmod n = m_i.$$

Na bezpieczeństwo szyfru wpływa dobór liczb pierwszych *p* i *q*. Powinny to być liczby o długości powyżej stu cyfr. W celu lepszego zabezpieczenia *n* przed rozkładem na czynniki pierwsze zaleca się, aby:

– liczby *p* i *q* różniły się o kilka cyfr,

– zarówno $p-1$, jak i $q-1$ miały duże czynniki pierwsze,

– wartość $NWD(p-1, q-1)$ była niewielka.

W realizacji praktycznej algorytmu występuje problem wyboru algorytmu generowania liczb pierwszych i realizacji funkcji szyfrującej. Do generowania dużych liczb pierwszych nie nadaje się klasyczna metoda sita Eratostenesa. Również nie powinno się używać powszechnie znanych liczb pierwszych, takich jak np. liczby pierwsze Mersenna lub Fermata.

W kryptografii poszukuje się liczb pierwszych w ten sposób, że za pomocą generatora losowego losuje się liczby nieparzyste, a następnie poddaje się je testom pierwszości. Jeśli wylosowana liczba przechodzi pomyślnie te testy, to jest bardzo prawdopodobne, że jest ona liczbą pierwszą.

Znanych jest kilka testów pierwszości. Jeden z tych testów opiera się na małym twierdzeniu Fermata (1.1.1), które mówi, że liczba pierwsza *p* spełnia zależność

$$a^{p-1} \bmod p = 1,$$

dla *a* całkowitych niepodzielnych przez *p*. Na przykład dla $p = 5$ i $a = 4$, otrzymamy $4^4 = 256 \pmod 5 \equiv 1$.

P r z y k ł a d 3.6.3.

Algorytm RSA.

Stosując algorytm RSA, wykonać szyfrowanie i deszyfrowanie informacji $m_i = 3$. Przyjąć, że wygenerowane elementy klucza tajnego wynoszą: $p = 5$, $q = 7$ i $e = 11$. Dla przyjętych wyżej wartości p, q i e ze wzorów (3.6.12) otrzymamy: $n = pq = 35$ i $\varphi(n) = (p-1)(q-1) = 24$.

Obliczamy d, wykorzystując algorytm Euklidesa. Najpierw wyznaczamy $NWD(11,24)$.

$$24 = 2 \cdot 11 + 2,$$

$$11 = 5 \cdot 2 + 1.$$

Aby znaleźć d, należy rozwiązać równanie podobne do (3.3.7)

$$ed - t\varphi(n) = 1,$$

$$11d - t24 = 1$$

W celu przedstawienia liczby 1 jako kombinacji liniowej liczb 11 i 24 wykorzystujemy ciąg równości od ostatniej do pierwszej w algorytmie Euklidesa. W każdym kroku zapisujemy liczbą 1 za pomocą wcześniejszych reszt, aż dojdziemy do samych liczb 11 i 24

$$1 = 11 - 5 \cdot 2 =$$

$$= 11 - 5(24 - 2 \cdot 11) = 11 \cdot 11 - 5 \cdot 24.$$

Szukana liczba d wynosi 11.

Szyfrowanie informacji przeprowadzamy według wzoru (3.6.14)

$$c_i = m_i^e \bmod n = 3^{11} \bmod 35 = 177147 \bmod 35 = 12.$$

Deszyfrowanie kryptogramu odbywa się według (3.6.15)

$$m_i = c_i^d \bmod n = 12^{11} \bmod 35 = 743008370688 \bmod 35 = 3.$$

W 1994 r. szyfr RSA został złamany. Odczytano wtedy zdanie zaszyfrowane przez twórców RSA siedemnaście lat wcześniej. Pracowało nad tym 600 osób na pięciu kontynentach, przez osiem miesięcy, za pomocą sieci Internet. Mimo to algorytm RSA jest obecnie uważany za algorytm bezpieczny.

Algorytm RSA został zrealizowany sprzętowo przez wiele firm. Największa szybkość transmisji, jaką osiągnięto w realizacji sprzętowej, wynosi 64 Kbit/s w blokach 512-bitowych. Algorytm RSA jest około 1000 razy wolniejszy od algorytmu DES w realizacji sprzętowej, a około 100 razy wolniejszy w realizacji programowej. Algorytm RSA jest standardem do podpisów cyfrowych Międzynarodowej Organizacji Normalizacyjnej ISO (International Standards Organisation), ISO/IEC 9796.

3.7. Techniki szyfrowania i implementacje

3.7.1. Szyfry strumieniowe

Algorytmy kryptograficzne wykorzystuje się w różny sposób zależnie od potrzeb użytkownika. Szyfry ze względu na sposób implementacji można podzielić na:

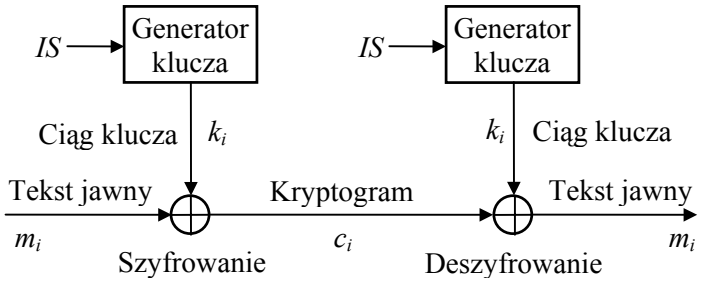
- szyfry strumieniowe (stream ciphers),
- szyfry blokowe (block ciphers).

W szyfrowaniu strumieniowym lub potokowym przetwarzaną jednostką jest bit lub znak, a w szyfrowaniu blokowym – blok, zawierający najczęściej od kilku do kilkunastu znaków. Obie te metody szyfrowania mogą być używane zarówno do transmisji, jak i do przechowywania danych w pamięciach. Przyjęta technika szyfrowania wynika najczęściej z założonego algorytmu kryptograficznego i ma wpływ na konstrukcję układu szyfratora i deszyfratora. Szyfry blokowe opisano w p. 3.7.2.

Szyfr strumieniowy dzieli wiadomość M na znaki lub bity $m_1, m_2, \ldots,$ a następnie szyfruje każdy element m_i za pomocą elementu klucza k_i, należącego do strumienia klucza $K = k_1, k_2, \ldots$

$$E_K(M) = E_{k1}(m_1)\,E_{k2}(m_2)\ldots \tag{3.7.1}$$

Szyfr strumieniowy jest okresowy, jeśli powtarza się strumień klucza. Do algorytmów strumieniowych okresowych należą szyfry podstawieniowe, szyfry Vigenère'a i szyfry realizowane w maszynach rotorowych. Szyfr Vernama jest szyfrem strumieniowym nieokresowym.



Rys. 3.7.1. Strumieniowy szyfr synchroniczny

Klucz w urządzeniu realizującym szyfr strumieniowy może być umieszczony w pamięci lub generowany na bieżąco. Pamiętanie klucza w przypadku długich strumieni kluczy jest niepraktyczne. Implementację synchronicznego szyfru strumieniowego z generatorem klucza pokazano na rys. 3.7.1.

Gdy urządzenie przetwarza ciągi bitów, wtedy szyfrowanie i deszyfrowanie odbywa się zgodnie z zasadami rachowania w ciele $GF(2)$, a szyfrator i deszyfrator jest

bramką Ex-OR. Zatem algorytm szyfrowania ma postać

$$c_i = E_{k_i}(m_i) = m_i + k_i, \tag{3.7.2}$$

gdzie każdy element jest bitem, a dodawanie jest dodawaniem modulo dwa.

Do obliczenia elementu tekstu jawnego z kryptogramu używa się następującego algorytmu

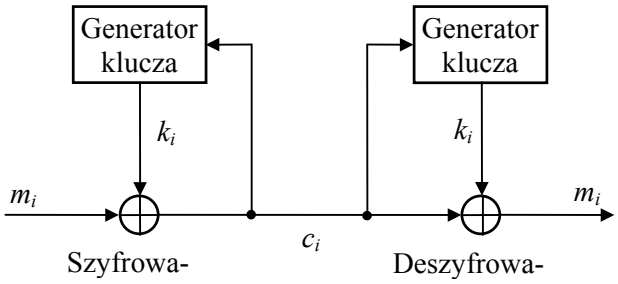
$$D_{k_i}(c_i) = c_i + k_i = (m_i + k_i) + k_i = m_i. \tag{3.7.3}$$

Szyfry strumieniowe należą do systemów z kluczem tajnym, gdzie bezpieczeństwo systemu zależy od klucza. Algorytm generowania klucza musi być algorytmem deterministycznym, aby klucz mógł być łatwo odtworzony po stronie odbiorczej. Klucze najczęściej są ciągami okresowymi, ale najlepszym rozwiązaniem jest klucz jednorazowy. Generowanie kluczy opisano w rozdziale 3.5.

Generator klucza synchronizuje się za pomocą impulsów synchronizujących *IS*. System kryptograficzny działa poprawnie, jeśli oba generatory, po stronie nadawczej i odbiorczej, pracują synchronicznie. Gdy generatory stracą synchronizm, musi on być przywrócony. Sygnały synchronizacji blokowej w systemie zapewniają protokoły komunikacyjne, a sygnały synchronizacji bitowej – urządzenia transmisyjne modemy.

System z szyfrem strumieniowym nie powoduje propagacji błędów transmisyjnych. Błąd transmisji jednego znaku nie wpływa na następne znaki.

Odmianą szyfrów strumieniowych są *samosynchronizujące się szyfry strumieniowe* (self-synchronous stream cipher). Systemy te, do generowania kluczy, powszechnie wykorzystują sprzężenie zwrotne kryptogramu (cipher-feedback mode). Schemat takiego układu pokazano na rys. 3.7.2.



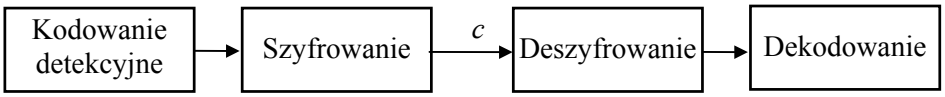
Rys. 3.7.2. Samosynchronizujcy się szyfr strumieniowy

Generator klucza deszyfrującego synchronizuje się automatycznie po odebraniu n bitów kryptogramu. W systemach samosynchronizujących się każda wiadomość zaczyna się nagłówkiem losowym o długości n bitów, który służy do synchronizacji systemu. Deszyfrowanie tego ciągu będzie niepoprawne.

Do generowania kluczy w szyfrach strumieniowych mogą być stosowane kryptograficzne algorytmy blokowe. Algorytmy blokowe zapewniają systemowi kryptograficznemu dobrą ochronę kryptograficzną.

Złożoność kryptograficzna samosynchronizującego się szyfru strumieniowego zależy od algorytmu generującego klucz. Kryptoanaliza tych szyfrów jest trudna, ponieważ różne fragmenty kryptogramu szyfruje się innym kluczem. Ujemną stroną takich szyfrów jest propagacja błędów transmisyjnych. Każdy błąd transmisyjny może spowodować *n* błędów w tekście zdeszyfrowanym.

Szyfry strumieniowe można stosować z kodami detekcyjnymi i korekcyjnymi. Zastosowanie kodów detekcyjnych umożliwia zachowanie autentyczności. Stosujemy je w sposób pokazany na rys. 3.7.3. Zmiana tekstu przed szyfrowaniem zostanie wykryta przez dekoder.



Rys. 3.7.3. Szyfrowanie z wykorzystaniem kodu detekcyjnego

W szyfrach blokowych i samosynchronizujących się szyfrach strumieniowych zachodzi propagacja błędów i kod detekcyjny może być stosowany. W szyfrach strumieniowych nie ma propagacji błędów, a liniowy kod detekcyjny daje się łatwo rozszyfrować i dlatego w zastosowaniach kryptograficznych należałoby korzystać z kodów nieliniowych.

Kody korygujące błędy trzeba stosować po zaszyfrowaniu wiadomości. Zadaniem kodów korekcyjnych jest eliminacja błędów transmisyjnych.

3.7.2. Szyfry blokowe

Szyfr blokowy dzieli wiadomość M na bloki M_1, M_2, ... i każdy z nich szyfruje, używając tego samego klucza K.

$$E_K(M) = E_K(M_1)E_K(M_2)\ldots \tag{3.7.4}$$

Do algorytmów blokowych należą: szyfry przestawieniowe, Lucifer, DES i szyfry z kluczem jawnym. W szyfrach blokowych następuje propagacja błędów transmisyjnych o zakresie jednego bloku, ale błędy te nie mają wpływu na inne bloki.

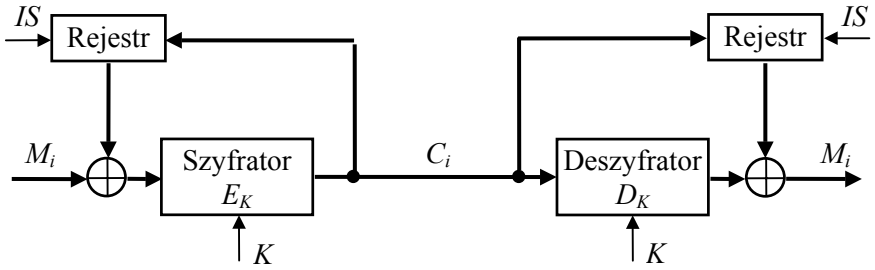
Najprostszą metodą szyfrowania blokowego jest przekształcenie każdego bloku tekstu jawnego w blok szyfrogramu niezależnie od innych bloków. Taka metoda pracy nazywa się *trybem elektronicznej książki kodowej* (electronic codebook – ECB).

Tryb ECB jest najłatwiejszy do implementacji. Brak związków między blokami ułatwia szyfrowanie w bazach danych i transmisji pracującej w trybie datagramowym. W trybie datagramowym pakiety wiadomości są przekazywane niezależnie, a stacja odbiorcza układa je w odpowiedniej kolejności.

Słabą stroną ECB jest możliwość zgromadzenia przez kryptoanalityka takiej ilości materiałów, która umożliwia złamanie szyfru, gdy nie znamy klucza. W tym celu kryptoanalitycy wykorzystują powtarzające się fragmenty dokumentów, np. w poczcie elektronicznej. Szyfry blokowe można uodpornić na kryptoanalizę, stosując technikę

wiązania blokowego.

W *trybie wiązanie bloków* wykorzystano technikę sprzężenia zwrotnego, w której blok poprzedni służy do modyfikacji szyfrowania bloku następnego. W ten sposób każdy blok szyfrogramu zależy nie tylko od bieżącego bloku tekstu jawnego, ale również od wszystkich poprzednich bloków. Istnieją różne tryby wiązania bloków [3.9], lecz najczęściej korzysta się z wiązania bloków zaszyfrowanych (cipher block chaining – CBC) i jego modyfikacji.



Rys. 3.7.4. Szyfr blokowy z wiązaniem bloków zaszyfrowanych

Schemat szyfrowania z wiązaniem bloków zaszyfrowanych pokazano na rys. 3.7.4. Grube linie oznaczają połączenia dla bloków wiadomości. W układzie sprzężenia zwrotnego znajdują się rejestry przesuwne umożliwiające zapamiętanie bloku kryptogramu. Kolejny blok wiadomości jest sumowany modulo dwa z poprzednim blokiem kryptogramu, a następnie szyfrowany za pomocą klucza K w szyfratorze. Szyfrowanie wyraża zależność

$$C_i = E_K\left(M_i + C_{i-1}\right). \tag{3.7.5}$$

Deszyfrowanie wykonuje się obliczając

$$D_K\left(C_i\right) + C_{i-1} = D_K\left(E_K\left(M_i + C_{i-1}\right)\right) + C_{i-1} = \left(M_i + C_{i-1}\right) + C_{i-1} = M_i. \tag{3.7.6}$$

W celu rozpoczęcia pracy systemu stosuje się wektor początkowy, który jest zwykle ciągiem losowym.

Ponieważ każdy zaszyfrowany blok ma powiązanie z blokiem poprzednim, przeto jeden błąd transmisyjny wpływa najwyżej na dwa bloki wyjściowe. System ten jest wrażliwy na błędy synchronizacji, gdyż przesunięcie szyfrogramu nawet o jeden bit powoduje błędną pracę systemu. Jednocześnie każdy zaszyfrowany blok zależy od wszystkich poprzednich bloków zaszyfrowanych, więc cechy statystyczne tekstu jawnego rozprzestrzeniają się na cały zaszyfrowany tekst, co znacznie utrudnia kryptoanalizę.

Tryb wiązania bloków zaszyfrowanych opracowano w IBM i zastosowano w Systemie Ochrony Informacji (Information Protection System – IPS). System ten pracuje z algorytmem DES. Tryb wiązanie bloków zaszyfrowanych jest najlepszy do szyfrowania plików. Do szyfrowania baz danych wygodniejszy jest tryb bezpośredniego szyfrowania bloków. Dla baz danych opracowano też inne rodzaje szyfrów blokowych jak np. szyfr blokowy z podkluczami [3.2].

W szyfrach blokowych stosuje się również techniki szyfrowania wielokrotnego. Szyfrowanie wielokrotne ma miejsce wtedy, kiedy ten sam blok tekstu jawnego jest szyfrowany wiele razy. W literaturze opisano metody szyfrowania dwukrotnego lub trzykrotnego z różnymi kluczami [3.9].

3.7.3. Uwierzytelnianie użytkownika

Uwierzytelnianie jest metodą sprawdzania tożsamości użytkownika lub systemu. Do uwierzytelnienia tożsamości użytkowników komputerów używa się haseł, kart magnetycznych lub innych metod dialogowych. Jeśli użytkownik może podać odpowiednie informacje, to zostaje zaakceptowany przez komputer. Hasło jest ciągiem znaków wprowadzanych przez użytkownika i sprawdzanych przez komputer. Hasło powinno być pseudolosowym ciągiem znaków.

Plik z hasłami należy chronić za pomocą szyfrowania funkcjami jednokierunkowymi. Plik taki nie powinien być przez nikogo zdeszyfrowany. Każde wejście do systemu komputerowego wymaga podania identyfikatora i hasła. System wyznacza zaszyfrowaną postać hasła i porównuje ją z zaszyfrowanym hasłem przechowywanym w pliku hasłowym. Na bezpieczeństwo hasła ma wpływ jego długość i czas użytkowania.

Dobrym przykładem uwierzytelniania jest metoda stosowana w systemie operacyjnym Unix. W systemie SCO Unix przewidziano cztery poziomy bezpieczeństwa: niski, tradycyjny, podwyższony i wysoki. System umożliwia wprowadzenie hasła przez użytkownika lub wygenerowanie go przez generator systemowy. Użytkownicy zwykle wybierają hasła krótkie i łatwe do zapamiętania. W systemie Unix do hasła wprowadzonego przez użytkownika przed zaszyfrowaniem dodaje się 12-bitową liczbę losową. Liczba ta utrudnia znalezienie hasła metodą generowania haseł losowych. W zależności od poziomu bezpieczeństwa hasło ma określoną długość minimalną i maksymalną. Parametry te mogą być zmienione przez administratora systemu. Gdy poziom bezpieczeństwa jest podwyższony lub wysoki, okres stosowania hasła i okres jego życia są ograniczone.

Obecne tendencje, zmierzające do tworzenia ogólnodostępnych sieci komputerowych i rozproszonego przetwarzania, wymagają rozbudowy algorytmów sterowania dostępem do zasobów sieci. Zapewniają one legalność dostępu do urządzeń, danych i programów. W wielu mechanizmach sterowania dostępem stosuje się koncepcję własności (ownership) polegającą na tym, że właściciele obiektów decydują o prawach dostępu do obiektu.

Skuteczność sterowania dostępem wymaga spełnienia dwóch warunków:

- poprawnej identyfikacji użytkownika,
- informacje opisujące prawa dostępu muszą być chronione przed nielegalną modyfikacją.

Do uwierzytelniania użytkowników w sieciach z zestawem protokołów TCP/IP i systemem Unix opracowano protokół o nazwie Kerberos [3.9]. Protokoły TPC/IP (Transmission Control Protocol/Internet Protocol) są standardem komunikacyjnym

wiążącym sieć Internet. Kerberos jest systemem symetrycznym, w którym wykorzystano algorytm DES.

Kerberos zainstalowany w sieci spełnia rolę zaufanego arbitra. Dla systemu jednostkami sieciowymi są klient i serwer. Podstawą identyfikacji jednostki jest znajomość klucza tajnego. System utrzymuje bazę klientów i ich tajnych kluczy. W systemie rejestruje się tajne klucze usług sieciowych wymagających uwierzytelniania i klientów korzystających z tych usług. Klient jest identyfikowany na podstawie hasła. Dostęp do serwera i usług klienci uzyskują od serwera uwierzytelniającego.

W protokołach dostępu do systemów operacyjnych i baz danych często jest stosowana macierz dostępu, która stanowi podstawę opisu systemu ich ochrony [3.2]. Wiersze macierzy dostępu odpowiadają podmiotom (użytkownikom), a kolumny obiektom (plikom, relacjom rekordom lub polom w rekordach). Elementy macierzy dostępu stanowią reguły decyzyjne określające warunki, jakie użytkownik musi spełnić, aby uzyskać dostęp do obiektu, i jakie operacje może wykonać.

3.7.4. Podpisy cyfrowe

Dokumenty przesyłane w systemach teleinformatycznych powinny posiadać znaczniki, które określają autentyczność nadawcy i odpowiadają podpisom odręcznym. Podpis dokumentu odzwierciedla indywidualną cechę autora dokumentu i stanowi dowód jego autentyczności. Rolę podpisów odręcznych w dokumentach przesyłanych w sieciach komputerowych spełnia podpis cyfrowy. Podpis cyfrowy jest sekwencją bitów dołączonych do dokumentu. Proces sprawdzania podpisu nazywa się uwierzytelnianiem. Podpis cyfrowy powinien spełniać następujące wymagania:

- Odbiorca dokumentu musi mieć możliwość zweryfikowania autentyczności podpisu.
- Nikt, łącznie z odbiorcą dokumentu, nie powinien mieć możliwości podrobienia podpisu.
- Powinna istnieć możliwość rozstrzygnięcia sporu dotyczącego autentyczności podpisu, gdyby nadawca wyparł się autorstwa podpisanego dokumentu.

Systemy kryptograficzne z kluczem tajnym zapewniają ochronę autentyczności danych, ale nie mają mechanizmów potwierdzenia autentyczności nadawcy. W systemach tych nadawca i odbiorca dysponują takimi samymi kluczami, a więc istnieje możliwość sfałszowania przez odbiorcę podpisu nadawcy bez możliwości wykrycia fałszerstwa. Stosowanie podpisów cyfrowych w systemach z kluczem tajnym wymaga wprowadzenia bezstronnego pośrednika uczestniczącego w wymianie danych między dwoma użytkownikami. Każdy z nich ma wtedy swój własny klucz tajny, który udostępnia tylko pośrednikowi.

W systemach z kluczem jawnym można stosować podpisy cyfrowe z jednoczesną ochroną autentyczności. Schemat systemu zapewniającego poufność i autentyczność danych pokazano na rys. 3.2.3. Jako podpis nadawcy służy przekształcenie D_{a2}. Odbiorca wiadomości może stwierdzić autentyczność podpisu i danych za pomocą prze-

kształcenia E_{a1}. Do podpisów cyfrowych nadają się bezpośrednio niektóre algorytmyz kluczem jawnym jak np. algorytm ElGamala opisany w p. 3.6.3.

W USA istnieje standard podpisów cyfrowych DSS (Digital Signature Standard), w którym stosuje się algorytm podpisów cyfrowych DSA (Digital Signature Algorithm). Algorytm DSA jest algorytmem z kluczem jawnym i odmianą algorytmu ElGamala.

W rzeczywistych zastosowaniach algorytmy z kluczem jawnym są często niewystarczające do szyfrowania długich dokumentów. Gdy algorytm jest zorientowany na podpisywanie bloków wiadomości, wówczas podpisywanie wiadomości dłuższej niż blok, to jest blok po bloku, zabiera dużo czasu i graniczy z ryzykiem. W takich przypadkach protokóły podpisów cyfrowych są często implementowane łącznie z *jednokierunkowymi funkcjami skrótu*, zwanymi również funkcjami haszowymi.

Większość jednokierunkowych funkcji skrótu buduje się na bazie funkcji jednokierunkowych, które działają na dwa bloki wejściowe. W ogólnym przypadku argumentem funkcji jest blok tekstu i skrót poprzedniego bloku tekstu. W ten sposób skrót ostatniego bloku staje się skrótem całej wiadomości. Zamiast podpisywać bloki dokumentu, podpisuje się wtedy cały dokument w postaci skompresowanej.

3.7.5. Kryptografia w sieciach komputerowych

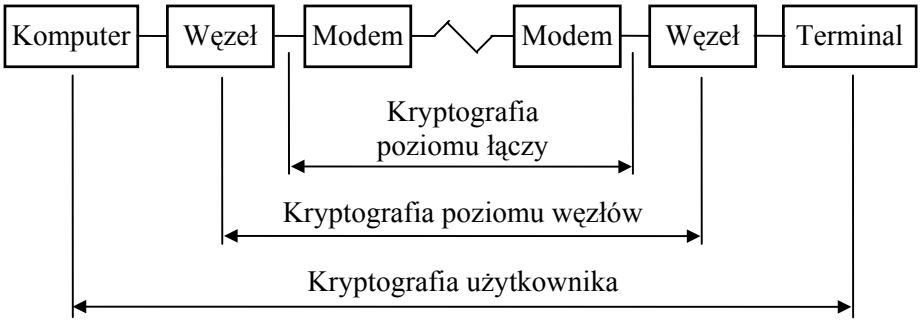
Międzynarodowy Komitet Normalizacyjny (International Standards Organization – ISO) przyjął wzorcową architekturę sieci komputerowej (Open System Interconnection – OSI) zawierającą siedem poziomów: fizyczny, łączy transmisji danych, sieciowy, transportowy, sesji, prezentacji i aplikacji. W sieci komputerowej istnieje wiele możliwości wyboru punktów szyfrowania. Jednak najczęściej kryptografię stosuje się na poziomach:

- łączy transmisji danych (link encryption),
- sieciowym (node encryption),
- aplikacji (end-to-end encryption).

Ilustruje to rys. 3.7.5.

W kryptografii poziomu łączy urządzenia kryptograficzne są umieszczone między węzłem a modemem, a szyfrowanie jest logicznie niezależne od systemu komunikacyjnego. System kryptograficzny używa tylko jednego klucza. Kryptografia poziomu węzłów komunikacyjnych jest podobna do kryptografii na poziomie łącza. Różnica polega na tym, że w kryptografii poziomu węzłów urządzenia kryptograficzne są urządzeniami dołączonymi do węzłów, a każde łącze chroni inny klucz.

Najlepszą ochronę daje szyfrowanie na poziomie aplikacji, gdyż wiadomość jest deszyfrowana dopiero w urządzeniu końcowym. Jednak system ten okazał się wrażliwy na próbę złamania szyfru w wyniku analizy przepływu danych w sieci. Kryptografia użytkownika wymaga specjalnego protokołu do obsługi kluczy kryptograficznych. Protokół zarządzania kluczami firmy IBM opisano w p. 3.5.4. W systemach wymagających wysokiego stopnia utajnienia można stosować kilka metod jednocześnie.



Rys. 3.7.5. Kryptografia w sieci komputerowej

W szyfrowaniu na poziomie łączy adresy węzłów docelowych, podobnie jak i dane, mogą być również szyfrowane. Nie jest to możliwe w kryptografii użytkownika, gdyż węzły komunikacyjne muszą znać adresy docelowe, aby wyznaczyć drogi transmisji. Z drugiej strony na podstawie tych adresów można dowiedzieć się, czy została dokonana jakaś transakcja.

Szyfrowanie na poziomie łączy i węzłów można zrealizować urządzeniowo za pomocą szyfrów strumieniowych lub blokowych. Do szyfrowania na poziomie użytkownika najodpowiedniejsze są szyfry blokowe z algorytmami z kluczami jawnymi. Implementacja tych szyfrów jest najczęściej programowa.

Ochrona sieci wymaga kombinacji szyfrowania i innych mechanizmów ochrony. Szyfrowanie chroni dane przesyłane w sieci, ale przed dostępem do procedur zarządzania kluczami chronią mechanizmy sterowania dostępem. Podobnie chroni się dostęp do zasobów sieci i urządzeń.

3.7.6. Szyfrowanie plików

Szyfrowanie danych do przechowywania jest innym problemem niż szyfrowanie danych do transmisji. Przede wszystkim wymaga się długotrwałego i bezpiecznego przechowywania kluczy. Zarządzanie kluczami jest bardziej skomplikowane, ponieważ różni użytkownicy potrzebują uzyskać dostęp do różnych plików lub ich części.

Programy do szyfrowania plików są popularne i dostępne w niektórych systemach operacyjnych. W systemie operacyjnym Unix dostępne jest polecenie CRYPT oparte na algorytmie Enigmy. Dystrybucję tego polecenia kontrolują agendy rządowe Stanów Zjednoczonych i na ogół nie jest ono dostępne poza terytorium USA.

Przykładem programu służącego do szyfrowania i deszyfrowania plików oraz tworzenia i obsługi dysków logicznych z szyfrowanym dostępem jest program DISCREET w pakiecie Norton Utilities. W wersji 7.0, podobnie jak w starszych wersjach, są przeznaczone do tego celu dwa pliki: discreet.exe i discreet.sys. Dostęp do zaszyfrowanych danych uzyskuje się po podaniu hasła. Aby można było korzystać z tego programu, należy zainstalować program sterujący w pliku config.sys za pomocą instrukcji device=ścieżka\descreet.sys. Program umożliwia wybranie jednego z dwóch algoryt-

mów: wolniejszego DES albo szybszego Fast.

Szyfrowanie plików wymaga utworzenia bezpiecznego schematu zarządzania kluczami. Klucze i pliki nieszyfrowane powinny być wymazane po zaszyfrowaniu. Zasady zarządzania kluczami do ochrony plików w systemach z algorytmem DES opisano w p. 3.5.4.

Usuwając klucze i pliki w komputerze, należy zachować zasady bezpieczeństwa. Komputer realizuje własne metody zarządzania pamięcią i ciągle przerzuca dane do i z pamięci operacyjnej. W tej sytuacji nie ma żadnego pewnego sposobu bezpiecznego skasowania klucza czy pliku w komputerze. Jeśli klucz został zapisany na dysku, to powinien on być nadpisany wielokrotnie, aby wymazanie było skuteczne. Do tego celu stosuje się specjalne programy. Większość programów ma zaimplementowany standard DoD (Data-on-Data) z trzykrotnym nadpisywaniem: najpierw samymi jedynkami, następnie zerami i w końcu ciągiem 1-0.

Literatura

1. Teoria ciał skończonych

[1.1] BORATYŃSKI M., *Elementy teorii Galois*, Wydawnictwo „Alfa", Warszawa, 1985.

[1.2] LIDL R., NIEDERREITER H., *Finite Fields*, Addison-Wesley Publishing Company, London, 1983.
Tłumaczenie rosyjskie: *Конечные поля*, Издательство МИР, Москва, 1988.

[1.3] MCWILIAMS F. J., SLOANE N. J. A., *Pseudo-random sequences and arrays*, Proc. IEEE, Vol. 64, No 12, 1976.

2. Kody korekcyjne

[2.1] DRÓŻDŻ J., *Podstawy kodowania nadmiarowego*, Wydawnictwo Politechniki Warszawskiej, Warszawa, 1978.

[2.2] KOŚCIELNY C., *Programowa realizacja działań w ciałach skończonych do zastosowań w technice kodowania korekcyjnego i kryptografii*, Wydawnictwo Politechniki Wrocławskiej, Wrocław, 1983.

[2.3] KOŚCIELNY C., *Constructing a Better Cyclic Code than Cyclic Reed-Solomon Code*, IEEE Transactions on Information Theory, Vol. 41, No. 4, July 1995.

[2.4] PETERSON W. W., WELDON E. J., *Error-Correcting Codes*, The MIT Press, Cambridge, Massachusetts and London, 1972,
Tłumaczenie rosyjskie: *Коды исправляющие ошибки*, Издательство МИР, Москва, 1976.

[2.5] SHU LIN, *An Introduction to Error-Correcting Codes*, Prentice-Hall, Inc., London, 1970.

[2.6] SZWAJA Z., *Realizacja binarnych kodów Bose-Chaudhuri-Hocquenghema*, Wydawnictwo Politechniki Poznańskiej, Poznań, 1968.

3. Kryptografia

[3.1] *Data Encryption Standard*, Federal Information Processing Standard Publication 46, National Bureau of Standards, U. S. Department of commerce, Washington, DC, January 1977.

[3.2] DENNING D. E. R., *Kryptografia i ochrona danych*, WNT, Warszawa, 1992.

[3.3] DIFFIE W., HELLMAN M. E., *New Directions in Cryptography*, IEEE Transactions on Information Theory, Vol. IT–22, No 6, Nov. 1976.

[3.4] EHRSAM W. F., MATIA S. M., MEYER C. H., TUCHMAN W. L., *A cryptographic key management scheme for implementing the data encryption standard,* IBM System Journal, Vol. 17(2), 1978.

[3.5] HELLMAN M. E., *The Mathematics of Public-Key Cryptography*, Scientific American, Vol. 241, No 2, 1979.

[3.6] KOBLITZ N., *Wykład z teorii liczb i kryptografii*, WNT, Warszawa, 1995.

[3.7] KOBLITZ N., *Algebraiczne aspekty kryptografii*, WNT, Warszawa, 2000.
[3.8] MATIA S. M., MEYER C. H., *Generation, distribution, and instalation of cryptographic keys,* IBM System Journal, Vol. 17(2), 1978.
[3.9] SCHNEIER B., *Kryptografia dla praktyków*, WNT, Warszawa, 1995.
[3.10] SCHNEIER B., *Ochrona poczty elektronicznej*, WNT, Warszawa, 1996.
[3.11] SHANNON C. E., *Communication Theory of Secrecy Systems*, BSTJ 28, 1949.
[3.12] STOKŁOSA J., *Kryptograficzna ochrona danych w systemach komputerowych*, Nakom, Poznań, 1994.

Indeks