Kryptografia

Wykład z podstaw klasycznej kryptografii z elementami kryptografii kwantowej

dla studentów IV roku

Ryszard Tanaś

Zakład Optyki Nieliniowej, Instytut Fizyki UAM

tanas@kielich.amu.edu.pl

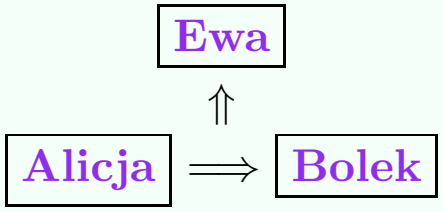
Serdecznie witam!

Literatura

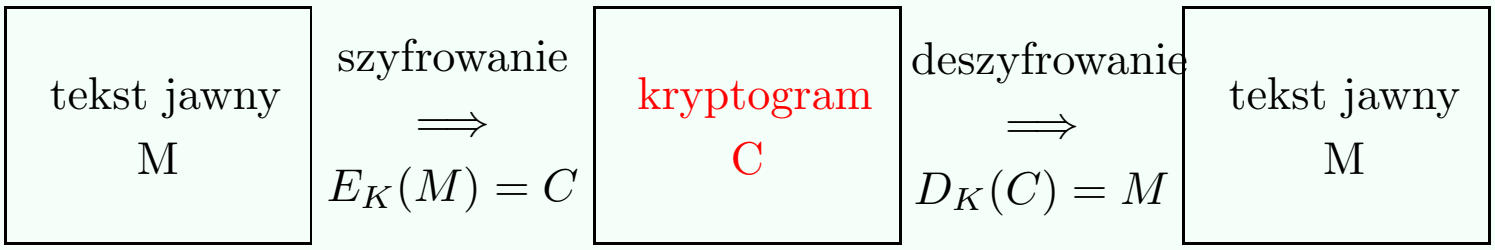
- M. Kutyłowski i W. B. Strothmann *Kryptografia: Teoria i praktyka zabezpieczania systemów komputerowych*, Wyd. READ ME, Warszawa, 1999, drugie wydanie dostępne w księgarniach
- B. Schneier *Kryptografia dla praktyków*, WNT, Warszawa, 2002, wydanie drugie
- R. Wobst, *Kryptologia. Budowa i łamanie zabezpieczeń*, RM, Warszawa, 2002
- A. J. Menezes, P. C. van Oorschot, S. A. Vanstone *Kryptografia stosowana*, WNT, Warszawa, 2005; *Handbook of Applied Cryptography*, CRC Press, 1997, New York, dostępna w Internecie
- W. Stein *An Explicit Approach to Elementary Number Theory* `http://modular.fas.harvard.edu/edu/Fall2001/124/lectures/`
- S. J. Lomonaco *A quick glance at quantum cryptography*, LANL quant-ph archive, quant-ph/9811056, 1998
- S. J. Lomonaco *A talk on quantum cryptography or how Alice outwits Eve*, LANL quantum-ph archive, quant-ph/0102016, 2001
- N. Gisin, G. Ribordy, W. Titel, H. Zbinden *Quantum cryptography*, LANL quant-ph archive, quant-ph/0101098, 2001

Terminologia

- **Kryptografia** — dziedzina wiedzy zajmująca się zabezpieczaniem informacji (szyfrowanie)
- **Kryptoanaliza** — łamanie szyfrów
- **Kryptologia** — dział matematyki, który zajmuje się podstawami metod kryptograficznych (kryptografia + kryptoanaliza)
- **Główne postacie**
  - ⋆ **Alicja — nadawca informacji**
  - ⋆ **Bolek — odbiorca informacji**
  - ⋆ **Ewa — podsłuchująca kanał przesyłowy i usiłująca przechwycić informację przeznaczoną dla Bolka**



Szyfrowanie i deszyfrowanie



Algorytmy

- **symetryczne** — klucz do szyfrowania i deszyfrowania jest ten sam (**klucz tajny**) — DES, IDEA, AES
- **asymetryczne** — klucze do szyfrowania i deszyfrowania są różne (**klucz jawny albo publiczny** — RSA, ElGamal)

Przykład kryptogramu

- tekst jawny

  Wykład z podstaw klasycznej kryptografii z elementami kryptografii kwantowej

- kryptogram (GnuPG)

hQEOA+npwcy1l0+VEAP+IrpTozmtpWBINXV5koW5sBC86EAelZTrEXrzUHohenPo
ohzkgIoBH17Rvu46hZUsHjeHyH74RI1Lv0klHbtBOLiCLvZfdtBWFFtzr4j4kDt7
n7kGMrJCxwOKuZIVCdMrRS9jvcBgFydYIeq/jkA3VvPGU4nT3AEyqiZ+xkrPRvsE
AJ59+4YDc1sbccJdu6nyRMJ2rcYH+SoS+BDgUmkopkG2KCjnQHArUWGq9N1v3ULH
dRfKwl4kgOK2EQGTFaQxjGXqyK41MS5noOZhZ8nHgJ4N9vE/TH/CaTiWgLQyXoKt
4J4xOJ5wx6rjNIK5MRl37XxWr3D8xDwWBGtKFGLllcV/0ogBymNlqBWZB6qi/xZo
cLdPWR94WmIvpkxWsR5HZhU06K6D7l/KgSarosSDwpOtT6c/21epCZvuvrfnq8pm
lpTXqVuHVsZNGCp599pJCkgLTxdQDyV0xjD8feVEtX2pfHxdWMORMdEG2QGfWSCa
z0hvf2t7B+7lFQsK+TPi3+YQMaoXK+XmAyPz =vRaX

Podstawowe zastosowania

- ochrona danych
  - ⋆ dane na dyskach
  - ⋆ przesyłanie danych poprzez linie narażone na podsłuch
- uwierzytelnianie dokumentów i osób
- ochrona prywatności korespondencji elektronicznej
- elektroniczny notariusz
- podpis cyfrowy
- pieniądze cyfrowe
- wybory elektroniczne

Jak to działa: algorytm symetryczny

1. Alicja i Bolek uzgadniają algorytm i klucz jakich będą używać
2. Alicja szyfruje tekst używając uzgodnionego algorytmu i klucza otrzymując kryptogram
3. Alicja przesyła kryptogram do Bolka
4. Bolek deszyfruje kryptogram używając tego samego algorytmu i klucza otrzymując tekst jawny

- **Problemy:**
  - ⋆ klucz musi być przekazywany w sposób tajny
  - ⋆ jeśli Ewa wejdzie w posiadanie klucza to może deszyszyfrować wszystko, a nawet podszyć się pod Alicję
  - ⋆ jeśli każda para korespondentów w sieci dysponuje własnym kluczem to liczba kluczy szybko rośnie dla kogoś kto utrzymuje kontakt z wieloma osobami

Jak to działa: algorytm asymetryczny

1. Alicja i Bolek uzgadniają kryptosystem z kluczem publicznym, którego będą używać
2. Bolek przesyła Alicji swój klucz publiczny
3. Alicja szyfruje wiadomość kluczem publicznym Bolka i przesyła kryptogram do Bolka
4. Bolek deszyfruje kryptogram używając swojego klucza prywatnego

lub użytkownicy sieci uzgadniają kryptosystem i przesyłają swoje klucze publiczne do bazy na znanym serwerze i wtedy protokół wygląda jeszcze prościej

1. Alicja i Bolek pobierają klucze publiczne z serwera
2. Alicja szyfruje wiadomość kluczem publicznym Bolka i wysyła kryptogram do Bolka
3. Bolek deszyfruje wiadomość Alicji używając własnego klucza prywatnego

Kryptosystem hybrydowy

1. Bolek wysyła do Alicji swój klucz publiczny
2. Alicja generuje losowy klucz K dla obecnej sesji, szyfruje go kluczem publicznym Bolka i wysyła kryptogram klucza $E_B(K)$ do Bolka
3. Bolek deszyfruje kryptogram klucza używając swojego klucza prywatnego, $D_B(E_B(K))=K$, otrzymując klucz K dla obecnej sesji
4. oboje używają klucza K i symetrycznego algorytmu do szyfrowania i deszyfrowania informacji przesyłanych w czasie tej sesji

- **Uwagi:**
  - ⋆ algorytmy symetryczne są szybsze niż algorytmy asymetryczne, co ma znaczenie przy przesyłaniu dużej ilości danych
  - ⋆ jeśli Ewa zdobędzie klucz K, to może go użyć do deszyfrowania jedynie aktualnej sesji, potem już jest bezużyteczny

Podpis cyfrowy: kryptosystem z kluczem publicznym

1. Alicja szyfruje dokument używając swojego klucza prywatnego, podpisując w ten sposób dokument
2. Alicja przesyła tak podpisany dokument do Bolka
3. Bolek deszyfruje dokument używając klucza publicznego Alicji, weryfikując w ten sposób podpis Alicji

- **Uwagi:**
  - ⋆ podpis jest prawdziwy; Bolek weryfikuje go deszyfrując kryptogram kluczem publicznym Alicji
  - ⋆ podpis nie może być sfałszowany; tylko Alicja zna jej klucz prywatny
  - ⋆ podpis nie może być przeniesiony do innego dokumentu
  - ⋆ podpisany dokument nie może być zmieniony; zmieniony dokument nie da się rozszyfrować kluczem publicznym Alicji
  - ⋆ podpis jest niezaprzeczalny;

Jednokierunkowe funkcje hashujące (skrótu)

- dla każdego X łatwo jest obliczyć $H(X)$
- $H(X)$ ma taką samą długość dla wszystkich tekstów X
- dla zadanego Y znalezienie takiego X, że $H(X) = Y$ jest praktycznie niemożliwe
- dla zadanego X trudno znaleźć X' takie, że $H(X) = H(X')$

Elektroniczny notariusz

- dla danego dokumentu X obliczamy wartość H(X) i publikujemy lub deponujemy u notariusza wartość H(X)
- chcąc udowodnić prawdziwość dokumentu X przedstawiamy dokument, obliczamy H(X) i porównujemy z opublikowaną wcześniej wartością

Szyfr Cezara

- szyfr podstawieniowy monoalfabetyczny

A B C D E F G H I J K L M N O P R S T U V W X Y Z
D E F G H I J K L M N O P R S T U V W X Y Z A B C

tekst jawny $\Longrightarrow$ K R Y P T O G R A F I A
kryptogram $\Longrightarrow$ N U B T W S J U D I L D

Szyfr Vigenère'a

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
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
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
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	W	X	Y

klucz $\Longrightarrow$ S Z Y M P A N S S Z Y M
tekst $\Longrightarrow$ K R Y P T O G R A F I A
krypt. $\Longrightarrow$ C P W C I O U I S E G M

Operacja XOR i one-time pad — szyfr Vernama

$$\begin{aligned} 0 \oplus 0 &= 0 \\ 0 \oplus 1 &= 1 \\ 1 \oplus 0 &= 1 \\ 1 \oplus 1 &= 0 \end{aligned}$$

- tekst jawny jest ciągiem bitów M=$m_1, m_2, \ldots, m_n$
- wybieramy losowy ciąg bitów K=$k_1, k_2, \ldots, k_n$, który stanowi klucz
- szyfrowanie polega na wykonaniu operacji XOR bit po bicie;

  otrzymujemy w ten sposób losowy ciąg bitów stanowiących kryptogram C=$c_1, c_2, \ldots, c_n$, gdzie $c_i = m_i \oplus k_i$
- operacja ta jest odwracalna;

  ponieważ $a \oplus a = 0$ i $a \oplus b \oplus b = a$, zatem $c_i \oplus k_i = (m_i \oplus k_i) \oplus k_i = m_i$
- kryptogram jest losowym ciągiem n bitów

  Jeśli $k_i = m_i$ to $c_i = 0$, w przeciwnym wypadku $c_i = 1$; prawdopodobieństwo, że $c_i = 0$ jest równe $\frac{1}{2}$ niezależnie od wartości m_i, zatem i-ty bit kryptogramu jest losowy
- szyfr ten jest nie do złamania — **bezpieczeństwo doskonałe** — nie można uzyskać żadnej informacji o tekście jawnym bez znajomości klucza

  Ponieważ $c_i = m_i \oplus k_i$ implikuje $k_i = m_i \oplus c_i$, a kryptogram $c_1, c_2, \ldots, c_n$ odpowiada każdemu możliwemu tekstowi jawnemu z takim samym prawdopodobieństwem, to na podstawie samego kryptogramu nie wiemy nic o tekście jawnym

- **Problemy:**
  - ⋆ klucz musi być wcześniej uzgodniony przez Alicję i Bolka
  - ⋆ klucz musi być wybrany naprawdę losowo, co nie jest łatwe
  - ⋆ klucz musi być przechowywany w bezpieczny sposób
  - ⋆ klucz musi być co najmniej tak długi jak szyfrowany tekst
- **Przykład:**

tekst jawny	$\Longrightarrow$	S	Z	Y	F	R
binarnie	$\Longrightarrow$	01010011	01011010	01011001	01000110	01010010
klucz	$\Longrightarrow$	01110010	01010101	11011100	10110011	00101011
kryptogram	$\Longrightarrow$	00100001	00001111	10000101	11110101	01111001

DES — Data Encryption Standard

- w 1981 r. przyjęty w USA jako standard do celów cywilnych
- algorytm symetryczny
- szczegóły algorytmu zostały opublikowane (podejrzenia o *tylne drzwi*)
- szyfruje bloki 64 bitowe (8 liter ASCII z bitem parzystości)
- klucze są efektywnie 56 bitowe (64 bity minus 8 bitów parzystości); obecnie uważa się, że taka długość klucza jest zbyt mała

Etapy DES

- wejście — 64 bitowy blok
- permutacja początkowa
- blok zostaje podzielony na lewą i prawą połowę po 32 bity każda
- 16 rund identycznych operacji opisanych funkcją f, w czasie których dane prawej połowy są przekształcane z użyciem klucza
  - ⋆ w czasie każdej rundy bity klucza są przesuwane, a następnie 48 bitowy podklucz jest wybierany z 56 bitowego klucza
  - ⋆ prawa część danych jest rozszerzana do 48 bitów za pomocą permutacji rozszerzającej a następnie podlega operacji XOR z 48 bitami podklucza
  - ⋆ wynik wysyłany jest do 8 S-boksów, które produkują nowe 32 bity

⋆ otrzymane 32 bity są permutowane w P-boksie

- wynik tych 4 operacji stanowiących funkcję f podlega operacji XOR z lewą połową i staje się nową prawą połową
- stara prawa połowa staje się nową lewą połową, i tak 16 razy
- permutacja końcowa
- kryptogram

Jeśli L_i i R_i są lewą i prawą połową dla i-tej rundy, K_i jest podkluczem dla tej rundy, to tej rundy mamy

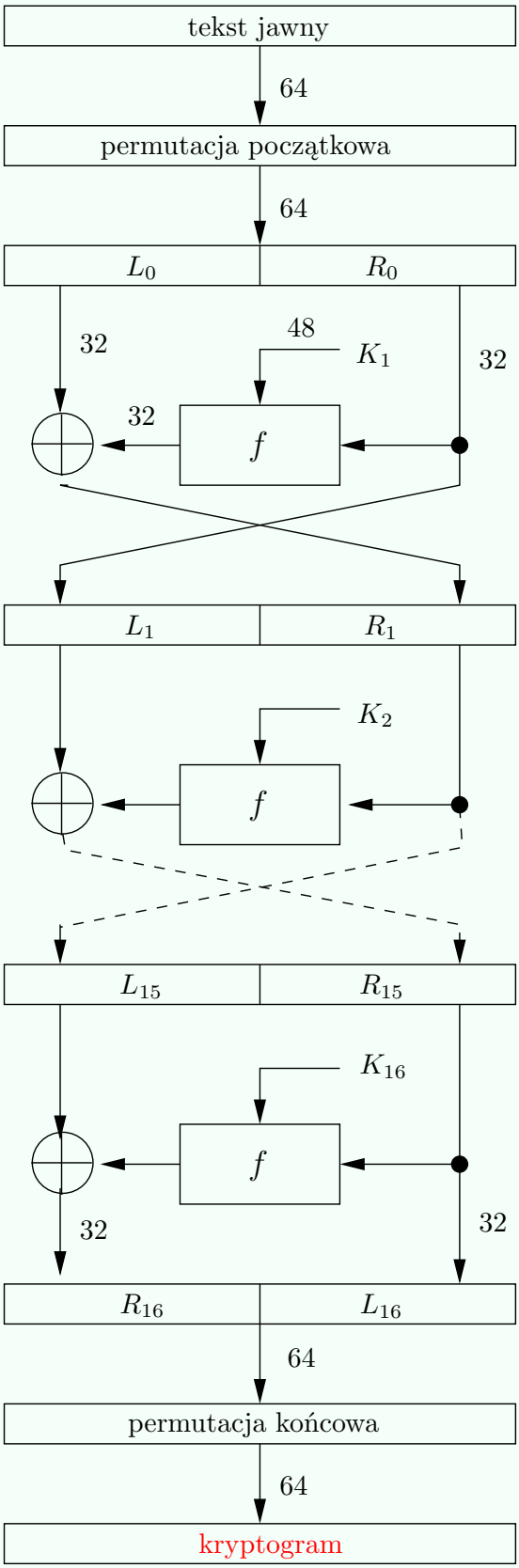
$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned}$$

- deszyfrowanie DES-em polega na przeprowadzeniu tych samych operacji co dla szyfrowania tylko podklucze występują w odwrotnej kolejności

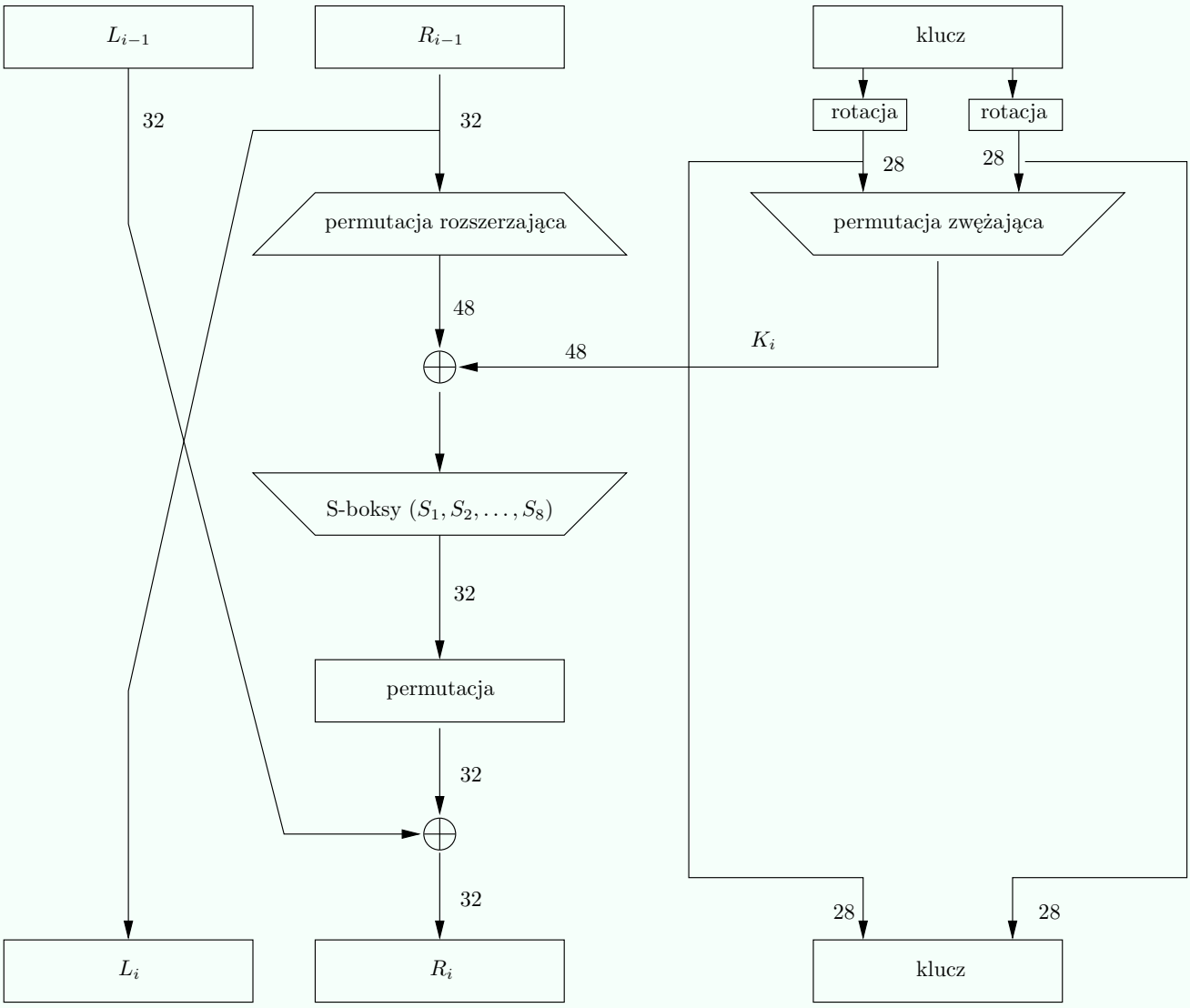
Ponieważ

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

to znając L_i, R_i oraz K_i możemy obliczyć L_{i-1} i R_{i-1}.



Rysunek 1: Schemat działania DES



Rysunek 2: Jedna runda DES

Elementy DES

- permutacje początkowa i końcowa nie mają znaczenia kryptograficznego (ułatwiają operowanie danymi w bajtach)

IP	58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
	62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
	57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
	61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7
IP^{-1}	40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
	38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
	36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
	34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

Tablica 1: Permutacja początkowa IP i końcowa IP^{-1} (tablice te czytamy od lewej do prawej, od góry w dół odczytując kolejne bity wyniku, a numery umieszczone na kolejnych pozycjach tabeli oznaczają numer bitu na wejściu, np. bit 58 wejścia jest pierwszym bitem wyjścia, itd.)

- generowanie podkluczy
  - ⋆ z 64 bitowego losowego klucza otrzymuje się 56 bitowy ignorując co ósmy bit i dokonując permutacji KP

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

Tablica 2: Permutacja klucza

  - ⋆ 56 bitowy klucz dzieli się na dwie połowy po 28 bitów
  - ⋆ połowy są przesuwane cyklicznie w lewo o 1 lub 2 bity w zależności od rundy wg reguły

runda	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
przesunięcie	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tablica 3: Przesunięcia połówek klucza

⋆ permutacja z kompresją CP (permutowany wybór) daje 48 bitów podklucza

CP	14	17	11	24	1	5	3	28	15	6	21	10
	23	19	12	4	26	8	16	7	27	20	13	2
	41	52	31	37	47	55	30	40	51	45	33	48
	44	49	39	56	34	53	46	42	50	36	29	32

Tablica 4: Permutacja zwężająca

- permutacja z rozszerzeniem rozszerza 32 bity R_i do 48 bitów

EP	32	1	2	3	4	5	4	5	6	7	8	9
	8	9	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	1

Tablica 5: Permutacja z rozszerzeniem

- S-boksy
  - ⋆ wynik operacji XOR na rozszerzonym R_i i K_i dzielony jest na 8 części po 6 bitów, z których każda przechodzi do oddzielnego S-boksu $(S_1, \ldots, S_8)$
  - ⋆ 6 bitów wchodzących do S-boksu przekształcanych jest w 4 bity wyjściowe w specjalny sposób pierwszy i ostatni bit 6 bitów wejściowych daje liczbę dwubitową od 0–3 oznaczającą wiersz S-boksu, zaś bity 2–5 dają liczbę 4-bitową od 0–15, która odpowiada kolumnie tabeli.

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tablica 6: S-boksy

Np. dla 110011 na wejściu, 1 i 6 bit dają $11_2 = 3_{10}$,

zaś bity 2–4 dają $1001_2 = 9_{10}$, na przecięciu wiersza 3 i kolumny 9 S_1 mamy liczbę $11_{10} = 1011_2$ (wiersze i kolumny liczymy od zera). W wyniku otrzymujemy 1011.

- wyniki z 8 S-boksów są łączone w 32 bitową liczbę
- na tych 32 bitach dokonuje się permutacji wg Tablicy 7 oraz operacji XOR z 32 bitami lewej połowy otrzymując nową prawą połowę, która przekazywana jest do następnej rundy

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Tablica 7: Pertmutacja (P-boks)

Trzykrotny DES

Rozszerzenie algorytmu DES, w którym stosuje się dwa klucze K_1 i K_2

- **Szyfrowanie**
  1. wiadomość szyfrowana jest kluczem K_1
  2. wynik kroku 1. deszyfrowany jest kluczem K_2
  3. wynik kroku 2. jest ponownie szyfrowany kluczem K_1
- **Deszyfrowanie**
  1. kryptogram deszyfrowany jest kluczem K_1
  2. wynik kroku 1. szyfrowany jest kluczem K_2
  3. wynik kroku 2. jest powtórnie deszyfrowany kluczem K_1

Tryby szyfrowania: szyfrowanie blokowe

- **ECB — Electronic Codebook** (Elektroniczna książka kodowa)
  Tekst jawny dzielony jest na bloki o długości 64 bity i każdy blok jest oddzielnie szyfrowany tym samym kluczem
  - ⋆ zaleta — utrata lub uszkodzenie pojedynczych bloków nie ma wpływu na możliwość deszyfrowania pozostałych; nadaje się do szyfrowania baz danych
  - ⋆ wada — możliwa jest modyfikacja kryptogramu bez znajomości klucza
- **CBC — Cipher Block Chaining** (Wiązanie bloków)
  Szyfrowanie kolejnego bloku zależy od wyniku szyfrowania poprzedniego bloku; taki sam blok tekstu jawnego jest w różnych miejscach szyfrowany inaczej — kolejny blok tekstu jawnego jest poddawany operacji XOR z kryptogramem poprzedniego bloku.
  Matematycznie wygląda to następująco:

$$\begin{aligned} C_1 &= E_K(M_1 \oplus I) \\ C_i &= E_K(M_i \oplus C_{i-1}) \end{aligned}$$

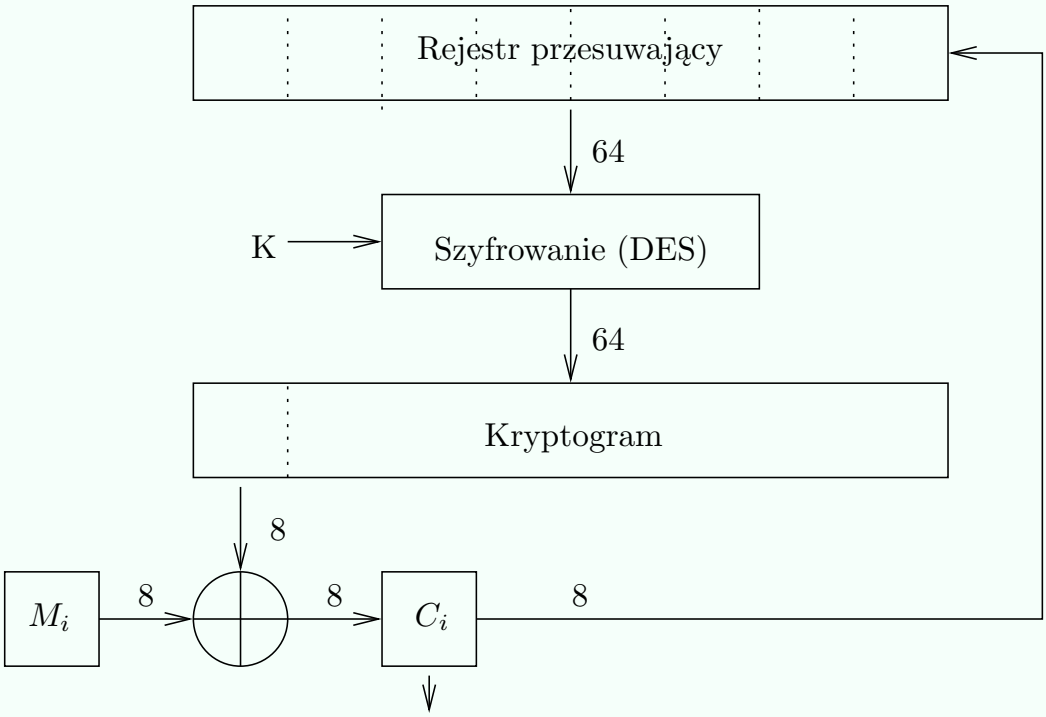
$$\begin{aligned} M_1 &= D_K(C_1 \oplus I) \\ M_i &= D_K(C_i) \oplus C_{i-1} \end{aligned}$$

  gdzie M_i jest i-tym blokiem wiadomości, C_i i-tym blokiem kryptogramu, zaś I jest losowym ciągiem bitów, który jest przesyłany bez szyfrowania
  - ⋆ zalety — takie same bloki tekstu jawnego mają różne kryptogramy; zmiana bitu (przekłamanie) wewnątrz jednego bloku prowadzi do zmiany tekstu po deszyfrowaniu tylko w danym bloku i następnym

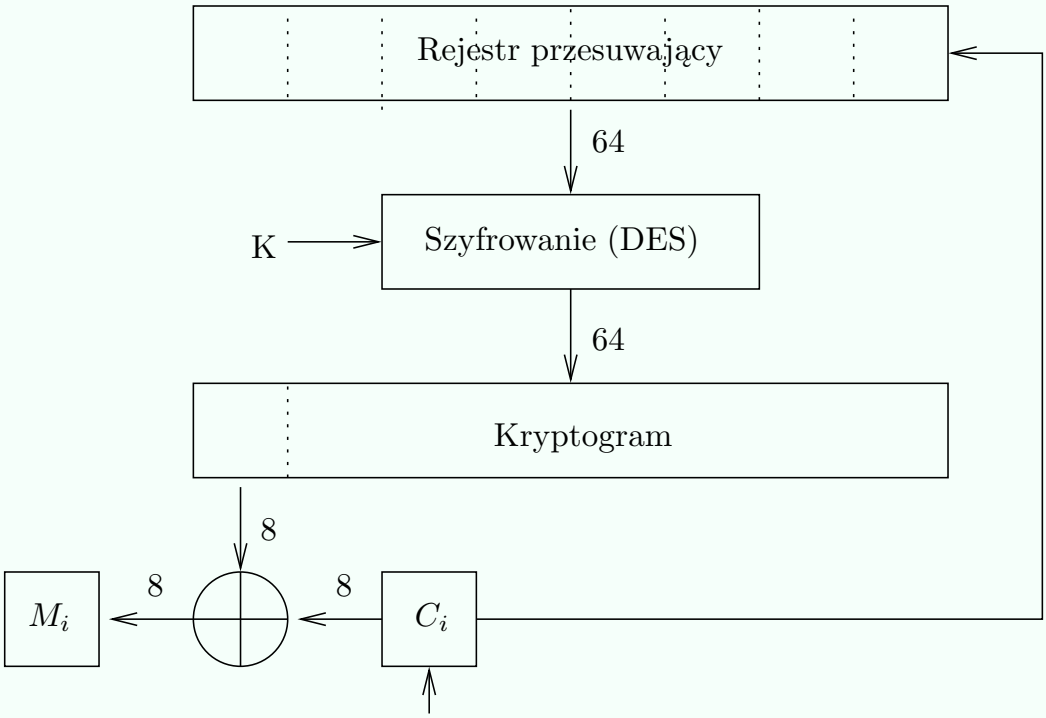
⋆ wady — nie można usunąć żadnego bloku z kryptogramu; nie nadaje się do szyfrowania baz danych; nieodporny na zakłócenia (dodatkowy bit lub utrata jednego bitu psują dalszy przekaz)

- **CFB — Cipher Feedback** (Szyfrowanie ze sprzężeniem zwrotnym)
  Szyfrowaniu podlegają jednostki mniejsze niż blok (64 bity), np. jeden znak ASCII (1 bajt = 8 bitów). Schemat działania przedstawiony jest na Rys. 3. Tryb ważny w zastosowaniach sieciowych, np. komunikacja pomiędzy klawiaturą i serwerem. Istotnym elementem CFB jest rejestr przesuwający.

  - Działanie CFB
    1. na początku rejestr przesuwający zawiera losowy ciąg 64 bitów
    2. zawartość rejestru przesuwającego jest szyfrowana za pomocą klucza K np. algorytmem DES
    3. 8 pierwszych bitów kryptogramu jest dodawane modulo 2 z 8 bitami reprezentującymi literę wiadomości (M_i) dając kryptogram C_i przesyłany do odbiorcy
    4. C_i jednocześnie przesyłane jest do rejestru przesuwającego zajmując ostatnie 8 bitów i przesuwając pozostałe bity o 8 pozycji w lewo; przesunięcie to nie jest cykliczne, tzn. pierwszych 8 bitów jest usuwanych
    5. przy deszyfrowaniu rola wejścia i wyjścia zostaje zamieniona



(a) Szyfrowanie



(b) Deszyfrowanie

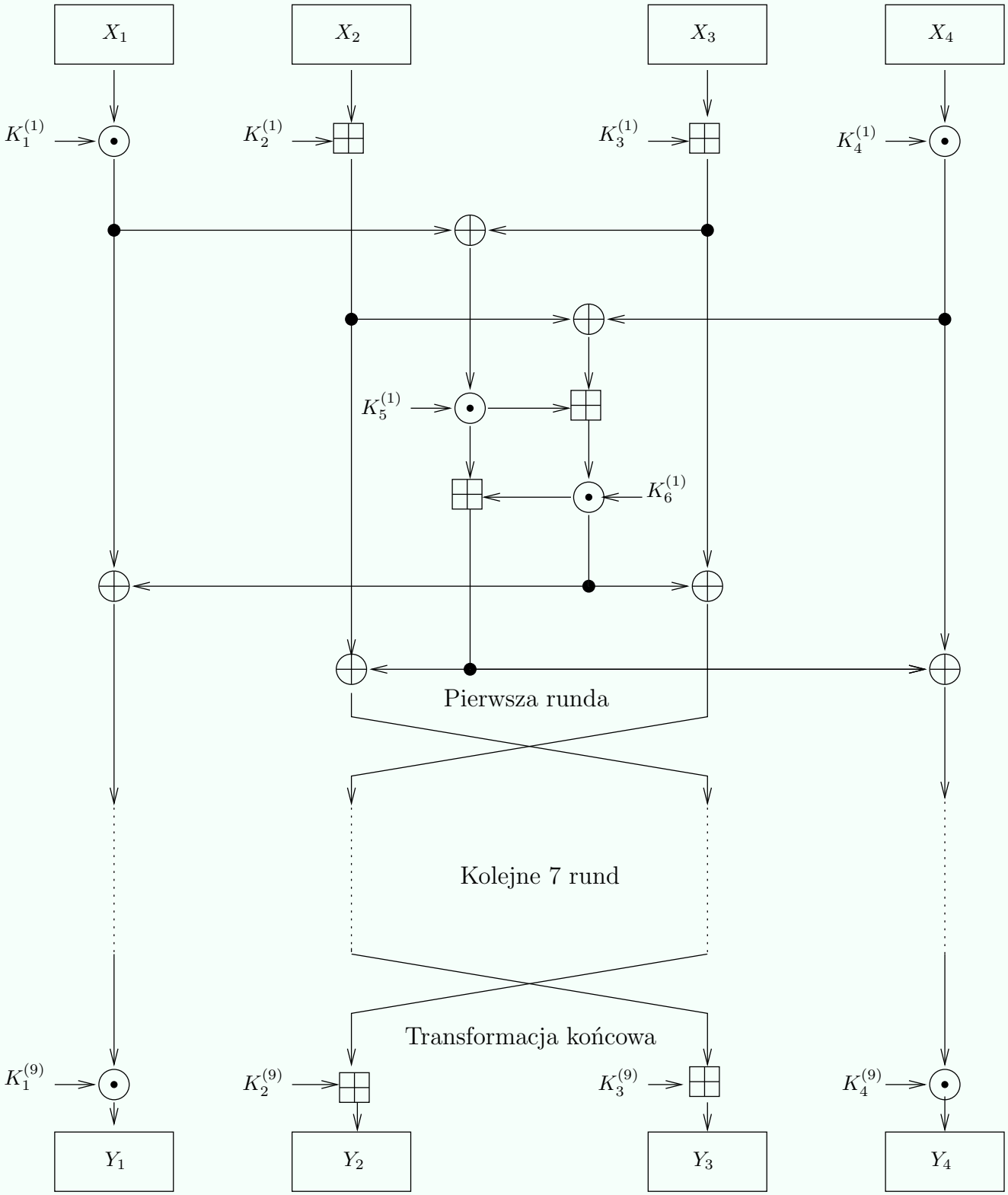
Rysunek 3: Schemat działania CFB

IDEA — International Data Encryption Algorithm

- IDEA jest algorytmem blokowym wprowadzonym w latach 90-tych
- IDEA używa kluczy 128 bitowych
- IDEA jest używana w pakiecie PGP
- IDEA jest algorytmem opatentowanym; można go używać bezpłatnie do celów niekomercyjnych
- IDEA działa na blokach 64 bitowych i wykorzystuje 3 różne operacje: XOR ($\oplus$), dodawanie modulo 2^{16} ($\boxplus$) oraz mnożenie modulo $2^{16}+1$ ($\odot$); schemat działania algorytmu przedstawiony jest na Rys. 4

Szyfrowanie

- 64 bitowy blok jest dzielony na 4 bloki po 16 bitów: X_1, X_2, X_3, X_4, które stanowią dane wejściowe dla pierwszej rundy algorytmu
- algorytm składa się z 8 rund
- w każdej rundzie wykonywane są wymienione wyżej 3 typy operacji na 16 bitowych blokach z 16 bitowymi podkluczami (każda runda wymaga 6 podkluczy)
- w wyniku otrzymuje się 4 bloki po 16 bitów: Y_1, Y_2, Y_3, Y_4
- pomiędzy rundami blok 2 i 3 są zamieniane
- algorytm kończy przekształcenie końcowe, które wymaga 4 podkluczy



Rysunek 4: Schemat działania algorytmu IDEA:
$\oplus$ — operacja XOR, $\boxplus$ — dodawanie modulo 2^{16},
$\odot$ — mnożenie modulo $2^{16}+1$,
$K_i^{(r)}$ — i-ty podklucz dla r-tej rundy

Generowanie podkluczy

- IDEA używa klucza 128 bitowego i wymaga 8×6+4=52 podklucze
  1. 128 bitowy klucz jest dzielony na bloki 16 bitowe, co daje 8 podkluczy
  2. na kluczu wykonuje się przesunięcie cykliczne o 25 pozycji i znowu dzieli na bloki 16 bitowe, co daje kolejne 8 podkluczy
  3. operację z punktu 2 powtarza się tak długo aż wygeneruje się wszystkie podklucze

Deszyfrowanie

- Deszyfrowanie algorytmem IDEA przebiega wg schematu przedstawionego na Rys. 4, w którym zamiast X_1, X_2, X_3, X_4 na wejściu podaje się bloki Y_1, Y_2, Y_3, Y_4 kryptogramu oraz klucz K (ten sam co przy szyfrowaniu)
- z klucza K generuje się podklucze $K_i^{(r)}$
- generuje się podklucze deszyfrujące ${K'}_i^{(r)}$ wg schematu przedstawionego w Tablicy 8

Runda	${K'}_1^{(r)}$	${K'}_2^{(r)}$	${K'}_3^{(r)}$	${K'}_4^{(r)}$	${K'}_5^{(r)}$	${K'}_6^{(r)}$
$r=1$	$\left(K_1^{(10-r)}\right)^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$\left(K_4^{(10-r)}\right)^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$2\leq r\leq 8$	$\left(K_1^{(10-r)}\right)^{-1}$	$-K_3^{(10-r)}$	$-K_2^{(10-r)}$	$\left(K_4^{(10-r)}\right)^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$r=9$	$\left(K_1^{(10-r)}\right)^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$\left(K_4^{(10-r)}\right)^{-1}$	—	—

Tablica 8: Tworzenie podkluczy deszyfrujących ${K'}_i^{(r)}$ na podstawie podkluczy szyfrujących $K_i^{(r)}$ w algorytmie IDEA (dla $K_i = 0$ przyjmuje się $(K_i)^{-1} = 0$; $2^{16} \equiv -1 \bmod 2^{16}+1$)

AES — Advanced Encryption Standard — Rijndael

- Nowy standard przyjęty w 2001 r. w USA
- algorytm blokowy, który zaprojektowali Joan Daemen i Vincent Rijmen
- zarówno długość bloku jak i klucza może być wybrana jako 128, 192 lub 256 bitów
- Rijndael jest ogólnie dostępny
- liczba rund zależy od długości bloku
- w każdej rundzie wykonywane są 4 operacje (macierzowe): podstawienie w S-boksie, przesunięcie wierszy, mieszanie kolumn i XOR z podkluczem
- podklucze są generowane algorytmem, który zależy od rundy

Algorytmy asymetryczne — RSA

- Witfield Diffie i Martin Hellman — idea kryptografii z kluczem publicznym, rok 1976
- **RSA** — Ron **R**ivest, Adi **S**hamir i Leonard **A**dleman, rok 1978
- bezpieczeństwo algorytmu RSA opiera się na trudności obliczeniowej związanej z rozkładem dużych liczb na czynniki (faktoryzacja)
  Wybierzmy dwie duże liczby pierwsze p i q i obliczmy ich iloczyn (iloczyn łatwo obliczyć)
  $$n = pq,$$
  następnie wybierzmy losowo liczbę $e < n$ względnie pierwszą z liczbą $(p-1)(q-1)$. Liczba e będzie kluczem szyfrującym. Teraz znajdźmy liczbę d taką, że
  $$ed \equiv 1 \pmod{(p-1)(q-1)},$$
  lub inaczej
  $$d \equiv e^{-1} \pmod{(p-1)(q-1)}.$$
  Liczby d i n są także względnie pierwsze. Do obliczenia d można użyć rozszerzonego algorytmu Euklidesa. Liczba d jest kluczem deszyfrującym. Liczby $\{e, n\}$ stanowią klucz publiczny, który ujawniamy, zaś liczby $\{d, n\}$ stanowią klucz prywatny, który powinien być ściśle chroniony (liczba d)
- **Szyfrowanie**
  Wiadomość dzielimy na bloki m_i mniejsze niż n, które szyfrujemy używając formuły
  $$c_i \equiv m_i^e \pmod{n}$$

- **Deszyfrowanie**

  Tekst jawny z kryptogramu otrzymujemy obliczając

$$m_i \equiv c_i^d \pmod{n}$$

- **Uzasadnienie**

  Ponieważ $ed \equiv 1 \pmod{(p-1)(q-1)}$, to istnieje liczba całkowita k taka, że $ed = 1 + k(p-1)(q-1)$. Z małego twierdzenia Fermata, dla $NWD(m,p) = 1$, mamy

$$m^{p-1} \equiv 1 \pmod{p}$$

  podnosząc obie strony tej kongruencji do potęgi $k(q-1)$ oraz mnożąc przez m otrzymujemy

$$m^{1+k(p-1)(q-1)} \equiv m \pmod{p}$$

  Kongruencja ta jest także prawdziwa dla $NWD(m,p) = p$, ponieważ wtedy obie strony przystają do $0 \pmod{p}$. Zatem, zawsze mamy

$$m^{ed} \equiv m \pmod{p}.$$

  Podobnie,

$$m^{ed} \equiv m \pmod{q},$$

  a ponieważ p i q są różnymi liczbami pierwszymi, to z chińskiego twierdzenia o resztach otrzymujemy

$$m^{ed} \equiv m \pmod{n}$$

- **Przykład (trywialny):**
  Znajdowanie klucza:

$$\begin{aligned} p &= 1123 \qquad q = 1237 \\ n &= pq = 1389151 \\ \phi &= (p-1)(q-1) = 1386792 \\ e &= 834781 \\ d &\equiv e^{-1} \pmod{\phi} = 1087477 \end{aligned}$$

Szyfrowanie:

$$\begin{aligned} m &= 983415 \\ c &\equiv m^e \pmod{n} \\ & 983415^{834781} \pmod{1389151} = 190498 \end{aligned}$$

Deszyfrowanie:

$$\begin{aligned} m &\equiv c^d \pmod{n} \\ & 190498^{1087477} \pmod{1389151} = 983415 \end{aligned}$$

Trochę matematyki

- **Podzielność liczb**
  - ⋆ Dla danych liczb całkowitych a i b mówimy, że liczba b jest podzielna przez a lub, że liczba a dzieli liczbę b, jeżeli istnieje taka liczba całkowita d, że $b = ad$. Liczbę a nazywamy dzielnikiem liczby b, a fakt ten zapisujemy $a|b$.
  - ⋆ Każda liczba $b > 1$ ma co najmniej dwa dzielniki dodatnie: 1 i b.
  - ⋆ Dzielnikiem nietrywialnym liczby b nazywamy dzielnik dodatni różny od 1 i b.
  - ⋆ Liczba pierwsza to liczba większa od 1 nie mająca innych dzielników dodatnich niż 1 i ona sama.
  - ⋆ Liczba mająca co najmniej jeden nietrywialny dzielnik jest liczbą złożoną.

- **Twierdzenie o rozkładzie na czynniki pierwsze**
  Każda liczba naturalna n może być przedstawiona jednoznacznie (z dokładnością do kolejności czynników) jako iloczyn liczb pierwszych.
  Zwykle taki rozkład zapisujemy jako iloczyn odpowiednich potęg różnych liczb pierwszych, np. $6600 = 2^3 \cdot 3 \cdot 5^2 \cdot 11$.

- **Własności relacji podzielności**
  1. Jeśli $a|b$ i c jest dowolną liczbą całkowitą, to $a|bc$.
  2. Jeśli $a|b$ i $b|c$, to $a|c$
  3. Jeśli $a|b$ i $a|c$, to $a|b \pm c$
  4. Jeśli liczba pierwsza p dzieli ab, to $p|a$ lub $p|b$
  5. Jeśli $m|a$ i $n|a$ oraz m i n nie mają wspólnych dzielników większych od 1, to $mn|a$

- **Największy wspólny dzielnik** — $NWD(a,b)$
  Największy wspólny dzielnik, $NWD(a,b)$, dla danych dwóch liczb całkowitych (nie będących jednocześnie zerami), to największa liczba całkowita d będąca dzielnikiem zarówno a, jak i b.
  Przykład: $NWD(12,18) = 6$
- **Najmniejsza wspólna wielokrotność** — $NWW(a,b)$
  Najmniejsza wspólna wielokrotność, $NWW(a,b)$, to najmniejsza dodatnia liczba całkowita, którą dzielą a i b.
  $NWW(a,b) = a \cdot b / NWD(a,b)$
  Przykład: $NWW(12,18) = 36 = 12 \cdot 18 / NWD(12,18)$
- **Liczby względnie pierwsze**
  Liczby a i b są względnie pierwsze jeżeli $NWD(a,b) = 1$, tzn. liczby a i b nie mają wspólnego dzielnika większego od 1.
  $NWD(841,160) = 1$ zatem liczby 841 i 160 są względnie pierwsze (patrz niżej)
- **Algorytm Euklidesa**
  Algorytm Euklidesa pozwala znaleźć $NWD(a,b)$ w czasie wielomianowym (dla $a > b$, $O(\ln^2(a))$)
  Dla $a > b$, dzielimy a przez b otrzymując iloraz q_1 i resztę r_1, tzn. $a = q_1 b + r_1$, w następnym kroku b gra rolę a, zaś r_1 gra rolę b: $b = q_2 r_1 + r_2$. Postępowanie to kontynuujemy dzieląc kolejne reszty, $r_{i-2} = q_i r_{i-1} + r_i$, aż do momentu kiedy otrzymamy resztę, która dzieli poprzednią resztę. Ostatnia niezerowa reszta jest $NWD(a,b)$.

Obliczmy $NWD(841, 160)$

$$\begin{aligned} 841 &= 5 \cdot 160 + 41 \\ 160 &= 3 \cdot 41 + 37 \\ 41 &= 1 \cdot 37 + 4 \\ 37 &= 9 \cdot 4 + 1 \\ 4 &= 4 \cdot 1 + 0 \end{aligned}$$

Ponieważ $NWD(841, 160) = 1$, to liczby 841 i 160 są względnie pierwsze.

- **Twierdzenie**
  Największy wspólny dzielnik dwóch liczb może być przedstawiony w postaci kombinacji liniowej tych liczb ze współczynnikami całkowitymi: $NWD(a, b) = xa + yb$, przy czym liczby x i y można znaleźć w czasie $O(\ln^2(a))$.
  W poprzednim przykładzie $NWD(841, 160) = 1$. Korzystając z ciągu równości w algorytmie Euklidesa (idąc w przeciwną stronę) otrzymujemy

$$\begin{aligned} 1 &= 37 - 9 \cdot 4 \\ &= 37 - 9(41 - 1 \cdot 37) = 10 \cdot 37 - 9 \cdot 41 \\ &= 10(160 - 3 \cdot 41) - 9 \cdot 41 = 10 \cdot 160 - 39 \cdot 41 \\ &= 10 \cdot 160 - 39 \cdot (841 - 5 \cdot 160) \\ &= -39 \cdot 841 + 205 \cdot 160 \end{aligned}$$

  Zatem $x = -39$ i $y = 205$.

- **Rozszerzony algorytm Euklidesa**
  Rozszerzony algorytm Euklidesa znajduje zarówno największy wspólny dzielnik $NWD(a, b)$ liczb a i b jak i liczby x i y będące współczynnikami kombinacji liniowej $NWD(a, b) = xa + yb$.

q	r	a	b	x	x_2	x_1	y	y_2	y_1
—	—	841	160	—	1	0	—	0	1
5	41	160	41	1	0	1	-5	1	-5
3	37	41	37	-3	1	-3	16	-5	16
1	4	37	4	4	-3	4	-21	16	-21
9	1	4	1	-39	4	-39	205	-21	205

Na każdyn etapie mamy $r = x \cdot 841 + y \cdot 160$. Z ostatniego wiersza odczytujemy:

$NWD(841, 160) = 1 = -39 \cdot 841 + 205 \cdot 160$

Przyporządkowania w algorytmie są następujące:

$q = \lfloor a/b \rfloor, \quad r \leftarrow a - qb, \quad x \leftarrow x_2 - qx_1, \quad y \leftarrow y_2 - qy_1$

$a \leftarrow b, \quad b \leftarrow r, \quad x_2 \leftarrow x_1, \quad x_1 \leftarrow x, \quad y_2 \leftarrow y_1, \quad y_1 \leftarrow y$

- **Kongruencje**

  Dla danych trzech liczb całkowitych a, b i m mówimy, że liczba a przystaje do liczby b modulo m i piszemy $a \equiv b \pmod{m}$, gdy różnica $a - b$ jest podzielna przez m. Liczbę m nazywamy modułem kongruencji.

  Własności

  1. $a \equiv a \pmod{m}$
  2. $a \equiv b \pmod{m}$ wtedy i tylko wtedy, gdy $b \equiv a \pmod{m}$
  3. Jeśli $a \equiv b \pmod{m}$ oraz $b \equiv c \pmod{m}$, to $a \equiv c \pmod{m}$
  4. Jeśli $a \equiv b \pmod{m}$ i $c \equiv d \pmod{m}$, to $a \pm c \equiv b \pm d \pmod{m}$ oraz $ac \equiv bd \pmod{m}$

     Kongruencje względem tego samego modułu można dodawać, odejmować i mnożyć stronami.

5. Jeśli $a \equiv b \pmod{m}$, to $a \equiv b \pmod{d}$ dla każdego dzielnika $d|m$
6. Jeśli $a \equiv b \pmod{m}$, $a \equiv b \pmod{n}$, oraz m i n są względnie pierwsze, to $a \equiv b \pmod{mn}$
7. Dla ustalonej liczby m, każda liczba przystaje modulo m do jednej liczby zawartej pomiędzy 0 i $m-1$.

Przykłady:

$27 \equiv 7 \pmod{5}$ bo $27 - 7 = 4 \cdot 5$

$27 \equiv 2 \pmod{5}$ bo $27 - 2 = 5 \cdot 5$

$-8 \equiv 7 \pmod{5}$ bo $-8 - 7 = -3 \cdot 5$

- **Twierdzenie**

  Liczbami a, dla których istnieje liczba b taka, że $ab \equiv 1 \pmod{m}$, są dokładnie te liczby a, dla których $NWD(a, m) = 1$. Taka liczba odwrotna $b = a^{-1}$ może być znaleziona w czasie $O(\ln^2(m))$.

  Ponieważ $NWD(841, 160) = 1$ (patrz poprzedni przykład), to istnieje liczba $160^{-1} \pmod{841}$. Liczbę tę można obliczyć za pomocą rozszerzonego algorytmu Euklidesa. Ponieważ $1 = -39 \cdot 841 + 205 \cdot 160$, to $205 \cdot 160 \equiv 1 \pmod{841}$, a więc $160^{-1} \pmod{841} = 205$.

- **Małe twierdzenie Fermata**

  Niech p będzie liczbą pierwszą. Wtedy każda liczba a spełnia kongruencję $a^p \equiv a \pmod{p}$ i każda liczba a niepodzielna przez p spełnia kongruencję $a^{p-1} \equiv 1 \pmod{p}$.

  Liczba 1231 jest liczbą pierwszą i $NWD(1231, 5871) = 1$, więc $5871^{1230} \equiv 1 \pmod{1231}$

- **Chińskie twierdzenie o resztach**

  Jeśli liczby $m_1, m_2, \ldots, m_k$ są parami względnie pierwsze, tzn. $NWD(m_i, m_j) = 1$ dla $i \neq j$, wtedy układ kongruencji

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ \ldots & \quad \ldots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

  ma wspólne rozwiązanie modulo $m = m_1 m_2 \ldots m_k$.
  Przykład:

$$\begin{aligned} x &\equiv 1 \pmod{11} \\ x &\equiv 2 \pmod{12} \\ x &\equiv 3 \pmod{13} \end{aligned}$$

  Niech $M_i = m/m_i$ będzie iloczynem wszystkich modułów z wyjątkiem i-tego. Wtedy $NWD(m_i, M_i) = 1$, a więc istnieje taka liczba N_i, że $M_i N_i \equiv 1 \pmod{m_i}$, wtedy wspólnym rozwiązaniem modulo m jest $x = \sum_i a_i M_i N_i$. Dla każdego i wszystkie składniki sumy poza i-tym są podzielne przez m_i, gdyż $m_i | M_j$ dla $j \neq i$ zatem dla każdego i mamy $x \equiv a_i M_i N_i \equiv a_i \pmod{m_i}$. W naszym przykładzie mamy: $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, $m_1 = 11$, $m_2 = 12$, $m_3 = 13$, $m = 1716$, $M_1 = 156$, $M_2 = 143$, $M_3 = 132$. Aby znaleźć wspólne rozwiązanie tego układu kongruencji należy znaleźć liczby N_i będące odwrotnościami liczb M_i modulo m_i. W tym celu możemy użyć algorytmu Euklidesa. W wyniku otrzymujemy liczby: $N_1 = 6$, $N_2 = 11$ i $N_3 = 7$. Zatem wspólnym rozwiązaniem jest $x \equiv 6 \cdot 156 + 2 \cdot 11 \cdot 143 + 3 \cdot 7 \cdot 132$

$(\mathrm{mod}\ 1716) \equiv 1706\ (\mathrm{mod}\ 1716)$. W tym przykładzie widać, że liczba -10 daje takie reszty zatem $x = -10 + 1716$.

- **Funkcja Eulera**

  Dla $n \geq 1$, niech $\phi(n)$ będzie liczbą tych nieujemnych liczb b mniejszych od n, które są względnie pierwsze z n. Funkcja $\phi(n)$ nazywa się funkcją Eulera.

  Funkcja Eulera ϕ jest „multiplikatywna", tzn. $\phi(mn) = \phi(m)\phi(n)$, jeśli tylko $NWD(m,n) = 1$.

- **Twierdzenie Eulera**

  Jeśli $NWD(a,m) = 1$, to $a^{\phi(m)} \equiv 1\ (\mathrm{mod}\ m)$.

  Wniosek

  Jeśli $NWD(a,m) = 1$ i jeśli n' jest resztą z dzielenia n przez $\phi(m)$, to $a^n \equiv a^{n'}\ (\mathrm{mod}\ m)$

- **Potęgowanie modulo metodą iterowanego podnoszenia do kwadratu**

  Podstawowym działaniem w kryptografii jest obliczanie $a^n\ (\mathrm{mod}\ m)$, gdzie m i n są bardzo dużymi liczbami. Zauważmy, że rozwinięcie dwójkowe liczby n ma postać

$$n = \sum_{i=0}^{k-1} n_i 2^i = n_0 + 2n_1 + 4n_2 + \cdots + 2^{k-1} n_{k-1}$$

  ,

  gdzie $n_i \in \{0,1\}$ są cyframi rozwinięcia dwójkowego. Zatem

$$a^n = \prod_{i=0}^{k-1} a^{n_i 2^i} = \left(a^{2^0}\right)^{n_0} \left(a^{2^1}\right)^{n_1} \ldots \left(a^{2^{k-1}}\right)^{n_{k-1}}$$

  Załóżmy, że $a < m$ oraz przyjmijmy, że przez b będziemy oznaczali częściowe iloczyny. Na początku $b = 1$. Jeżeli

$n_0 = 1$ to zastępujemy b przez a, w przeciwnym przypadku nadal $b = 1$. Następnie liczymy $a_1 \equiv a^2 \pmod{m}$. Jeśli $n_1 = 1$, to mnożymy b przez a_1 i redukujemy modulo m, zaś jeśli $n_1 = 0$ nie zmieniamy b. Następnie liczymy $a_2 \equiv a_1^2 \pmod{m}$. Znowu, jeśli $n_2 = 1$, to mnożymy b przez a_2; w przeciwnym przypadku nie zmieniamy b. Postępując dalej w ten sposób, w j-tym kroku mamy obliczoną potęgę $a_j \equiv a^{2j} \pmod{m}$. Jeśli $n_j = 1$ to włączamy a_j do iloczynu b, jeśli $n_j = 0$ to b się nie zmienia. Po $k-1$ krokach otrzymamy $b \equiv a^n \pmod{m}$.

Przykład:

Obliczmy $7^{698} \pmod{1234} = 287$

i	0	1	2	3	4	5	6	7	8	9
n_i	0	1	0	1	1	1	0	1	0	1
a_i	7	49	1167	787	1135	1163	105	1153	391	1099
b	1	49	49	309	259	121	121	71	71	287

- **Czy wystarczy liczb pierwszych?**

  Twierdzenie

  Niech $\pi(x)$ oznacza liczbę liczb pierwszych $\leq x$. Wtedy

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1$$

  Dla $x \geq 17$

$$\pi(x) > \frac{x}{\ln x}$$

  Dla przykładu, dla $x = 10^{10}$, $\pi(x) = 455052511$, natomiast $\lfloor x/\ln x \rfloor = 434294481$.

- **Testy pierwszości**

  Istnieją probabilistyczne testy pierwszości liczb, które pozwalają z dużym prawdopodobieństwem w skończonym czasie dać odpowiedź czy dana liczba jest pierwsza.

  Test Fermata

  Testujemy czy liczba n jest pierwsza. Wybieramy losowo liczbę $a < n-1$, obliczamy $r = a^{n-1} \pmod n$, jeśli $r \neq 1$ to n jest liczbą złożoną. Test przeprowadzamy t-krotnie, $t \geq 1$. Jeśli wszystkie testy wypadną pomyślnie, tzn. $r = 1$, to liczbę uznajemy za pierwszą, choć może tak nie być.

  Test Millera-Rabina

  Testujemy czy liczba n jest pierwsza. Piszemy $n-1 = 2^s r$, gdzie r jest nieparzyste. Wybieramy losowo liczbę a, $1 < a < n-1$. Obliczamy $b = a^r \pmod n$. Jeśli $b \equiv \pm 1 \pmod n$ to uznajemy, że n jest pierwsza. W przeciwnym przypadku obliczamy $a^{2^j r} \pmod n$ dla $0 < j < s$. Jeśli dla pewnego $j < s$ otrzymamy $a^{2^j r} \equiv -1 \pmod n$ to uznajemy, że liczba n jest pierwsza. W przeciwnym przypadku liczba n jest złożona. Test przeprowadzamy t-krotnie losując różne a.

- **Reszty kwadratowe w $\mathbb{Z}_p^*$**

  ⋆ Oznaczmy przez $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ zbiór reszt modulo p, gdzie $p > 2$ jest nieparzystą liczbą pierwszą; np. $\mathbb{Z}_{11} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

  ⋆ Przez $\mathbb{Z}_p^*$ będziemy oznaczali zbiór niezerowych elementów zbioru $\mathbb{Z}_p$, a więc np. $\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

  ⋆ W zbiorze $\mathbb{Z}_p^*$ szukamy takich elementów, które są kwadratami innych elementów, tzn. spełniona jest kongruencja $x^2 \equiv a \pmod{p}$, dla $\{x, a\} \in \mathbb{Z}_p^*$.
  Liczby a, które są kwadratami nazywamy resztami kwadratowymi modulo p, zaś pozostałe elementy nazywamy nieresztami.

Przykład:
Weźmy $\mathbb{Z}_{11}^*$ i policzmy $x^2 \pmod{11}$ dla wszystkich x, mamy wtedy

x	1	2	3	4	5	6	7	8	9	10
$a = x^2 \pmod{11}$	1	4	9	5	3	3	5	9	4	1

Resztami kwadratowymi w $\mathbb{Z}_{11}^*$ są więc liczby $\{1, 3, 4, 5, 9\}$, a pozostale liczby $\{2, 6, 7, 8, 10\}$ są nieresztami

Symbol Legendre'a

Niech a będzie liczbą całkowitą zaś $p > 2$ liczbą pierwszą; symbol Legendre'a definiujemy

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{jeśli } p|a \\ 1, & \text{jeśli } a \text{ jest } \textit{resztą kwadratową} \text{ modulo } p \\ -1, & \text{jeśli } a \text{ jest } \textit{nieresztą} \text{ modulo } p \end{cases}$$

Twierdzenie

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \pmod{p}$$

Własności symbolu Legendre'a

(1) $\left(\frac{a}{p}\right)$ zależy tylko od a modulo p

(2) $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$

(3) $\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right)$, jeśli $NWD(b,p) = 1$

(4) $\left(\frac{1}{p}\right) = 1$ oraz $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$

Twierdzenie

$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} = \begin{cases} 1, & \text{jeśli } p \equiv \pm 1 \pmod 8 \\ -1, & \text{jeśli } p \equiv \pm 3 \pmod 8 \end{cases}$$

Prawo wzajemności

Niech p i q będą dwiema nieparzystymi liczbami pierwszymi. Wtedy

$$\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}} \left(\frac{p}{q}\right)$$

$$= \begin{cases} -\left(\frac{p}{q}\right), & \text{jeśli } p \equiv q \equiv 3 \pmod 4 \\ \left(\frac{p}{q}\right), & \text{w przeciwnym przypadku} \end{cases}$$

Przykład:

$$\begin{aligned}
\left(\frac{91}{167}\right) &= \left(\frac{7\cdot 13}{167}\right) = \left(\frac{7}{167}\right)\left(\frac{13}{167}\right) \\
&= (-1)^{\frac{7-1}{2}\frac{167-1}{2}}\left(\frac{167}{7}\right)(-1)^{\frac{13-1}{2}\frac{167-1}{2}}\left(\frac{167}{13}\right) \\
&= -\left(\frac{167}{7}\right)\left(\frac{167}{13}\right) = -\left(\frac{6}{7}\right)\left(\frac{11}{13}\right) \\
&= -\left(\frac{2}{7}\right)\left(\frac{3}{7}\right)\left(\frac{11}{13}\right) = -\left(\frac{3}{7}\right)\left(\frac{11}{13}\right) \\
&- (-1)^{\frac{3-1}{2}\frac{7-1}{2}}\left(\frac{7}{3}\right)(-1)^{\frac{11-1}{2}\frac{13-1}{2}}\left(\frac{13}{11}\right) \\
&= \left(\frac{1}{3}\right)\left(\frac{2}{11}\right) = 1\cdot(-1) = -1
\end{aligned}$$

Pierwiastki kwadratowe modulo p

⋆ Prawo wzajemności pozwala szybko stwierdzić czy a jest resztą kwadratową modulo p, a więc mówi, że istnieje rozwiązanie kongruencji

$$x^2 \equiv a \pmod{p},$$

chociaż nie daje wskazówek jak takie rozwiązanie znaleźć.

⋆ Nie jest znany efektywny deterministyczny algorytm obliczania pierwiastków kwadratowych w $\mathbb{Z}_p^*$. Istnieje natomiast *efektywny algorytm probabilistyczny* dla obliczania takich pierwiastków jeśli p jest liczbą pierwszą.

• Reszty kwadratowe w $\mathbb{Z}_n^*$

⋆ Oznaczmy przez $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ zbiór reszt modulo n, gdzie n jest dodatnią liczbą całkowitą; np. $\mathbb{Z}_{15} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$

⋆ Przez $\mathbb{Z}_n^*$ będziemy oznaczali podzbiór tych elementów

zbioru $\mathbb{Z}_n$, które są względnie pierwsze z n, a więc np. $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$. Liczba elementów zbioru $\mathbb{Z}_n^*$ jest równa wartości funkcji Eulera $\phi(n)$.

⋆ W zbiorze $\mathbb{Z}_n^*$ szukamy takich elementów, które są kwadratami innych elementów, tzn. spełniona jest kongruencja $x^2 \equiv a \pmod{n}$, dla $\{x, a\} \in \mathbb{Z}_n^*$.

Symbol Jacobiego

Niech a będzie liczbą całkowitą i niech n będzie dowolną dodatnią liczbą nieparzystą. Niech $n = p_1^{\alpha_1} \cdot \ldots \cdot p_r^{\alpha_r}$ będzie rozkładem liczby n na czynniki pierwsze. Wtedy definiujemy *symbol Jacobiego* (uogólnienie symbolu Legendre'a) jako iloczyn symboli Legendre'a dla dzielników pierwszych n

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \cdot \ldots \cdot \left(\frac{a}{p_r}\right)^{\alpha_r}$$

Twierdzenie

Dla dowolnej dodatniej liczby nieparzystej n mamy

$$\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}$$

Twierdzenie

Dla dowolnych dodatnich liczb nieparzystych m i n mamy

$$\left(\frac{m}{n}\right) = (-1)^{\frac{m-1}{2}\frac{n-1}{2}} \left(\frac{n}{m}\right)$$

Uwaga

Jeśli liczba $a \in \mathbb{Z}_n^*$ jest resztą kwadratową to $\left(\frac{a}{n}\right) = 1$.
Jeśli symbol Jacobiego $\left(\frac{a}{n}\right) = 1$ dla liczby złożonej n to a nie musi być resztą kwadratową!

- **Pierwiastki kwadratowe modulo n**
  - ⋆ Jeśli $n = pq$ jest iloczynem dwóch dużych, różnych liczb pierwszych, to uważa się, że znajdowanie pierwiastków kwadratowych w $\mathbb{Z}_n^*$ należy do problemów trudnych obliczeniowo! Trudność ta jest równoważna trudności z faktoryzacją liczby n. (Faktoryzując n znajdujemy liczby pierwsze p i q, znajdujemy pierwiastki kwadratowe w $\mathbb{Z}_p^*$ oraz $\mathbb{Z}_q^*$, a następnie korzystając z chińskiego twierdzenia o resztach znajdujemy pierwiastki w $\mathbb{Z}_n^*$.)
- **Logarytm dyskretny**
  - ⋆ Niech p będzie liczbą pierwszą, przez $\mathbb{Z}_p^*$ oznaczamy zbiór liczb $\{1, \ldots, p-1\}$ i niech g będzie generatorem $\mathbb{Z}_p^*$, tzn. takim elementem, że dla każdej liczby $a \in \mathbb{Z}_p^*$ istnieje takie i, że $a \equiv g^i \pmod{p}$ (wszystkie elementy mogą być wygenerowane z g).
  - ⋆ Problem logarytmu dyskretnego polega na znalezieniu dla danej liczby $0 < b < p$ takiej liczby a, że $g^a \equiv b \pmod{p}$.
  - ⋆ Problem znajdowania logarytmu dyskretnego jest problemem trudnym obliczeniowo!

Przykład:
Weźmy $\mathbb{Z}_{19}^*$, czyli zbiór liczb $\{1, \ldots, 18\}$ oraz $g = 2$.
Niech $b = 2^a \pmod{19}$, mamy wtedy

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
b	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1

Tak więc w $\mathbb{Z}_{19}^*$, np.

$$\log_2 13 = 5$$

$$2^5 \equiv 13 \pmod{19}$$

- **PARI/GP — teoria liczb na komputerze**

```
                GP/PARI CALCULATOR Version 2.1.6 (released)
                     i386 running linux 32-bit version
               (readline v4.3 enabled, extended help available)

                            Copyright (C) 2002 The PARI Group

PARI/GP is free software, covered by the GNU General Public License, and
comes WITHOUT ANY WARRANTY WHATSOEVER.

Type ? for help, \q to quit.
Type ?12 for how to get moral (and possibly technical) support.

   realprecision = 28 significant digits
   seriesprecision = 16 significant terms
   format = g0.28

parisize = 4000000, primelimit = 500000
```

Dzielenie z resztą

```
? divrem(841,160)
%1 = [5, 41]~
? divrem(2987634211432123123,8765392)
%2 = [340844335476, 5476531]~
```

Algorytm Euklidesa — $NWD(a, b)$

```
? gcd(841,160)
%2 = 1
? gcd(2987634211432123123,8765392)
%3 = 1
? gcd(739834587231984763212876546,497563772132052)
%4 = 6
```

Rozszerzony algorytm Euklidesa

```
? bezout(841,160)
%4 = [-39, 205, 1]
? bezout(2987634211432123123,8765392)
%5 = [2987931, -1018419356145006986, 1]
```

Odwrotność modulo n

```
? Mod(160,841)^(-1)
%6 = Mod(205, 841)
? lift(Mod(160,841)^(-1))
%7 = 205

? Mod(8765392,2987634211432123123)^(-1)
%8 = Mod(1969214855287116137,2987634211432123123)
? 2987634211432123123-1018419356145006986
%9 = 1969214855287116137
```

Małe twierdzenie Fermata

```
? isprime(1231)
%10 = 1
? gcd(1231,5871)
%11 = 1
```

```
? Mod(5871^1230, 1231)
%12 = Mod(1, 1231)
? Mod(5871,1231)^1230
%13 = Mod(1, 1231)

? Mod(40547201659, 85115991299)^85115991298
%14 = Mod(1, 85115991299)
? Mod(461730729350412, 2461654953439061)^2461654953439060
%15 = Mod(1, 2461654953439061)
```

Chińskie twierdzenie o resztach

```
? a=Mod(1,11)
%16 = Mod(1, 11)
? b=Mod(2,12)
%17 = Mod(2, 12)
? c=Mod(3,13)
%18 = Mod(3, 13)
? d=chinese(a,b)
%19 = Mod(122, 132)
? chinese(c,d)
%20 = Mod(1706, 1716)
```

Funkcja Eulera

```
? eulerphi(841)
%21 = 812
? factorint(841)
%22 =
[29 2]
? eulerphi(1231)
%23 = 1230
? isprime(1231)
%24 = 1
? eulerphi(1000)
```

```
%25 = 400
? eulerphi(1200)
%26 = 320
```

Potęgowanie modulo n

```
? lift(Mod(7,1234)^698)
%27 = 287
? Mod(461730729350412,2461654953439061)^2461654953439060
%28 = Mod(1, 2461654953439061)
```

Liczby pierwsze

```
? primes(10)
%31 = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
? prime(1000)
%32 = 7919
? nextprime(10^30)
%34 = 1000000000000000000000000000057
? nextprime(random(10^30))
%35 = 42517003983368073383323753668l
? isprime(%35)
%36 = 1
```

Symbol Jacobiego

```
? kronecker(91,167)
%37 = -1
? kronecker(7,167)
%38 = 1
? for(a=1,167,if(Mod(a,167)^2==7,print1(a, " ")))
72 95
? kronecker(13,167)
%39 = -1
? kronecker(6,7)
%40 = -1
```

```
? kronecker(11,13)
%41 = -1

? kronecker(12987616654165515 51,97869853217651987616651 1871)
%42 = 1
```

Logarytm dyskretny

```
? znprimroot(19)
%43 = Mod(2, 19)
? znorder(Mod(2,19))
%44 = 18
? znlog(13,Mod(2,19))
%45 = 5
? znlog(15,Mod(2,19))
%46 = 11

? znprimroot(966099377)
%47 = Mod(3, 966099377)
? znlog(124332,Mod(3, 966099377)
%48 = 120589994
? Mod(3, 966099377)^120589994
%49 = Mod(124332, 966099377)
```

RSA

```
? p=1123;q=1237;n=p*q
%50 = 1389151
? phin=eulerphi(n)
%51 = 1386792
? e=834781
%52 = 834781
? gcd(e,phin)
%53 = 1
? d=lift(Mod(e,phin)^(-1))
```

```
%54 = 1087477
? m=983415
%55 = 983415
? c=lift(Mod(m,n)^e)
%56 = 190498
? lift(Mod(c,n)^d)
%57 = 983415

? p=nextprime(random(10^25))
%60 = 6394410543977819029567513
? q=nextprime(random(10^24))
%61 = 574229193973116022705411
? n=p*q
%62 = 3671857212601577387349834975533584930459534912843
? phin=(p-1)*(q-1)
%63 = 3671857212601577387349828006893846979524482639920
? e=random(10^10)
%64 = 6534579775
? while(gcd(e,phin)!=1,e=e+1)
? e
%65 = 6534579779
? d=lift(Mod(e,phin)^(-1))
%66 = 1069086500747478961348196600845385395334981162219
? m=random(10^30)
%67 = 446763233106745131823069978264
? c=lift(Mod(m,n)^e)
%68 = 366071378740244632828540738063744965348554865640 0
? lift(Mod(c,n)^d)
%69 = 446763233106745131823069978264
```

Algorytm ElGamala

U podstaw działania algorytmu ElGamala leży matematyczny problem logarytmu dyskretnego.

- **Wybór klucza**
  Wybieramy odpowiednio dużą liczbę pierwszą p, taką, że obliczenie logarytmu dyskretnego jest praktycznie niewykonalne, wybieramy liczbę całkowitą $0 < a < p-1$ oraz liczbę g, następnie obliczamy $b \equiv g^a \pmod{p}$. Liczby $\{b, g, p\}$ stanowią klucz publiczny, zaś liczby $\{a, g, p\}$ klucz prywatny.

- **Szyfrowanie**
  Aby zaszyfrować wiadomość M wybieramy losowo liczbę k względnie pierwszą z $p-1$, a następnie obliczamy

$$\begin{aligned} c_1 &\equiv g^k \pmod{p} \\ c_2 &\equiv M\, b^k \pmod{p} \end{aligned}$$

  Para liczb c_1 i c_2 tworzy kryptogram, który jest dwukrotnie dłuższy od tekstu jawnego.

- **Deszyfrowanie**

$$M = c_2 (c_1^a)^{-1} \pmod{p}$$

- **Uzasadnienie**
  $c_2 (c_1^a)^{-1} \equiv M\, b^k (g^{ka})^{-1} \equiv M g^{ka} (g^{ka})^{-1} \equiv M \pmod{p}$

- **Prosty przykład**

  Znajdowanie klucza

  Niech $p = 229$ i $g = 6$, wybieramy $a = 70$, wtedy $b \equiv 6^{70} \pmod{229} = 97$, zatem klucz publiczny stanowią liczby $\{97, 6, 229\}$, zaś klucz prywatny stanowią liczby $\{70, 6, 229\}$

  Szyfrowanie

  Niech wiadomość $M = 130$, wybieramy $k = 127$ takie, że $NWD(127, 228) = 1$ (liczby tej nie ujawniamy)

$$\begin{aligned} c_1 &\equiv 6^{127} \pmod{229} = 155 \\ c_2 &\equiv 130 \cdot 97^{127} \pmod{229} = 169 \end{aligned}$$

  Deszyfrowanie

$$M = 169 \cdot (155^{70})^{-1} \pmod{229} \equiv 169 \cdot 36 \pmod{229} = 130$$

  PARI/GP

```
? p=229;a=70;
? g=znprimroot(p)
%1 = Mod(6, 229)
? b=lift(g^a)
%2 = 97
? M=130;k=127;
? gcd(k,p-1)
%3 = 1
? c1=lift(g^k)
%4 = 155
? c2=lift(Mod(M*b^k,p))
%5 = 169
? lift(Mod(c2*c1^(-a),p))
%6 = 130
```

Jednokierunkowe funkcje hashujące (skrótu)

- dla każdego X łatwo jest obliczyć $H(X)$
- $H(X)$ ma taką samą długość dla wszystkich tekstów X
- dla zadanego Y znalezienie takiego X, że $H(X) = Y$ jest praktycznie niemożliwe; funkcja **jednokierunkowa**
- dla danego X trudno znaleźć X' takie, że $H(X) = H(X')$; funkcja **słabo bezkonfliktowa**
- nie jest praktycznie możliwe znalezienie X i X' takich, że $X \neq X'$ oraz $H(X) = H(X')$; funkcja **silnie bezkonfliktowa**

Typowe zastosowania

- Przechowywanie haseł w komputerach
- Ochrona integralności danych

Algorytm MD5 (Message Digest)

Algorytm, zaprojektowany przez Rivesta, jest modyfikacją wcześniejszego algorytmu MD4. Wiadomość dowolnej długości jest przekształcona w jej 128 bitowy „odcisk palca” (sumę kontrolną, skrót wiadomości). Zasadnicza procedura algorytmu działa na blokach 512 bitowych przekształcając je w 128 bitowe skróty.

Etapy MD5

- **Krok 1**
  Wiadomość dowolnej długości jest uzupełniana w taki

sposób, że na końcu dodawany jest bit „1” i odpowiednia ilość zer, tak aby ostatni blok miał długość 448 bitów.

- **Krok 2**

  Do ostatniego bloku dodawana jest 64 bitowa liczba reprezentująca długość wiadomości (w bitach) i w ten sposób przygotowana wiadomość ma długość będącą całkowitą wielokrotnością 512 bitów. Tym samym wiadomość jest wielokrotnością 16 słów 32-bitowych. Niech $M_0, M_1, \ldots M_{N-1}$ oznaczają kolejne słowa wiadomości, gdzie N jest jest wielokrotnością 16.

- **Krok 3**

  Algorytm operuje na 32-bitowych zmiennych a, b, c, d, których wartości początkowe A, B, C, D w zapisie szesnastkowym są następujące:
  A =0x67452301
  B =0xefcdab89
  C =0x98badcfe
  D =0x10325476

- **Krok 4**

  Główna pętla algorytmu składa się z 4 rund, w każdej rundzie 16 razy wykonywane są operacje na 32 bitowych słowach. Operacje te zdefiniowane są przez 4 funkcje

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee (\neg X) \wedge Z \\ G(X,Y,Z) &= (X \wedge Z) \vee Y \wedge (\neg X) \\ H(X,Y,Z) &= X \oplus Y \oplus Z \\ I(X,Y,Z) &= Y \oplus (X \vee (\neg Z)) \end{aligned}$$

  gdzie $\oplus$ oznacza operację XOR, $\wedge$ operację AND, $\vee$ operację OR, zaś $\neg$ operację NOT.

$$\begin{array}{llll} 0 \oplus 0 = 0 & 0 \wedge 0 = 0 & 0 \vee 0 = 0 & \neg 0 = 1 \\ 0 \oplus 1 = 1 & 0 \wedge 1 = 0 & 0 \vee 1 = 1 & \neg 1 = 0 \\ 1 \oplus 0 = 1 & 1 \wedge 0 = 0 & 1 \vee 0 = 1 & \\ 1 \oplus 1 = 0 & 1 \wedge 1 = 1 & 1 \vee 1 = 1 & \end{array}$$

Niech $X_j = M_{i*16+j}$ oznacza j-te słowo w i-tym bloku wiadomości M $(j = 0, \ldots, 15, i = 0, \ldots, N/16 - 1)$, $\hookleftarrow s$ niech oznacza cykliczne przesunięcie w lewo o s bitów oraz zdefiniujmy liczby T_k $(k = 1, \ldots, 64)$ tak, że $T_k = \lfloor 2^{32} \cdot |\sin k| \rfloor$, gdzie k jest w radianach. Mamy wtedy:

⋆ **Runda 1**

Niech $[abcd\ j\ s\ k]$ oznacza operację
$a = b + ((a + F(b, c, d) + X_j + T_k) \hookleftarrow s)$,
wtedy 16 operacji tej rundy to:

$$
\begin{array}{llll}
[abcd\ \ 0\ 7\ \ 1] & [dabc\ \ 1\ 12\ \ 2] & [cdab\ \ 2\ 17\ \ 3] & [bcda\ \ 3\ 22\ \ 4] \\
[abcd\ \ 4\ 7\ \ 5] & [dabc\ \ 5\ 12\ \ 6] & [cdab\ \ 6\ 17\ \ 7] & [bcda\ \ 7\ 22\ \ 8] \\
[abcd\ \ 8\ 7\ \ 9] & [dabc\ \ 9\ 12\ 10] & [cdab\ 10\ 17\ 11] & [bcda\ 11\ 22\ 12] \\
[abcd\ 12\ 7\ 13] & [dabc\ 13\ 12\ 14] & [cdab\ 14\ 17\ 15] & [bcda\ 15\ 22\ 16]
\end{array}
$$

⋆ **Runda 2**

Niech $[abcd\ j\ s\ k]$ oznacza operację
$a = b + ((a + G(b, c, d) + X_j + T_k) \hookleftarrow s)$,
wtedy 16 operacji tej rundy to:

$$
\begin{array}{llll}
[abcd\ \ 1\ 5\ 17] & [dabc\ \ 6\ 9\ 18] & [cdab\ 11\ 14\ 19] & [bcda\ \ 0\ 20\ 20] \\
[abcd\ \ 5\ 5\ 21] & [dabc\ 10\ 9\ 22] & [cdab\ 15\ 14\ 23] & [bcda\ \ 4\ 20\ 24] \\
[abcd\ \ 9\ 5\ 25] & [dabc\ 14\ 9\ 26] & [cdab\ \ 3\ 14\ 27] & [bcda\ \ 8\ 20\ 28] \\
[abcd\ 13\ 5\ 29] & [dabc\ \ 2\ 9\ 30] & [cdab\ \ 7\ 14\ 31] & [bcda\ 12\ 20\ 32]
\end{array}
$$

⋆ **Runda 3**

Niech $[abcd\ j\ s\ k]$ oznacza operację
$a = b + ((a + H(b, c, d) + X_j + T_k) \hookleftarrow s)$,
wtedy 16 operacji tej rundy to:

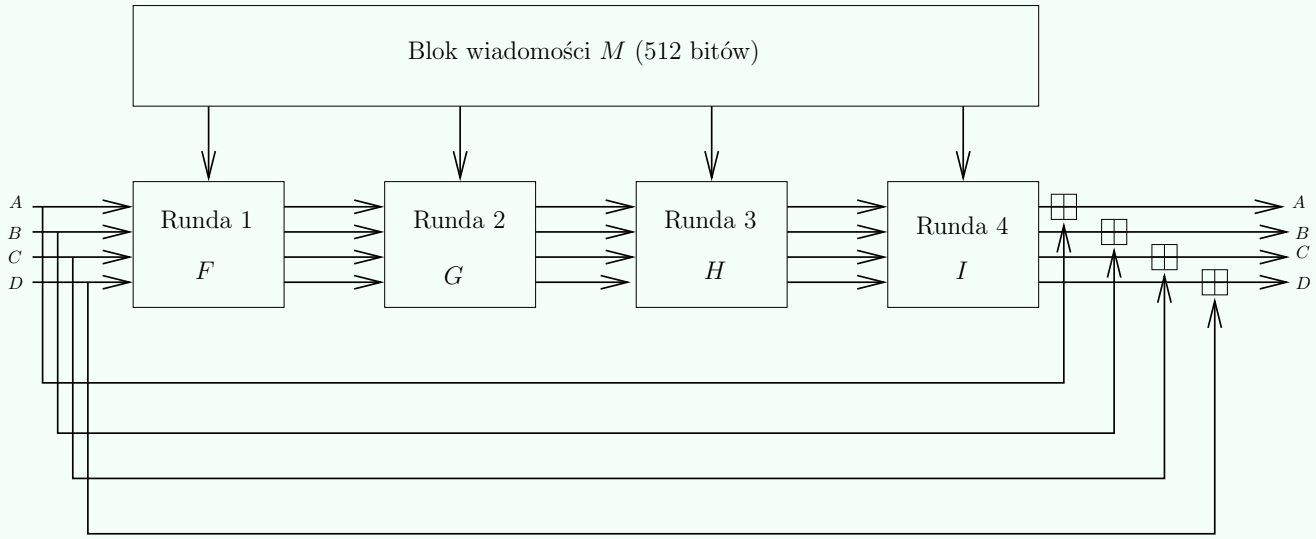
$$
\begin{array}{llll}
[abcd\ \ 5\ 4\ 33] & [dabc\ \ 8\ 11\ 34] & [cdab\ 11\ 16\ 35] & [bcda\ 14\ 23\ 36] \\
[abcd\ \ 1\ 4\ 37] & [dabc\ \ 4\ 11\ 38] & [cdab\ \ 7\ 16\ 39] & [bcda\ 10\ 23\ 40] \\
[abcd\ 13\ 4\ 41] & [dabc\ \ 0\ 11\ 42] & [cdab\ \ 3\ 16\ 43] & [bcda\ \ 6\ 23\ 44] \\
[abcd\ \ 9\ 4\ 45] & [dabc\ 12\ 11\ 46] & [cdab\ 15\ 16\ 47] & [bcda\ \ 2\ 23\ 48]
\end{array}
$$

⋆ **Runda 4**

Niech $[abcd\ j\ s\ k]$ oznacza operację
$a = b + ((a + I(b,c,d) + X_j + T_k) \hookleftarrow s)$,
wtedy 16 operacji tej rundy to:

$$
\begin{array}{llll}
[abcd\ \ 0\ 6\ 49] & [dabc\ \ 7\ 10\ 50] & [cdab\ 14\ 15\ 51] & [bcda\ \ 5\ 21\ 52] \\
[abcd\ 12\ 6\ 53] & [dabc\ \ 3\ 10\ 54] & [cdab\ 10\ 15\ 55] & [bcda\ \ 1\ 21\ 56] \\
[abcd\ \ 8\ 6\ 57] & [dabc\ 15\ 10\ 58] & [cdab\ \ 6\ 15\ 59] & [bcda\ 13\ 21\ 60] \\
[abcd\ \ 4\ 6\ 61] & [dabc\ 11\ 10\ 62] & [cdab\ \ 2\ 15\ 63] & [bcda\ \ 9\ 21\ 64]
\end{array}
$$

⋆ Po tych 4 rundach wartości a, b, c, d są dodawane do wartości A, B, C, D i algorytm przechodzi do następnego bloku M.



Rysunek 5: Główna pętla algorytmu MD5

- **Krok 5**

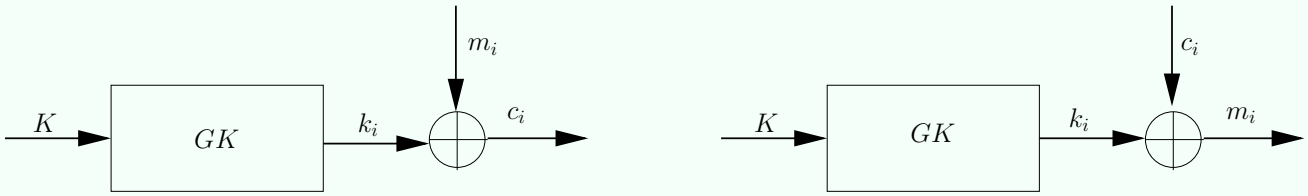
  Po przejściu wszystkich 512-bitowych bloków wiadomości M algorytm łączy rejestry a, b, c, d dając 128 bitową liczbę będącą wartością funkcji hashującej.

SHA (Secure Hash Algorithm)

Algorytm opracowany przez NIST przy udziale NSA opublikowany w 1993. Nowsza wersja SHA-1 opublikowana w 1995 r. Idea tego algorytmu oparta jest na MD4 i MD5. Wartość funkcji hashującej to liczba 160 bitowa i w związku z tym algorytm wymaga 5 rejestrów zamiast 4. Używa też w nieco inny sposób nieliniowych funkcji transformujących, dokonuje dodatkowych operacji na poszczególnych słowach wiadomości, w każdej rundzie wykonuje 20 operacji zamiast 16. Zasada działania jest jednak bardzo podobna do MD5. Ogólnie uważa się, że jest bezpieczniejszy niż MD5 ze względu na „dłuższą" wartość funkcji hashującej i pewne ulepszenia samego algorytmu.

Szyfrowanie strumieniowe i generatory ciągów pseudolosowych

- **Synchroniczne szyfrowanie strumieniowe**
  Ciąg bitów klucza generowany jest niezależnie od szyfrowanej wiadomości i kryptogramu.
  - ⋆ Musi być zachowana synchronizacja pomiędzy nadawcą i odbiorcą.
  - ⋆ Zmiana bitu kryptogramu (przekłamanie) nie wpływa na możliwość deszyfrowania pozostałych bitów.
  - ⋆ Dodanie lub usunięcie bitu powoduje utratę synchronizacji.
  - ⋆ Istnieje możliwość zmiany wybranych bitów kryptogramu, a co za tym idzie zmiany deszyfrowanej wiadomości.

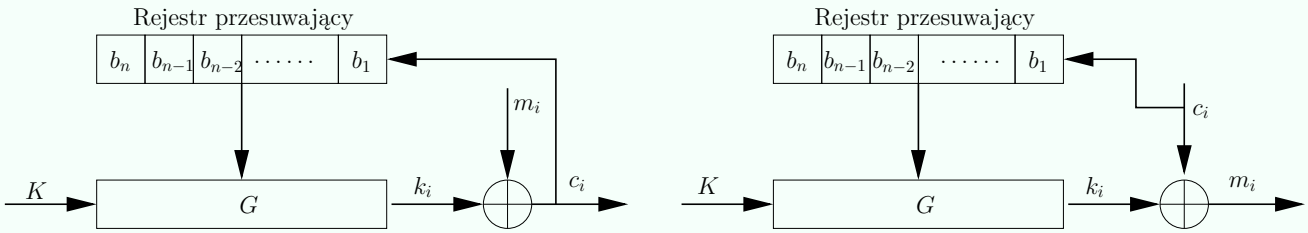


(a) Szyfrowanie (b) Deszyfrowanie

Rysunek 6: Model synchronicznego szyfrowania strumieniowego z dodawaniem modulo 2 ($\oplus$), GK jest generatorem ciągu bitów klucza, zaś K jest kluczem inicjalizującym generator

Tekst jawny szyfrowany jest bit po bicie (one-time pad). Losowo generowane bity $k_1, k_2, \ldots, k_i$ stanowią bity klucza, które są dodawane modulo 2 (operacja XOR) do bitów wiadomości $m_1, m_2, \ldots, m_i$ w sposób ciągły dając kolejne bity kryptogramu $c_1, c_2, \ldots, c_i$, gdzie $c_i = m_i \oplus k_i$

- **Samosynchronizujące (asynchroniczne) szyfrowanie strumieniowe**
  - ⋆ CFB (Cipher Feedback) z blokiem jednobitowym
  - ⋆ Utrata lub dodanie bitu w kryptogramie powoduje utratę tylko kawałka wiadomości — samosynchronizacja.
  - ⋆ Ograniczona propagacja błędów.
  - ⋆ Zmiana bitu kryptogramu powoduje, że kilka innych bitów będzie deszyfrowanych błędnie — łatwiej wykryć taką zmianę.
  - ⋆ Jednak na skutek samosynchronizacji wykrycie zmian w kryptogramie jest trudniejsze (jeśli zmiany dotyczą tylko części kryptogramu, to dalsza część jest deszyfrowana poprawnie).



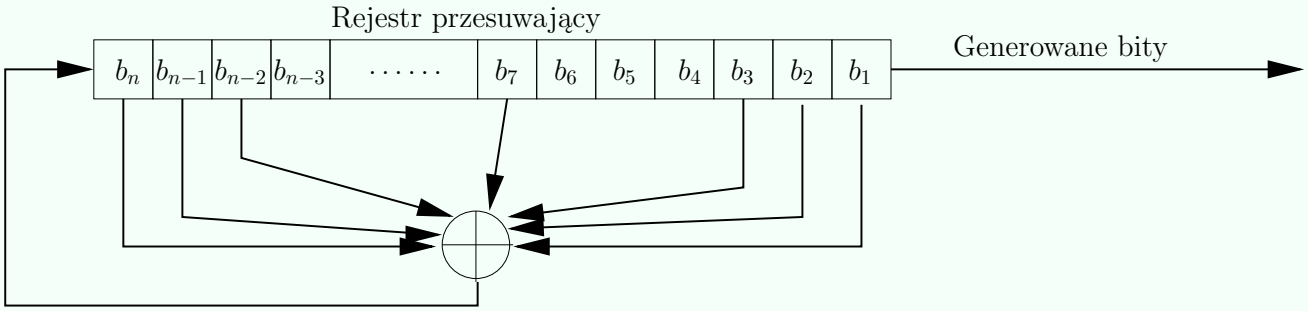
(a) Szyfrowanie (b) Deszyfrowanie

Rysunek 7: Model samosynchronizującego szyfrowania strumieniowego

- **Generatory ciągów pseudolosowych**
  Do generowania klucza potrzebny jest generator losowego ciągu bitów. Generowanie prawdziwie losowego ciągu jest trudne, więc zwykle stosuje się ciągi pseudolosowe. Ciągi pseudolosowe to ciągi, które spełniają statystyczne własności ciągów losowych, ale generowane są w sposób

deterministyczny: generator startujący z takiego samego stanu początkowego generuje taki sam ciąg bitów. Z tego względu ciągi pseudolosowe używane w kryptografii muszą spełniać warunki znacznie ostrzejsze niż np. ciągi pseudolosowe używane w symulacjach.

- **LFSR — Linear Feedback Shift Register** (Rejestr przesuwający z liniowym sprzężeniem zwrotnym)



Rysunek 8: Generowanie ciągu bitów za pomocą LFSR

⋆ LFSR posiada rejestr przesuwający o długości n bitów, który na początku zawiera losowe bity.

⋆ Niektóre bity rejestru są poddawane operacji XOR ($\oplus$) i wynik zastępuje najstarszy bit rejestru, jednocześnie pozostałe bity przesuwane są o jedną pozycję w prawo i najmłodszy bit staje się kolejnym bitem generowanego ciągu.

Przykład:

Weźmy rejestr 4-bitowy, którego pierwszy i czwarty bit są poddawane operacji XOR i niech początkowo rejestr zawiera same jedynki. Wtedy otrzymujemy następujące stany rejestru:

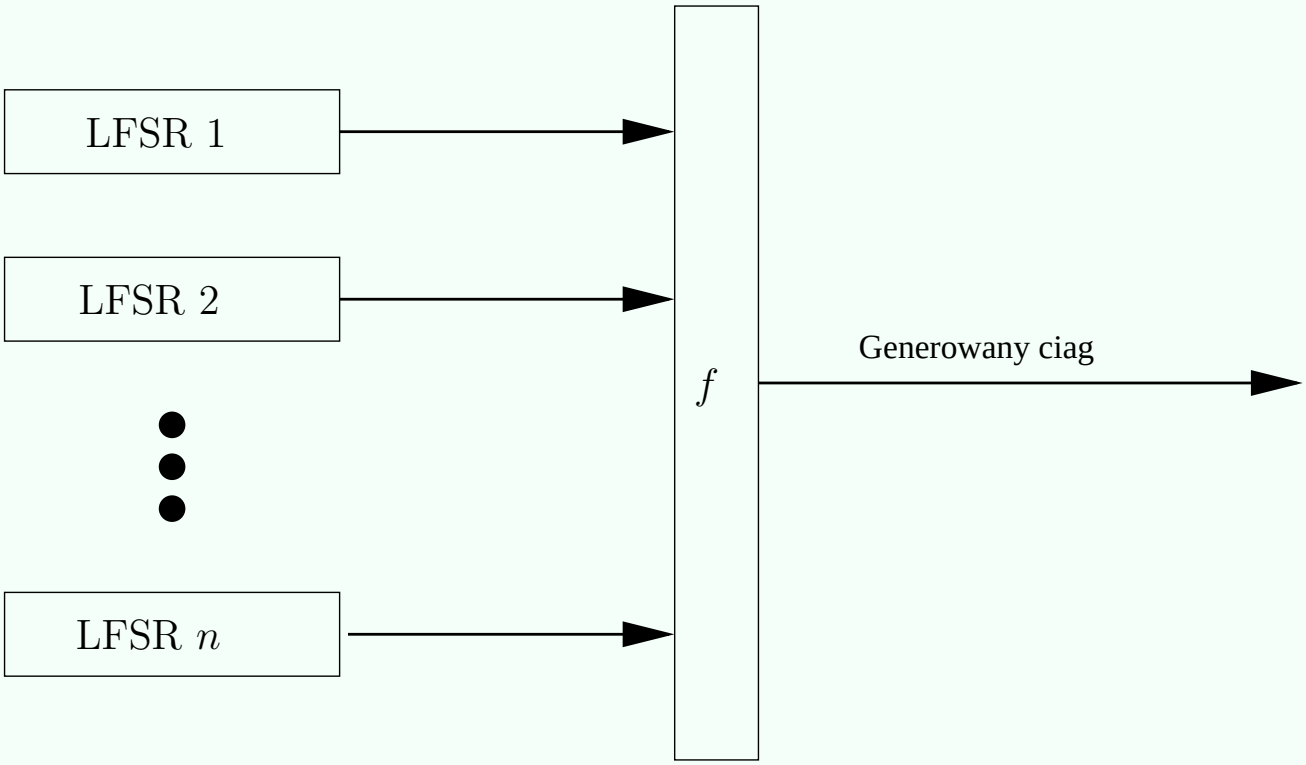
1 1 1 1
0 1 1 1
1 0 1 1
0 1 0 1
1 0 1 0
1 1 0 1
0 1 1 0
0 0 1 1
1 0 0 1
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
1 1 0 0
1 1 1 0

Generowany ciąg to najmłodsze (prawe) bity kolejnych stanów rejestru, czyli:

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

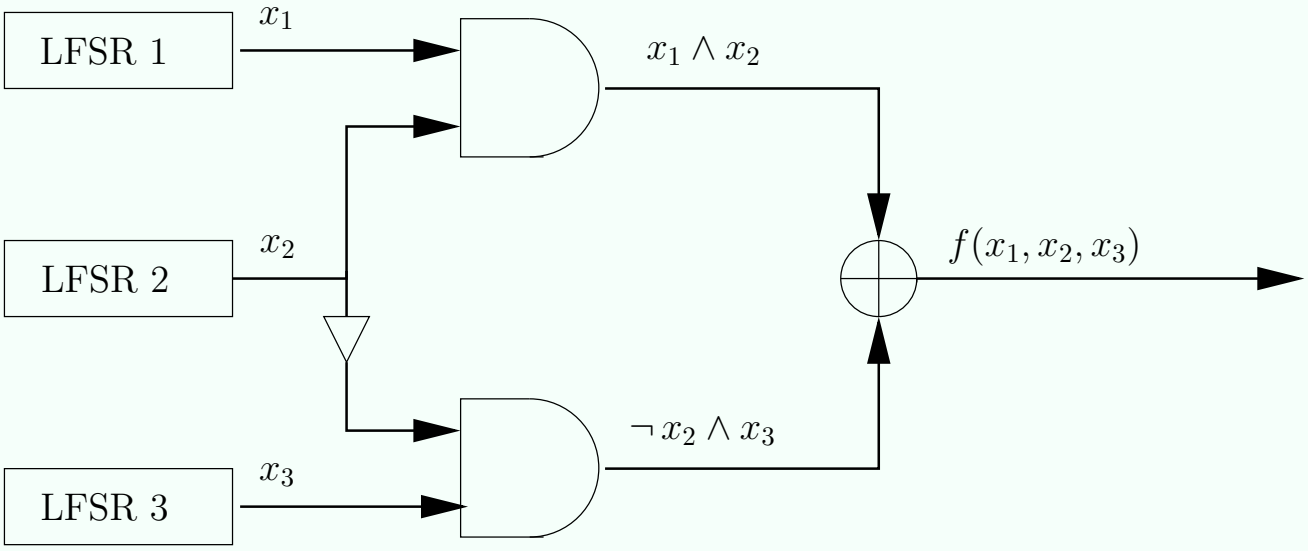
Ponieważ n-bitowy rejestr może znaleźć się w jednym z $2^n - 1$ stanów, więc teoretycznie może on generować ciąg o długości $2^n - 1$ bitów. Potem ciąg się powtarza. (Wykluczamy ciąg samych zer, który daje niekończący się ciąg zer)

- LFSR ma słabą wartość kryptograficzną gdyż znajomość $2n$ kolejnych bitów ciągu pozwala na znalezienie wartości generowanych od tego miejsca.
- LFSR działa jednak bardzo szybko, zwłaszcza jeśli jest to układ hardware'owy, i stąd jest on bardzo atrakcyjny w praktycznych zastosowaniach. Można konstruować bardziej skomplikowane układy zawierające kilka LFSR i nieliniową funkcję f przekształcającą bity generowane przez poszczególne LFSR.



Rysunek 9: Układ złożony z wielu LFSR

Przykład: Generator Geffe



Rysunek 10: Generator Geffe: $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \oplus (\neg x_2 \wedge x_3)$

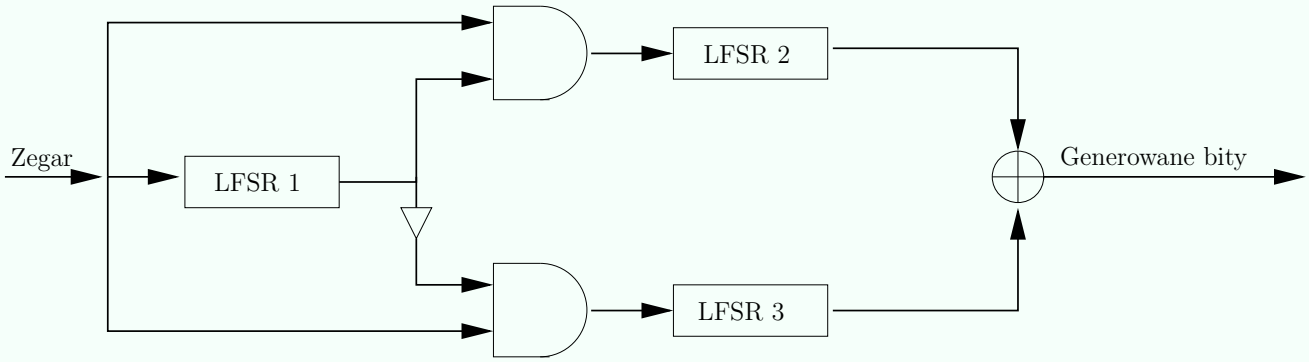
⋆ Generator Geffe ma słabe własności kryptograficzne ze względu na korelacje pomiędzy generowanymi bitami i

bitami LFSR 1 lub LFSR 2

Niech $y(t) = f(x_1(t), x_2(t), x_3(t))$, wtedy

$P(y(t) = x_1(t)) = P(x_2(t) = 1) + P(x_2(t) = 0) \cdot P(x_3(t) = x_1(t)) = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$, i podobnie dla $x_3(t)$.

- **Generatory sterowane zegarem**
  Generator o zmiennym kroku (alternating step generator)



Rysunek 11: Generator o zmiennym kroku

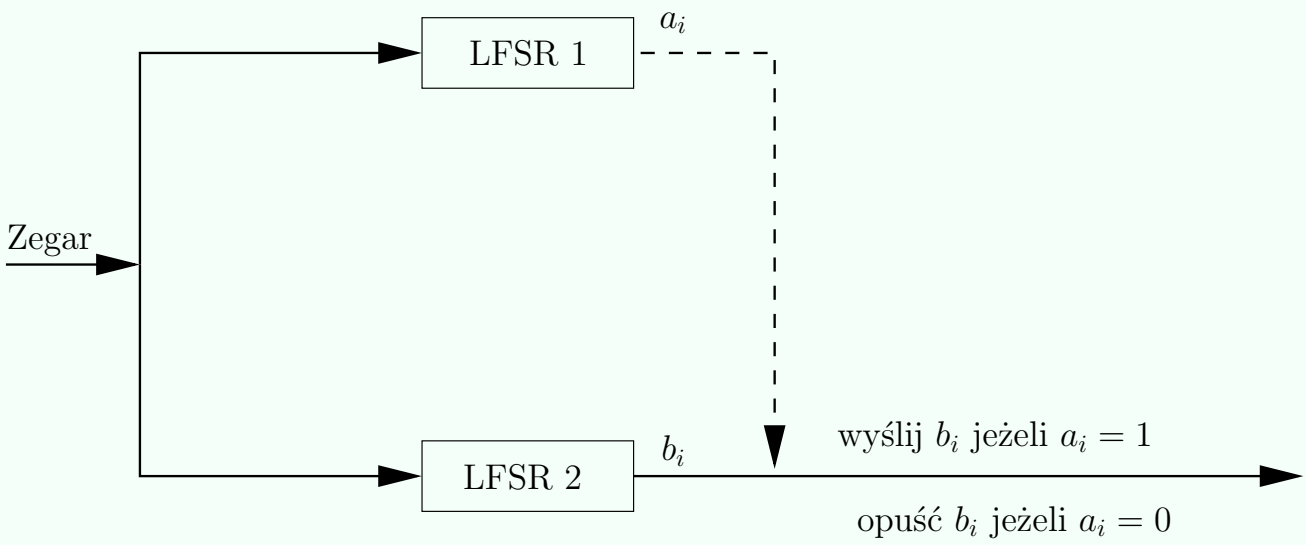
⋆ LFSR 1 jest przesuwany w każdym takcie zegara.

⋆ Jeśli na wyjściu LFSR 1 jest 1 to LFSR 2 jest przesuwany; LFSR 3 nie jest przesuwany (poprzedni bit jest powtarzany).

⋆ Jeśli na wyjściu LFSR 1 jest 0 to LFSR 3 jest przesuwany; LFSR 2 nie jest przesuwany (poprzedni bit jest powtarzany).

⋆ Wyjściowe bity LFSR 2 i LFSR 3 są dodawane modulo 2 ($\oplus$) dając kolejny bit generowanego ciągu.

Shrinking generator



Rysunek 12: Shrinking generator

- **Generatory, których bezpieczeństwo oparte jest na trudnościach obliczeniowych**

  Generator Blum-Micali

  W generatorze tym wykorzystuje się trudność w obliczaniu logarytmu dyskretnego. Wybieramy dwie liczby pierwsze a i p oraz liczbę x_0 (zarodek), a następnie obliczamy

  $$x_{i+1} = a^{x_i} \mod p \quad \text{dla} \quad i = 1, 2, 3, \ldots$$

  Pseudolosowy ciąg bitów tworzymy w następujący sposób:

  $$k_i = \begin{cases} 1 & \text{jeżeli} \quad x_i < (p-1)/2 \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

  Generator RSA

  Generator oparty na trudności z faktoryzacją liczb. Wybieramy dwie liczby pierwsze p i q ($N = pq$) oraz liczbę e względnie pierwszą z $(p-1)(q-1)$. Wybieramy losową

liczbę (zarodek) x_0 mniejszą od N, a następnie obliczamy

$$x_{i+1} \quad = \quad x_i^e \pmod N$$

generowanym bitem jest najmłodszy bit x_i

Generator Blum-Blum-Shub — BBS

Znajdujemy dwie duże liczby pierwsze p i q, takie, że $p \equiv 3 \pmod 4$ oraz $q \equiv 3 \pmod 4$; $N = pq$. Wybieramy losową liczbę x względnie pierwszą z N, a następnie obliczamy

$$x_0 \quad = \quad x^2 \pmod N$$

x_0 stanowi zarodek dla generatora. Teraz liczymy

$$x_{i+1} \quad = \quad x_i^2 \pmod N$$

Generowanym bitem k_i jest najmłodszy bit x_{i+1}.

- **Generator RC 4**

  Generator RC 4 został opracowany przez Rona Rivesta w 1987 r. Przez kilka lat był to algorytm tajny. W 1994 r. ktoś w Internecie opublikował program realizujący ten algorytm. Od tego czasu algorytm nie stanowi tajemnicy. Algorytm ten pracuje w trybie **OFB (Output Feedback)**. Ciąg generowany przez RC 4 jest losowym ciągiem bajtów.

  ⋆ Algorytm używa dwóch wskaźników i, j przyjmujących wartości $0, 1, 2, \ldots, 255$ oraz S-boksu z wartościami $S_0, S_1, \ldots, S_{255}$, które tworzą permutację liczb $0, 1, \ldots, 255$.

  ⋆ Inicjalizacja: Na początku $i = j = 0$, $S_l = l$ dla $l = 0, 1, \ldots, 255$, kolejna 256-bajtowa tablica wypełniana jest bajtami klucza, przy czym klucz jest

używany wielokrotnie, aż do wypełnienia całej tablicy $K_0, K_1, \ldots, K_{255}$. Następnie wykonujemy:
for $i = 0$ to 255:
$j = (j + S_i + K_i) \pmod{256}$
zamień S_i z S_j

- ⋆ Generowanie kolejnego bajtu:
  $i = i + 1 \pmod{256}$
  $j = j + S_i \pmod{256}$
  zamień S_i z S_j
  $l = S_i + S_j \pmod{256}$
  $K = S_l$
- ⋆ Otrzymany bajt K jest dodawany modulo 2 (XOR) z kolejnym bajtem wiadomości dając kolejny bajt kryptogramu (przy deszyfrowaniu role tekstu jawnego i kryptogramu się zamieniają).

Algorytm RC 4 jest używany w wielu programach komercyjnych.

Podpis cyfrowy

Przypomnijmy:
System kryptograficzny z kluczem publicznym może być wykorzystany do podpisywania dokumentów cyfrowych.

1. Alicja szyfruje dokument używając swojego klucza prywatnego, podpisując w ten sposób dokument
2. Alicja przesyła tak podpisany dokument do Bolka
3. Bolek deszyfruje dokument używając klucza publicznego Alicji, weryfikując w ten sposób podpis Alicji

- **Uwagi:**
  - ⋆ podpis jest prawdziwy; Bolek weryfikuje go deszyfrując kryptogram kluczem publicznym Alicji
  - ⋆ podpis nie może być sfałszowany; tylko Alicja zna jej klucz prywatny
  - ⋆ podpis nie może być przeniesiony do innego dokumentu
  - ⋆ podpisany dokument nie może być zmieniony; zmieniony dokument nie da się rozszyfrować kluczem publicznym Alicji
  - ⋆ podpis jest niezaprzeczalny;
  - ⋆ wadą tego sposobu podpisywania dokumentów jest jednak to, że podpis jest conajmniej tak długi jak sam dokument

Podpis z wykorzystaniem jednokierunkowej funkcji hashującej

1. Alicja używa funkcji hashującej do dokumentu, który ma podpisać, otrzymując skrót („odcisk palca”) dokumentu

2. Alicja podpisuje skrót dokumentu szyfrując go swoim kluczem prywatnym
3. Alicja przesyła Bolkowi dokument i podpisany skrót
4. Bolek używa tej samej funkcji hashującej do otrzymania skrótu dokumentu, a następnie deszyfruje podpisany skrót używając klucza publicznego Alicji; jeśli zdeszyfrowany skrót zgadza się z otrzymanym przez niego to podpis jest prawdziwy
   - ⋆ podpis jest znacznie krótszy od dokumentu
   - ⋆ można sprawdzić istnienie podpisu bez oglądania samego dokumentu

Schemat ElGamala podpisu cyfrowego

- **Generowanie kluczy**
  - ⋆ Alicja wybiera dużą liczbę pierwszą p oraz liczbę $g \in \mathbb{Z}_p$ (generator grupy multiplikatywnej $\mathbb{Z}_p^*$)
  - ⋆ Alicja wybiera liczbę losową $0 < a < p-1$ oraz oblicza $b \equiv g^a \pmod{p}$
  - ⋆ Kluczem publicznym Alicji są liczby $\{b, g, p\}$ zaś kluczem prywatnym liczby $\{a, g, p\}$
- **Podpisywanie**
  - ⋆ Alicja wybiera liczbę losową k (tajną), taką, że $0 < k < p-1$ oraz $NWD(k, p-1) = 1$
  - ⋆ Alicja oblicza

    $r = g^k \pmod{p}$,

    $k^{-1} \pmod{p-1}$,

    $s = k^{-1}\left[H(M) - ar\right] \pmod{p-1}$.
  - ⋆ Podpisem Alicji dla wiadomości M jest para liczb $\{r, s\}$

- **Weryfikacja**
  - ⋆ Bolek aby stwierdzić prawdziwość podpisu Alicji pobiera klucz publiczny Alicji $\{b, g, p\}$
  - ⋆ Bolek sprawdza czy $0 < r < p$, jeśli nie, podpis nie jest prawdziwy
  - ⋆ Bolek oblicza
    $x_1 = b^r r^s \pmod p$,
    $x_2 = g^{H(M)} \pmod p$.
  - ⋆ Bolek akceptuje podpis jeśli $x_1 = x_2$
- **Uzasadnienie**
  Ponieważ $s \equiv k^{-1}\left[H(M) - ar\right] \pmod{p-1}$, to mnożąc stronami przez k mamy $ks \equiv H(M) - ar \pmod{p-1}$ i po przekształceniu $H(M) \equiv ar + ks \pmod{p-1}$, a co za tym idzie $g^{H(M)} \equiv g^{ar+ks} \equiv (g^a)^r r^s \equiv b^r r^s \pmod p$. Tak więc $x_1 = x_2$.

DSA — Digital Signature Algorithm

Algorytm podpisu cyfrowego zatwierdzony w 1994 r. przez NIST jako standard podpisu cyfrowego w USA (Digital Signature Standard — DSS). Wykorzystuje funkcję hashującą SHA-1.

- **Generacja klucza**
  - ⋆ Alicja wybiera liczbę pierwszą q o długości 160 bitów
  - ⋆ Alicja wybiera liczbę pierwszą p o długości $512 \le l \le 1024$, przy czym $64|l$, taką, że $q|p-1$
  - ⋆ Alicja wybiera element $g \in \mathbb{Z}_p$ i oblicza
    $b = g^{(p-1)/q} \pmod p$;
    jeśli $b = 1$ to wybiera inne g.

⋆ Alicja wybiera liczbę losową a, $0 < a < q$, i oblicza $c = b^a \pmod p$

⋆ Kluczem publicznym Alicji jest zbiór liczb $\{b, c, p, q\}$

- **Podpisywanie**

⋆ Alicja wybiera tajną liczbą losową k, $0 < k < q$,

⋆ Alicja oblicza

$$r = (b^k \pmod p) \pmod q,$$

$$k^{-1} \pmod q,$$

$$s = k^{-1}\,[H(M) + ar] \pmod q.$$

⋆ Podpisem Alicji dla wiadomości M jest para liczb $\{r, s\}$

- **Weryfikacja**

⋆ Bolek pobiera klucz publiczny Alicji $\{b, c, p, q\}$

⋆ Bolek sprawdza czy $0 < r < q$ i $0 < s < q$, jeśli nie, to podpis jest fałszywy

⋆ Bolek oblicza

$$H(M) \text{ i } w = s^{-1} \pmod q,$$

$$u_1 = w\,H(M) \pmod q,$$

$$u_2 = rw \pmod q,$$

$$v = (b^{u_1} c^{u_2} \pmod p) \pmod q.$$

⋆ Bolek uznaje podpis za prawdziwy jeśli $v = r$.

- **Uzasadnienie**

Jeśli $\{r, s\}$ jest prawdziwym podpisem Alicji dla wiadomości M, to $H(M) \equiv -ar + ks \pmod q$. Mnożąc stronami przez w i przekształcając otrzymujemy $w\,H(M) + arw \equiv k \pmod q$, co jest równoważne $u_1 + au_2 \equiv k \pmod q$. Podnosząc b do potęgi lewej i prawej strony tej kongruencji otrzymujemy $(b^{u_1+au_2} \pmod p) \pmod q \equiv (b^k \pmod p) \pmod q$ i dalej mamy

$(b^{u_1} c^{u_2} \pmod p) \pmod q \equiv (b^k \pmod p) \pmod q$, a to oznacza $v = r$.

Ślepe podpisy cyfrowe

Zasadniczym założeniem protokołów podpisów cyfrowych jest, że podpisujący dokument wie co podpisuje. Nie należy podpisywać dokumentów wyglądających na losowy ciąg bitów. Od powyższej zasady są jednak odstępstwa. Przypuśćmy, że Bolek jest notariuszem, zaś Alicja chce aby Bolek potwierdził notarialnie istnienie dokumentu, ale nie chce aby ten dokument obejrzał. Mamy wtedy do czynienia z tzw. ślepym podpisem. Wyobraźmy sobie, że Alicja wkłada list do koperty łącznie z kalką, a potem prosi Bolka o złożenie podpisu na zaklejonej kopercie. Po otwarciu koperty na liście będzie kopia podpisu Bolka. Cyfrowo ślepy podpis można zrealizować korzystając np. z algorytmu RSA.

- **Ślepy podpis z użyciem RSA**
  - ⋆ Alicja pobiera klucz publiczny Bolka $\{e, n\}$
  - ⋆ Alicja wybiera liczbę losową k, $0 < k < n$,
  - ⋆ Alicja oblicza $z = M\, k^e \pmod n$ i przesyła z do Bolka
  - ⋆ Bolek oblicza $z^d = (M\, k^e)^d \pmod n$ używając swojego klucza prywatnego $\{d, n\}$ i wynik przesyła Alicji
  - ⋆ Alicja oblicza $s = z^d/k \pmod n$. Ponieważ $z^d \equiv (M\, k^e)^d \equiv M^d k \pmod n$, więc $z^d/k = M^d k/k \equiv M^d \pmod n$, czyli $s = M^d \pmod n$ jest podpisem Bolka na wiadomości M.

Niezaprzeczalne podpisy cyfrowe

Podpis niezaprzeczalny nie może być sprawdzony bez zgody osoby podpisującej. Podpisujący nie może się wyprzeć swojego podpisu, ale może także dowieść, że podpis jest fałszywy (jeśli jest).

- **Niezaprzeczalny podpis oparty na logarytmach dyskretnych**

  Przypuśćmy, że stroną podpisującą dokument jest Alicja.

  ⋆ **Generacja klucza**

  Alicja posiada klucz prywatny $\{a, g, p\}$ oraz klucz publiczny $\{b, g, p\}$ wygenerowany jak w algorytmie ElGamala.

  ⋆ **Podpisywanie**

  Alicja oblicza $z = M^a \pmod p$ i to jest jej podpis dla dokumentu M

  ⋆ **Weryfikacja**

  1. Bolek wybiera dwie liczby losowe r i s mniejsze od p, oblicza $w = z^r b^s \pmod p$ i przesyła Alicji
  2. Alicja oblicza
     $$t = a^{-1} \pmod{p-1}$$
     $$v = w^t \pmod p$$
     i przesyła Bolkowi v
  3. Bolek sprawdza czy $v = M^r g^s \pmod p$

  ⋆ **Uzasadnienie**

  Zauważmy, że $v = w^t = z^{rt} b^{st} = (z^t)^r (b^t)^s = (M^{at})^r (g^{at})^s = M^r g^s \pmod p$

Uwierzytelnianie

Kluczową sprawą dla bezpieczeństwa systemów komputerowych jest zapewnienie dostępu do systemu i zasobów tylko osobom do tego uprawnionym. W systemie musi więc być wbudowany mechanizm sprawdzania, czy użytkownik podający się za Alicję, naprawdę nią jest. Do tego celu służy mechanizm uwierzytelniania lub identyfikacji (tutaj nie rozróżniamy tych pojęć, chociaż czasem się je rozróżnia).

- **Hasła**
  Najczęściej stosowany system identyfikacji to system haseł. Alicja chcąc wejść do sytemu podaje tajne hasło znane tylko jej i systemowi.
  - ⋆ **Hasła w systemie Unix** szyfrowane są programem `crypt`, który stanowi pewną modyfikację DES. Użytkownik wybiera ośmioliterowe hasło. Z każdego bajtu reprezentującego literę hasła wybieranych jest 7 bitów, które w rezultacie tworzą 56 bitowy klucz. Klucz ten służy do szyfrowania 64 bitowego bloku znanego tekstu (zwykle same zera). Wynik podlega kolejnemu szyfrowaniu, i tak 25 razy. Dodatkowo używa się 12 bitów („salt") generowanych przez zegar systemowy w momencie tworzenia hasła. Bity te są wykorzystane w permutacji rozszerzającej DES. Wynik szyfrowania (64 bity) plus „salt" (12 bitów) jest „przepakowany" i zapisywany w postaci 11 znaków ASCII. Hasło przechowywane jest w postaci 13 znaków ASCII, które zawierają dwa znaki „salt" oraz 11 znaków zaszyfrowanego hasła. Dodanie 12 bitów „salt" powoduje, że

liczba możliwości dla wybranego hasła zwiększa się $2^{12} = 4096$ razy.

⋆ W nowszych systemach stosuje się bezpieczniejsze sposoby szyfrowania haseł, np. algorytm MD5

- **PIN — Personal Identification Number**
  Odmianą hasła jest także PIN używany w przypadku kart kredytowych, bankowych, czy tzw. tokenów. Jest to zwykle liczba czterocyfrowa (czasem ośmiocyfrowa), która ma zabezpieczać przed użyciem karty przez osoby niepowołane, np. złodzieja.
- **Protokół challenge-response**
  Idea tego sposobu identyfikacji polega na odpowiedzi Alicji na wyzwanie przesłane przez Bolka, która przekona Bolka, że ma do czynienie rzeczywiście z Alicją.

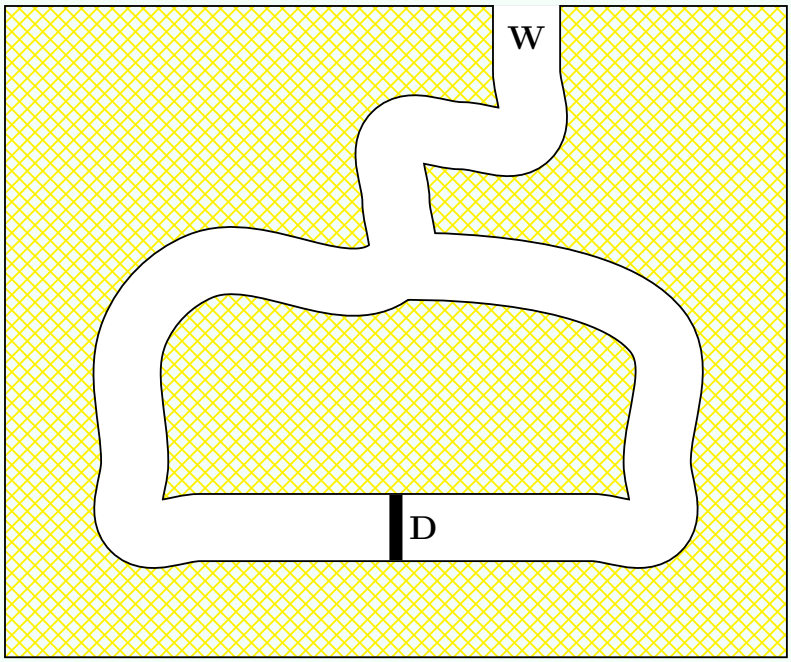
  ⋆ **Protokół challenge-response z tajnym kluczem**

  1. Alicja i Bolek dysponują takim samym tajnym kluczem K (algorytm symetryczny) oraz umówili się jakiej funkcji hashującej H będą używać.
  2. Alicja komunikuje się z Bolkiem przedstawiając się jako Alicja
  3. Bolek generuje liczbę losową r_B i wysyła ją Alicji
  4. Alicja oblicza $H(K, r_B)$ i przesyła wynik Bolkowi
  5. Bolek także oblicza $H(K, r_B)$ i jeśli wynik zgadza się z wynikiem przysłanym przez Alicję to tożsamość Alicji zostaje potwierdzona

⋆ **Protokół challenge-response z kluczem publicznym**

1. Alicja komunikuje się z Bolkiem przedstawiając się jako Alicja
2. Bolek generuje liczbę losową r_B i wysyła ją Alicji
3. Alicja szyfruje liczbę r_B używając swojego klucza prywatnego i kryptogram wysyła do Bolka
4. Bolek deszyfruje kryptogram otrzymany od Alicji używając jej klucza publicznego i jeśli w wyniku otrzyma r_B to tożsamość Alicji jest potwierdzona

- **Dowody z wiedzą zerową**



Rysunek 13: Jaskinia

⋆ Alicja chce przekonać Bolka, że zna pewien sekret, ale nie chce zdradzić samego sekretu. Alicja twierdzi, że potrafi otworzyć drzwi zamykające przejście w jaskini.

⋆ Bolek stoi przy wejściu do jaskini

- ⋆ Alicja wchodzi do jaskini i idzie albo w lewo albo w prawo dochodząc do drzwi zamykających przejście
- ⋆ Bolek dochodzi do rozwidlenia korytarza, rzuca monetą i w zależności od wyniku rzutu krzyczy, nakazując Alicji wyjść albo z lewego korytarza albo z prawego
- ⋆ Alicja wykonuje polecenie Bolka, otwierając drzwi jeśli to konieczne
- ⋆ Doświadczenie takie powtarzają n-krotnie. Jeśli n jest dostatecznie duże, to prawdopodobieństwo tego, że Alicja wykona polecenie Bolka nie potrafiąc otworzyć drzwi jest znikomo małe $(1/2^n)$.

- **Dowód o wiedzy zerowej dla logarytmu dyskretnego**

  Alicja chce przekonać Bolka, że zna wartość logarytmu dyskretnego bez zdradzanie tej wartości. Czyli chce udowodnić, że zna liczbę x, która spełnia zależność $a^x = b \pmod p$, gdzie p jest dużą liczbą pierwszą. Oboje znają p, a, b, natomiast Bolek nie zna x.

  - ⋆ Alicja generuje t liczb losowych $r_1, r_2, \ldots, r_t$ mniejszych od $p-1$
  - ⋆ Alicja oblicza $h_i \equiv a^{r_i} \pmod p$ i przesyła je Bolkowi
  - ⋆ Alicja i Bolek wspólnie rzucają t razy monetą generując w ten sposób t bitów $b_1, b_2, \ldots, b_t$
  - ⋆ Dla wszystkich bitów Alicja oblicza i przesyła Bolkowi następujące liczby

    $$r_i \qquad \text{jeśli } b_i = 0$$
    $$s_i = r_i - r_j \qquad \text{jeśli } b_i = 1$$

    gdzie j jest największą wartością, dla której $b_j = 1$
  - ⋆ Dla wszystkich bitów t Bolek sprawdza czy

$$a^{r_i} \equiv h_i \pmod{p} \qquad \text{dla } b_i = 0$$

$$a^{s_i} \equiv h_i h_j^{-1} \pmod{p} \qquad \text{dla } b_i = 1$$

⋆ Dla każdego i, dla którego $b_i = 1$, Alicja oblicza i wysyła Bolkowi
$z_i = (x - r_i) \pmod{p-1}$

⋆ Bolek sprawdza czy $a^{z_i} \equiv b h_i^{-1} \pmod{p}$

- **Protokół Fiata-Shamira**

Bezpieczeństwo tego protokołu opiera się na trudności obliczeniowej pierwiastków kwadratowych modulo n, gdzie n jest iloczynem dwóch liczb pierwszych. Protokół ten wymaga udziału strony trzeciej, zaufanego arbitra — Trusted Authority (TA)

⋆ TA wybiera dwie liczby pierwsze p i q, oblicza ich iloczyn $n = pq$

⋆ Alicja wybiera losową liczbę względnie pierwszą z n, oblicza liczbę $v = s^2 \pmod{n}$ i rejestruje u TA v jako swój klucz publiczny

⋆ TA udostępnia liczby n i v jako identyfikatory tożsamości Alicji

⋆ Alicja wybiera losowo liczbę r względnie pierwszą z n, oblicza $x = r^2 \pmod{n}$ i wysyła x Bolkowi

⋆ Bolek wysyła Alicji losowy bit b

⋆ Alicja wysyła Bolkowi

$$r \qquad \text{jeśli } b = 0$$

$$y = r \cdot s \pmod{n} \qquad \text{jeśli } b = 1$$

⋆ Bolek sprawdza czy

$$x = r^2 \pmod{n} \qquad \text{jeśli } b = 0$$

$$y^2 = x \cdot v \pmod{n} \qquad \text{jeśli } b = 1$$

Pierwsza równość dowodzi, że Alicja zna pierwiastek kwadratowy z x, druga natomiast dowodzi, że Alicja zna s. Protokół ten powtarza się t razy, wtedy prawdopodobieństwo oszustwa przez Alicję wynosi $1/2^t$.

- **Protokół Schnorra**

  Protokół ten opiera się na problemie logarytmu dyskretnego. Protokół wykorzystuje certyfikaty wydawane przez TA. W etapie wstępnym należy wybrać liczbę pierwszą p oraz drugą liczbę pierwszą q taką, że $q|p-1$. Liczby te powinny być dostatecznie duże (np. p długości 1024 bity a $q > 160$ bitów. Wybieramy także liczbę $b = g^{(p-1)/q}$, gdzie g jest generatorem $\mathbb{Z}_p$. Każda ze stron otrzymuje liczby $\{p, q, b\}$ oraz klucz publiczny pozwalający weryfikować podpisy TA. Ponadto należy wybrać parametr t ($t \geq 40, \quad 2^t < q$), który określa poziom bezpieczeństwa.

  ⋆ TA ustala tożsamość Alicji w konwencjonalny sposób i przydziela jej identyfikator I_A

  ⋆ Alicja wybiera losowo tajną liczbę a oraz oblicza $v = b^a \pmod p$ i rejestruje v u TA

  ⋆ TA generuje podpis cyfrowy $S(I_A, v)$ oraz wydaje Alicji certyfikat $C = (I_A, v, S(I_A, v))$ wiążący I_A z v

  ⋆ Alicja wybiera liczbę losową $r < q$ i oblicza $x = b^r \pmod p$

  ⋆ Alicja przesyła Bolkowi certyfikat C oraz liczbę x

  ⋆ Bolek sprawdza klucz publiczny Alicji sprawdzając podpis TA na certyfikacie

  ⋆ Bolek wybiera losowo liczbę k ($1 \leq k \leq 2^t$) i wysyła ją Alicji (*challenge*)

- ⋆ Alicja sprawdza $1 \leq k \leq 2^t$ i wysyła Bolkowi $y = ak + r \pmod{q}$ (*response*)
- ⋆ Bolek oblicza $z = b^y v^k \pmod{p}$ i jeśli $z = x$ uznaje, że tożsamość Alicji jest potwierdzona.

Zarządzanie kluczami

- **Generowanie kluczy**
  Do wytwarzania kluczy najlepiej nadają się generatory ciągów losowych.
  Przykład: standard ANSI X9.17
  (Financial Institution Key Management)
  $EDE_{K_1,K_2}(X)$ oznacza szyfrowanie 3-DES kluczami K_1, K_2 liczby X. Niech V_0 będzie tajną liczbą 64 bitową, a T znacznikiem czasu, wtedy klucz losowy R_i generuje się w następujący sposób:
  $R_i = EDE_{K_1,K_2}(EDE_{K_1,K_2}(T_i) \oplus V_i)$
  $V_{i+1} = EDE_{K_1,K_2}(EDE_{K_1,K_2}(T_i) \oplus R_i)$
- **Przesyłanie kluczy**
  Jeśli Alicja i Bolek zamierzają posługiwać się symetrycznym algorytmem kryptograficznym, to potrzebują tego samego klucza. Alicja może wygenerować taki klucz używając generatora ciągów losowych, ale pozostaje problem jak w bezpieczny sposób przekazać ten klucz Bolkowi.
- **Przechowywanie kluczy**
  Najbezpieczniejszym sposobem przechowywania klucza jest zapamiętanie go przez Alicję. Niestety sposób ten ma tę wadę, że Alicja może zapomnieć taki klucz. Klucze mogą być przechowywane w pamięci ROM. Klucz taki może być rozdzielony na połowy, z których jedna jest przechowywana w terminalu a druga w pamięci ROM. W wielu sytuacjach istnieje konieczność przechowywania kopii zapasowych klucza. Do tego celu wykorzystuje się np. karty inteligentne. Klucze na ogół mają strukturę hierarchiczną

⋆ **master keys**
klucze znajdujące się najwyżej w hierarchii, takie klucze nigdy nie są zmieniane. Taki klucz jest zwykle tylko zapamiętywany przez użytkownika lub zapisany w urządzeniu kryptograficznym. Może on być częściowo zapisany na kartach inteligentnych a częściowo zapamiętywany przez użytkownika. Master key służy do zabezpieczania innych kluczy.

⋆ **klucze do szyfrowania kluczy** (*keys-encrypting keys*)
klucze wykorzystywane w protokołach do uzgadniania (przesyłania) kluczy.

⋆ **klucze do szyfrowania danych** (*data keys*)
są to zwykle klucze o krótkim czasie ważności (np. klucze sesyjne) służące do szyfrowania danych

Uzgadnianie kluczy

- **Algorytm Diffiego-Hellmana**
  - ⋆ Alicja i Bolek uzgadniają dwie liczby: dużą liczbę pierwszą p oraz liczbę g, która jest generatorem $\mathbb{Z}_p$
  - ⋆ Alicja wybiera losowo dużą liczbę całkowitą $a < p-1$ i przesyła Bolkowi $x = g^a \pmod p$
  - ⋆ Bolek wybiera losowo dużą liczbę całkowitą $b < p-1$ i wysyła Alicji $y = g^b \pmod p$
  - ⋆ Alicja oblicza $K = y^a \pmod p$
  - ⋆ Bolek oblicza $K = x^b \pmod p$

Uzasadnienie:
$K = y^a = (g^b)^a = g^{ab} \pmod p$
$K = x^b = (g^a)^b = g^{ab} \pmod p$

- **Algorytm ElGamala**
  - ⋆ Bolek wybiera liczbę pierwszą p oraz generator g w $\mathbb{Z}_p$, a następnie wybiera losowo liczbę $0 < b < p-1$, oblicza $g^b \pmod p$ oraz ogłasza swój klucz publiczny $\{p, g, g^b\}$
  - ⋆ Alicja pobiera klucz publiczny Bolka, wybiera losowo liczbę $0 < a < p-1$ i wysyła Bolkowi $g^a \pmod p$ obliczając jednocześnie klucz $K = (g^b)^a \pmod p$
  - ⋆ Bolek oblicza ten sam klucz $K = (g^a)^b \pmod p$

- **Station-to Station protocol (STS)**

  W tym protokole przyjmuje się, że Alicja ma certyfikat klucza publicznego Bolka i na odwrót Bolek ma certyfikat klucza publicznego Alicji. Niech E oznacza symetryczny algorytm szyfrujący, $S_A(M)$ oznacza podpis Alicji pod wiadomością M: $S_A(M) = (H(M))^{d_A} \pmod{n_A}$ (RSA na wartości funkcji hashującej). Każda ze stron wybiera odpowiednią liczbę pierwszą p oraz generator g w $\mathbb{Z}_p$. Każda ze stron wybiera klucze RSA do podpisu.
  - ⋆ Alicja wybiera losowo liczbę a i wysyła Bolkowi $g^a \pmod p$
  - ⋆ Bolek wybiera losowo liczbę b i oblicza klucz $K = (g^a)^b \pmod p$
  - ⋆ Bolek podpisuje konkatenację g^b, g^a, szyfruje podpis kluczem K i wysyła Alicji $g^b \pmod p$ oraz $E_K(S_B(g^b, g^a))$
  - ⋆ Alicja oblicza klucz $K = (g^b)^a \pmod p$, deszyfruje otrzymane dane oraz używa klucza publicznego Bolka do sprawdzenia podpisu pod wartością funkcji hashu-

jącej z konkatenacji obu eksponensów. Jeśli wartość funkcji hashującej zgadza się z otrzymaną przez nią wartością, Alicja akceptuje klucz K i wysyła Bolkowi
$E_K(S_A(g^a, g^b))$

⋆ Bolek deszyfruje otrzymaną wiadomość, weryfikuje podpis Alicji i jeśli wynik jest pozytywny, akceptuje klucz K

- **Uzgadnianie klucza z szyfrowaniem**
  Zakładamy, że Alicja i Bolek (użytkownik i komputer) znają hasło P.
  ⋆ Alicja generuje losowo parę kluczy (publiczny i prywatny), szyfruje klucz publiczny K' używając algorytmu symetrycznego wykorzystującego hasło P jako klucz i wysyła do Bolka
  $E_P(K')$
  ⋆ Bolek deszyfruje wiadomość otrzymaną od Alicji otrzymując klucz K', następnie generuje klucz sesyjny K, szyfruje ten klucz kluczem publicznym K' oraz kluczem P i wysyła Alicji
  $E_P(E_{K'}(K))$
  ⋆ Alicja deszyfruje wiadomość otrzymaną od Bolka uzyskując klucz K. Następnie generuje ciąg losowy r_A, szyfruje go kluczem K i wysyła do Bolka
  $E_K(r_A)$
  ⋆ Bolek deszyfruje wiadomość otrzymaną od Alicji otrzymując ciąg r_A, generuje własny ciąg losowy r_B, szyfruje obydwa ciągi kluczem K i wysyła Alicji
  $E_K(r_A, r_B)$

⋆ Alicja deszyfruje wiadomość uzyskując r_A i r_B. Jeśli r_A jest prawidłowe, szyfruje r_B i wysyła do Bolka $E_K(r_B)$

⋆ Bolek deszyfruje wiadomość i jeśli r_B jest prawidłowe klucz K zostaje zaakceptowany jako klucz sesyjny.

Istnieją różne implementacje tego protokołu, z wykorzystaniem algorytmu RSA czy ElGamala. Protokół ten wzmacnia bezpieczeństwo przy uzgadnianiu klucza sesyjnego.

- **ssh** *secure shell*
  Protokół umożliwiający bezpieczne logowanie się do komputerów w sieci stworzony przez Tatu Ylönena, który skutecznie zapobiega takim metodom ataku jak IP Spoofing i DNS Spoofing, czy też podsłuchiwaniu haseł lub transmitowanych danych. Obecnie rozwijana jest wersja OpenSSH, na licencji GNU.

  ⋆ przy instalacji programu generowana jest para kluczy algorytmu asymetrycznego przynależna danemu komputerowi — klucz publiczny komputera — *host key*

  ⋆ przy uruchomieniu demona sshd generowana jest następna para kluczy — *server keys*. Publiczny jest dostępny do czytania, a prywatny jest przechowywany w pamięci komputera (nie jest zapisywany na dysku). Co godzinę para tych kluczy jest zmieniana.

  ⋆ każdy użytkownik generuje kolejną parę kluczy (*ssh-keygen*), które służą do uwierzytelniania użytkownika. Klucz prywatny jest szyfrowany.

  ⋆ Kiedy Alicja próbuje zalogować się na komputer Bolka,

to komputer ten wysyła jej swoje dwa klucze publiczne *host key* i *server key*. Komputer Alicji sprawdza czy *host key* zgadza się z kluczem zapisanym w lokalnym pliku *known-hosts*.

- ⋆ Jeśli wszystko się zgadza to Alicja generuje losowy klucz sesji i szyfruje go po kolei obydwoma kluczami publicznymi komputera Bolka i tak uzyskany kryptogram przesyła do Bolka.
- ⋆ Bolek potrzebuje dwóch kluczy prywatnych do odszyfrowania klucza sesyjnego
- ⋆ Bolek przesyła Alicji liczbę losową r_B zaszyfrowaną kluczem publicznym Alicji.
- ⋆ Alicja deszyfruje otrzymany kryptogram i odsyła Bolkowi $H(r_B)$ udowadniając mu swoją tożsamość. W ten sposób zostaje ustanowiony bezpieczny kanał komunikacyjny.

Kryptoanaliza

- **Podstawowe rodzaje ataku**

  ⋆ Atak typu *ciphertext-only* — znane są tylko kryptogramy — chcemy znaleźć klucz lub tekst jawny

  ⋆ Atak typu *known plaintext* — znane są pewne pary kryptogram-tekst jawny — szukamy klucza

  ⋆ Atak typu *chosen plaintext* — znane są kryptogramy dla dowolnie wybranego tekstu jawnego

  ⋆ Atak typu *chosen ciphertext* — atakujący ma możliwość uzyskania tekstu jawnego dla dowolnie wybranego tekstu tajnego

  ⋆ Atak typu *adaptive chosen plaintext* — atakujący ma możliwość wielokrotnego szyfrowania tekstu jawnego, który jest modyfikowany w zależności od uzyskanych wcześniej wyników

- **Kryptoanaliza różnicowa**

  Eli Biham i Adi Shamir w 1990 r. znaleźli metodę ataku na DES przy wybranym tekście jawnym, która okazała się bardziej efektywna niż przeszukiwanie wszystkich możliwości (atak brutalny). W kryptoanalizie różnicowej porównuje się pary kryptogramów, które powstały w wyniku zaszyfrowania par tekstów jawnych o ustalonych różnicach. Przypuśćmy, że mamy dwa bloki o równej długości X i X' i wprowadzimy różnicę obu bloków $\Delta X = X \oplus X'$ (operacja dodawania modulo dwa odpowiadających sobie bitów obu bloków — operacja XOR — w wyniku

dostajemy jedynki na tych miejscach gdzie ciągi bitów się różnią). Rozważmy parę wejściową X i X' tekstów jawnych i uwzględniając fakt, że operacje liniowe nie zmieniają różnicy ΔX, co dotyczy także operacji XOR z kluczem K_i, która daje:
$Y = X \oplus K_i$, $Y' = X' \oplus K_i$,
$\Delta Y = (X \oplus K_i) \oplus (X' \oplus K_i) = X \oplus X' = \Delta X$,
a więc przed wejściem do S-boksa różnice się zachowują i nie zależą od wartości klucza. Nieliniowym elementem DES'a są S-boksy i różnica ΔY zostanie przez S-boks na ogół zmieniona. Jeśli wynikiem działania S-boksu będzie $Z = S(Y)$, to różnica $\Delta Z = S(Y) \oplus S(Y')$ będzie zależała od klucza K_i i okazuje się, że tylko niektóre wartości dla ΔZ są możliwe, a to oznacza, że możliwe jest uzyskanie informacji o kluczu K_i. W tablicy 9 podane są liczby możliwych ΔZ dla danej różnicy ΔX (różnice ΔX i ΔZ podane są układzie szesnastkowym o czym przypomina wskaźnik x). Liczba znajdująca się w tablicy podzielona przez 64 określa prawdopodobieństwo wystąpienia danej różnicy ΔZ. W tabeli znajdujemy wiele zer, a to oznacza, że takie różnice nie mogą wystąpić. Prawdopodobieństwa wystąpienia poszczególnych różnic znacznie się różnią i ten fakt jest wykorzystywany w kryptoanalizie różnicowej. Np. dla $\Delta X = 1_x$ istnieją tylko cztery pary które dają różnicę $\Delta Z = F_x$. Takie pary można wcześniej wyznaczyć i w tym przypadku są to pary:
$\{1E_x, 1F_x\}$, $\{1F_x, 1E_x\}$, $\{2A_x, 2B_x\}$, $\{2B_x, 2A_x\}$.

ΔX	ΔZ															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
2_x	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
3_x	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
4_x	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
5_x	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
6_x	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
7_x	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
8_x	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
9_x	10	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
A_x	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
B_x	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
C_x	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
D_x	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
E_x	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
F_x	2	0	2	4	4	6	4	2	4	8	2	2	2	6	8	8
10_x	0	0	0	0	0	0	2	14	0	6	6	12	4	6	8	6
11_x	6	8	2	4	6	4	8	6	4	0	6	6	0	4	0	0
12_x	0	8	4	2	6	6	4	6	6	4	2	6	6	0	4	0
13_x	2	4	4	6	2	0	4	6	2	0	6	8	4	6	4	6
14_x	0	8	8	0	10	0	4	2	8	2	2	4	4	8	4	0
15_x	0	4	6	4	2	2	4	10	6	2	0	10	0	4	6	4
16_x	0	8	10	8	0	2	2	6	10	2	0	2	0	6	2	6
17_x	4	4	6	0	10	6	0	2	4	4	4	6	6	6	2	0
18_x	0	6	6	0	8	4	2	2	2	4	6	8	6	6	2	2
19_x	2	6	2	4	0	8	4	6	10	4	0	4	2	8	4	0
$1A_x$	0	6	4	0	4	6	6	6	6	2	2	0	4	4	6	8
$1B_x$	4	4	2	4	10	6	6	4	6	2	2	4	2	2	4	2
$1C_x$	0	10	10	6	6	0	0	12	6	4	0	0	2	4	4	0
$1D_x$	4	2	4	0	8	0	0	2	10	0	2	6	6	6	14	0
$1E_x$	0	2	6	0	14	2	0	0	6	4	10	8	2	2	6	2
$1F_x$	2	4	10	6	2	2	2	8	6	8	0	0	0	4	6	4

Tablica 9: Tablica rozkładu różnic w S-boksie S_1

ΔX	ΔZ															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
20_x	0	0	0	10	0	12	8	2	0	6	4	4	4	2	0	12
21_x	0	4	2	4	4	8	10	0	4	4	10	0	4	0	2	8
22_x	10	4	6	2	2	8	2	2	2	2	6	0	4	0	4	10
23_x	0	4	4	8	0	2	6	0	6	6	2	10	2	4	0	10
24_x	12	0	0	2	2	2	2	0	14	14	2	0	2	6	2	4
25_x	6	4	4	12	4	4	4	10	2	2	2	0	4	2	2	2
26_x	0	0	4	10	10	10	2	4	0	4	6	4	4	4	2	0
27_x	10	4	2	0	2	4	2	0	4	8	0	4	8	8	4	4
28_x	12	2	2	8	2	6	12	0	0	2	6	0	4	0	6	2
29_x	4	2	2	10	0	2	4	0	0	14	10	2	4	6	0	4
$2A_x$	4	2	4	6	0	2	8	2	2	14	2	6	2	6	2	2
$2B_x$	12	2	2	2	4	6	6	2	0	2	6	2	6	0	8	4
$2C_x$	4	2	2	4	0	2	10	4	2	2	4	8	8	4	2	6
$2D_x$	6	2	6	2	8	4	4	4	2	4	6	0	8	2	0	6
$2E_x$	6	6	2	2	0	2	4	6	4	0	6	2	12	2	6	4
$2F_x$	2	2	2	2	2	6	8	8	2	4	4	6	8	2	4	2
30_x	0	4	6	0	12	6	2	2	8	2	4	4	6	2	2	4
31_x	4	8	2	10	2	2	2	2	6	0	0	2	2	4	10	8
32_x	4	2	6	4	4	2	2	4	6	6	4	8	2	2	8	0
33_x	4	4	6	2	10	8	4	2	4	0	2	2	4	6	2	4
34_x	0	8	16	6	2	0	0	12	6	0	0	0	0	8	0	6
35_x	2	2	4	0	8	0	0	0	14	4	6	8	0	2	14	0
36_x	2	6	2	2	8	0	2	2	4	2	6	8	6	4	10	0
37_x	2	2	12	4	2	4	4	10	4	4	2	6	0	2	2	4
38_x	0	6	2	2	2	0	2	2	4	6	4	4	4	6	10	10
39_x	6	2	2	4	12	6	4	8	4	0	2	4	2	4	4	0
$3A_x$	6	4	6	4	6	8	0	6	2	2	6	2	2	6	4	0
$3B_x$	2	6	4	0	0	2	4	6	4	6	8	6	4	4	6	2
$3C_x$	0	10	4	0	12	0	4	2	6	0	4	12	4	4	2	0
$3D_x$	0	8	6	2	2	6	0	8	4	4	0	4	0	12	4	4
$3E_x$	4	8	2	2	2	4	4	14	4	2	0	2	0	8	4	4
$3F_x$	4	8	4	2	4	0	2	4	4	2	4	8	8	6	2	2

Tablica 9: Tablica rozkładu różnic w S-boksie S_1

c.d.

Znajomość takich par oraz prawdopodobieństw ich wystąpienia pozwala przy wykorzystaniu ataku typu *chosen plaintext* uzyskać informację o bitach klucza. Można w ten sposób znacznie ograniczyć przestrzeń możliwych kluczy. Prawdopodobnie twórcy DES'a zdawali sobie sprawę z możliwości kryptoanalizy różnicowej, chociaż pojawiła się ona później niż sam DES. Liczba rund DES'a została wybrana w taki sposób, że nawet korzystanie z kryptoanalizy różnicowej wymaga dużych nakładów (mocy obliczeniowych) dla złamania szyfru.

- **Kryptoanaliza liniowa**

  Inną metodą kryptoanalizy jest kryptoanaliza liniowa zaproponowana przez Mitsuru Matsui w 1993 r. Idea kryptoanalizy liniowej polega na opisie działania urządzenia szyfrującego poprzez aproksymację liniową. Mimo, że S-boksy DES'a są elementami nieliniowymi, to mogą być one aproksymowane formułami liniowymi. Oznacza to, że zależności liniowe aproksymujące działanie S-boksu są spełnione z prawdopodobieństwem różnym niż 1/2. Jeśli np. wiemy, że pomiędzy bitami klucza k_i, tekstu jawnego m_i oraz kryptogramu c_i zachodzą z prawdopodobieństwem 90% zależności

  $m_{15} \oplus k_2 \oplus m_7 \oplus k_6 = c_2 \oplus m_5 \oplus c_7$

  $m_8 \oplus k_2 \oplus k_6 = c_5 \oplus c_6$,

  to znając m_i i c_i możemy z takim samym prawdopodobieństwem wyznaczyć $k_2 \oplus k_6$. Oczywiście tego typu zależności należy najpierw znaleźć.

  Dla DES'a przy ataku typu *known plaintext* kryptoanaliza liniowa wymaga średnio 2^{43} par tekst jawny-kryptogram

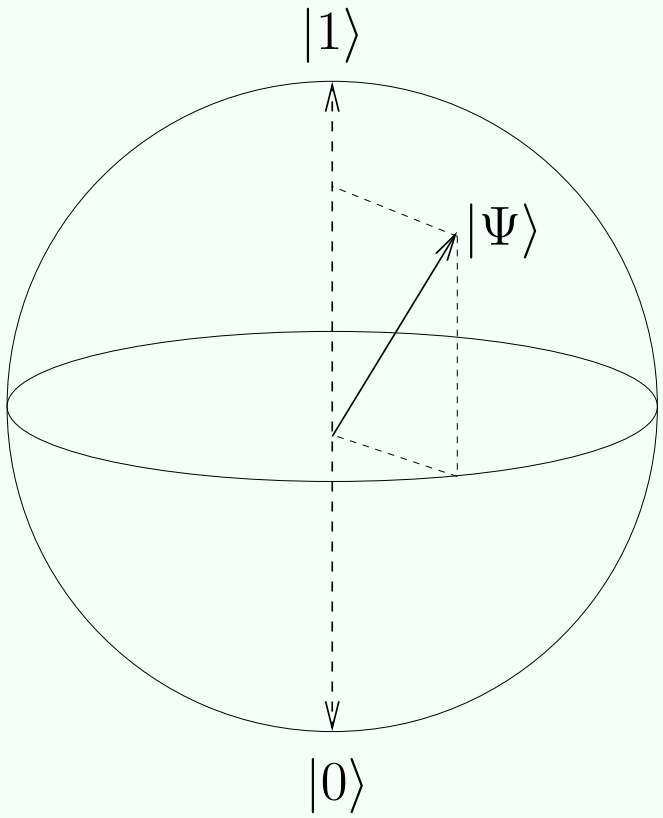
do znalezienia klucza. Matsui w 1994 r. potrzebował 50 dni aby na 12 komputerach HP 9735 obliczyć klucz DES'a!

Algorytmy kwantowe

- **Bit kwantowy — kubit (*qubit*)**
  Klasyczny bit może przyjmować dwie wartości $\{0, 1\}$. Układ kwantowy, który ma dwa możliwe stany $\{|0\rangle, |1\rangle\}$ może się znajdować w każdym z nich, ale także w stanie będącym superpozycją stanów bazowych

$$|\Psi\rangle = a|0\rangle + b|1\rangle$$

  i taki stan nazywamy kubitem. Oznacza to, że z prawdopodobieństwem $p_0 = |a|^2$ układ znajduje się w stanie $|0\rangle$ i z prawdopodobieństwem $p_1 = |b|^2$ w stanie $|1\rangle$, oczywiście $p_0 + p_1 = 1$. Stan układu kwantowego możemy przedstawić jako wektor na sferze Blocha



Rysunek 14: Kubit na sferze Blocha

- **Twierdzenie o nieklonowaniu**

  Nie istnieje transformacja unitarna U taka, że

  $$U|\Psi\rangle|0\rangle \quad = \quad |\Psi\rangle|\Psi\rangle$$

  dla dowolnego $|\Psi\rangle$.

  Dowód:

  Przypuśćmy, że istnieje U takie, że

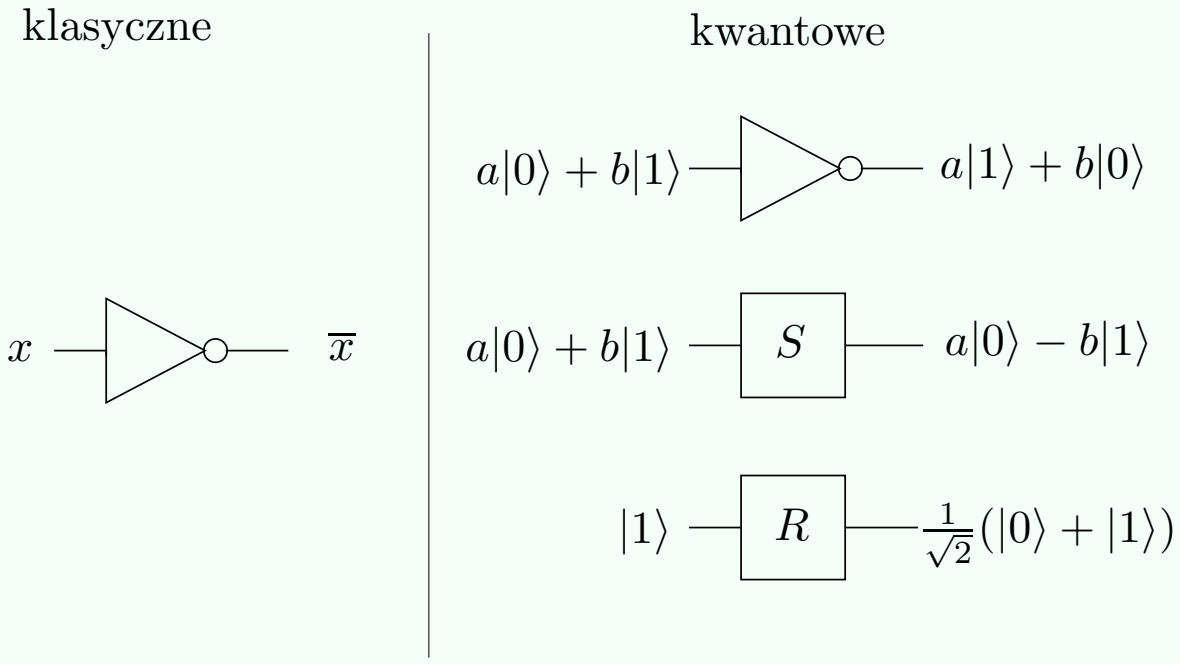
  $$\begin{aligned} U|\Psi\rangle|0\rangle &= |\Psi\rangle|\Psi\rangle \\ U|\Phi\rangle|0\rangle &= |\Phi\rangle|\Phi\rangle \end{aligned}$$

  dla dowolnych $|\Psi\rangle$ i $|\Phi\rangle$. Transformacja U reprezentowała by maszynę klonującą, gdyby taka istniała. Z unitarności U wynika jednak, że

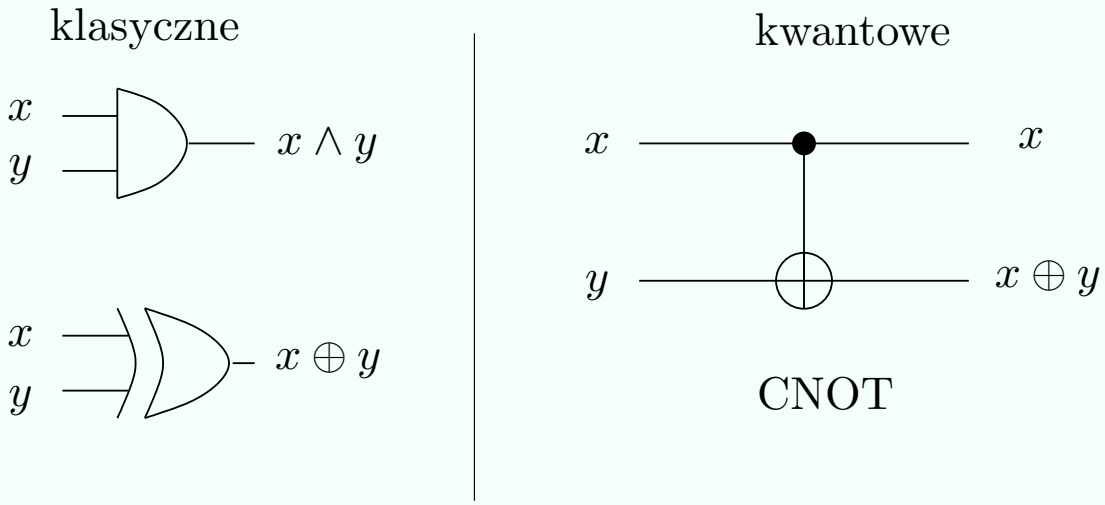
  $$\begin{aligned} \langle\Psi|\langle 0|U^\dagger U|\Phi\rangle|0\rangle &= \langle\Psi|\Phi\rangle\langle\Psi|\Phi\rangle \\ \langle\Psi|\Phi\rangle\langle 0|0\rangle &= \langle\Psi|\Phi\rangle\langle\Psi|\Phi\rangle \end{aligned}$$

  co nie jest prawdziwe dla dowolnych $|\Psi\rangle$ i $|\Phi\rangle$, natomiast może zachodzić dla stanów ortogonalnych $\langle\Psi|\Phi\rangle = \{0, 1\}$. Stany ortogonalne (klasyczne bity) mogą być kopiowane, natomiast dowolne stany kwantowe nie.

- **Bramki logiczne**



Rysunek 15: Jednobitowe bramki logiczne



Rysunek 16: Dwubitowe bramki logiczne

W bazie stanów $\{|0\rangle, |1\rangle\}$, mamy

$$|0\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |1\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Wtedy operacje na stanach kwantowych mają reprezentację macierzową, i tak na przykład

$$U_{\rm NOT} \quad = \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$U_{\rm NOT}|0\rangle \quad = \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \equiv |1\rangle$$

Operacja przesunięcia fazy, która nie zmienia stanu $|0\rangle$ zaś stan $|1\rangle$ zmienia na $-|1\rangle$, ma postać

$$U_S \quad = \quad \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Operacja Hadamarda, czasem nazywana pierwiastkiem kwadratowym z NOT ($\sqrt{\rm NOT}$), ma postać

$$H \quad = \quad \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Istnieje nieskończenie wiele bramek kwantowych generowanych przez rotacje o kąt θ

$$U_R(\theta) \quad = \quad \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

oraz przesunięcia faz

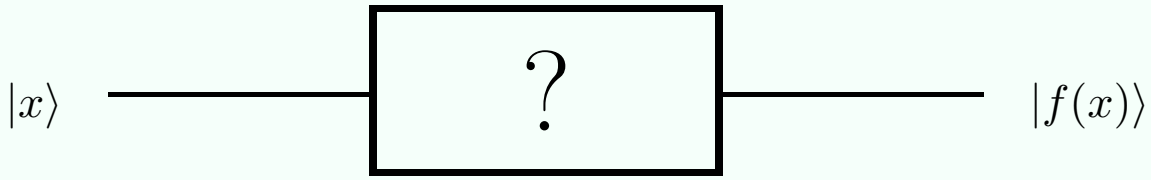
$$U_P(\varphi_1, \varphi_2) \quad = \quad \begin{bmatrix} e^{i\varphi_1} & 0 \\ 0 & e^{i\varphi_2} \end{bmatrix}$$

Operacja CNOT (kontrolowane NOT) na dwóch kubitach ma postać

$$U_{\mathrm{CN}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- **Problem Deutscha**

  Przypuśćmy, że mamy kwantową czarną skrzynkę obliczającą funkcję $f(x)$, tzn. wykonującą transformację unitarną na dwóch kubitach przedstawioną poniżej

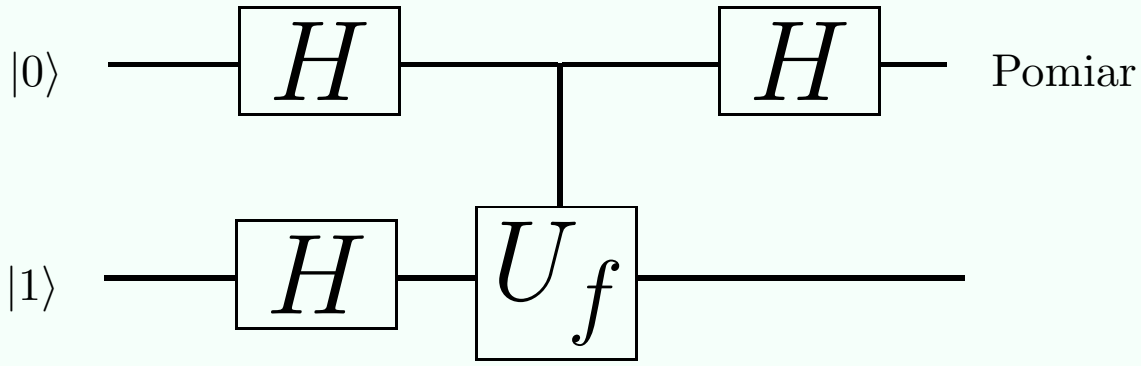


$$f : \{0,1\} \to \{0,1\}$$

$$U_f : |x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$$

Pytanie:
czy po jednym przebiegu kwantowego komputera możemy stwierdzić, że $f(0) = f(1)$?

Weźmy następujący kwantowy obwód



gdzie H jest kwantową bramką Hadamarda.

Działanie obwodu

$$\begin{aligned} H:|0\rangle &\rightarrow \frac{1}{\sqrt{2}}(|0\rangle+|1\rangle) \\ |1\rangle &\rightarrow \frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) \end{aligned}$$

$$\begin{aligned} |0\rangle|1\rangle &\rightarrow \frac{1}{2}(|0\rangle+|1\rangle)(|0\rangle-|1\rangle) \\ &\rightarrow \frac{1}{2}\left((-1)^{f(0)}|0\rangle+(-1)^{f(1)}|1\rangle\right)(|0\rangle-|1\rangle) \\ &\rightarrow \frac{1}{2}\Big[\left((-1)^{f(0)}+(-1)^{f(1)}\right)|0\rangle \\ &\quad +\left((-1)^{f(0)}-(-1)^{f(1)}\right)|1\rangle\Big]\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) \end{aligned}$$

$$|m\rangle = \begin{cases} |0\rangle & \text{dla} \quad f(0)=f(1) \\ |1\rangle & \text{dla} \quad f(0)\neq f(1) \end{cases}$$

- **Kwantowy paralelizm**

  Przypuśćmy, że mamy funkcję działającą na N bitów o 2^N możliwych wartościach. Aby obliczyć tablicę funkcji $f(x)$ musielibyśmy policzyć wartość funkcji 2^N razy (dla $N = 100 \sim 10^{30}$)! Na komputerze kwantowym działającym zgodnie z

$$U_f : |x\rangle|0\rangle \quad \rightarrow \quad |x\rangle|f(x)\rangle$$

  możemy wybrać stan początkowy (kwantowy rejestr) w postaci

$$|\psi_0\rangle \quad = \quad \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right]^N = \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle$$

  i obliczając $f(x)$ tylko raz otrzymujemy stan

$$|\psi\rangle \quad = \quad \frac{1}{2^{N/2}} \sum_{x=0}^{2^N-1} |x\rangle|f(x)\rangle$$

  Na przykład, dla $N = 2$

$$|\psi_0\rangle \quad = \quad \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

$$\begin{aligned} |00\rangle \quad &\rightarrow \quad |0\rangle \\ |01\rangle \quad &\rightarrow \quad |1\rangle \\ |10\rangle \quad &\rightarrow \quad |2\rangle \\ |11\rangle \quad &\rightarrow \quad |3\rangle \end{aligned}$$

$$|\psi_0\rangle \quad = \quad \frac{1}{2}(|0\rangle + |1\rangle + |2\rangle + |3\rangle)$$

Algorytm Shora

- **Etap 1**
  Przygotowujemy rejestr A komputera kwantowego w superpozycji wszystkich możliwych stanów
- **Etap 2**
  Liczba, którą chcemy sfaktoryzować jest N, $N = 15$ Wybieramy liczbę losową X, $1 < X < N-1$, $X = 2$. Wykonujemy operację $B = (X^A \mod N)$ np. dla $X = 2$ mamy wyniki przedstawione w tabelce
- **Etap 3**
  Obliczamy $P = X^{f/2} - 1$; $f = 4$ i sprawdzamy czy P jest dzielnikiem N w naszym przypadku
  $P = 2^{4/2} - 1 = 3$,
  $P = 2^{4/2} + 1 = 5$;

Hurra !!!

$15/3 = 5$

$15/5 = 3$

Rejestr A	Rejestr B
0	1
1	2
2	4
3	8
4	1
5	2
6	4
7	8
8	1
9	2
10	4
11	8
12	1
13	2
14	4
15	8

Kwantowa transformata Fouriera

$$QFT : |x\rangle \quad \rightarrow \quad \frac{1}{q}\sum_{y=0}^{q-1} \mathrm{e}^{2\pi i x y/q}|y\rangle$$

gdzie $q = 2^N$

- **Okres funkcji**

  Przygotowujemy stan

$$|\psi_0\rangle \quad = \quad \frac{1}{\sqrt{q}}\sum_{x=0}^{q-1}|x\rangle|f(x)\rangle$$

  Mierzymy drugi rejestr dostając $|f(x_0)\rangle$, co powoduje, że pierwszy rejestr staje się superpozycją wszystkich stanów, które dają wartość $f(x_0)$ (funkcja jest okresowa z okresem r)

$$\frac{1}{\sqrt{a}}\sum_{j=0}^{a-1}|x_0 + jr\rangle\,, \quad a-1 < \frac{q}{r} < a+1$$

  Stosujemy QTF

$$\frac{1}{\sqrt{qa}}\sum_{y=0}^{q-1}\mathrm{e}^{2\pi i x_0 y}\sum_{j=0}^{a-1}\mathrm{e}^{2\pi i j r y/q}|y\rangle$$

$$\mathrm{Prob}(y) = \frac{a}{q}\left|\frac{1}{a}\sum_{j=0}^{a-1}\mathrm{e}^{2\pi i j r y/q}\right|^2$$

  Jeśli q/r jest całkowite ($q/r = a$), to

$$\mathrm{Prob}(y) = \frac{1}{a}\left|\frac{1}{a}\sum_{j=0}^{a-1}\mathrm{e}^{2\pi i j y/q}\right|^2 \quad = \quad \begin{cases} \frac{1}{r} & y = a * integer \\ 0 & otherwise \end{cases}$$

Kryptografia kwantowa

- **Protokół BB84** (Bennett, Brassard, 1984)
  Wybierzmy dwie ortonormalne bazy dla pomiaru polaryzacji fotonu:
  - ⋆ **Baza prosta** (+)
    Tworzą ją dwa stany o polaryzacji poziomej oraz pionowej $\{|\rightarrow\rangle, |\uparrow\rangle\}$
  - ⋆ **Baza ukośna** (×)
    Tworzą ją dwa stany o polaryzacji 45° oraz polaryzacji 135° $\{|\nearrow\rangle, |\nwarrow\rangle\}$
  - ⋆ Zachodzą następujące relacje

$$
\begin{aligned}
|\nearrow\rangle &= \frac{1}{\sqrt{2}}\left(|\rightarrow\rangle + |\uparrow\rangle\right) \\
|\nwarrow\rangle &= -\frac{1}{\sqrt{2}}\left(|\rightarrow\rangle - |\uparrow\rangle\right)
\end{aligned}
$$

$$
\begin{aligned}
|\rightarrow\rangle &= \frac{1}{\sqrt{2}}\left(|\nearrow\rangle - |\nwarrow\rangle\right) \\
|\uparrow\rangle &= \frac{1}{\sqrt{2}}\left(|\nearrow\rangle + |\nwarrow\rangle\right)
\end{aligned}
$$

  Wynika z nich, że pomiar polaryzacji fotonu "ukośnego" w bazie prostej daje z prawdopodobieństwem 1/2 stan $|\rightarrow\rangle$ lub $|\uparrow\rangle$, co oznacza, że pomiar taki nie daje żadnych informacji o polaryzacji fotonu. To samo możemy powiedzieć o pomiarze fotonu "prostego" w bazie ukośnej. Polaryzacja prosta i polaryzacja ukośna to dwie wielkości fizyczne, które zgodnie z prawami mechaniki

kwantowej **nie są współmierzalne**. Obowiązuje tutaj **zasada nieoznaczoności Heisenberga**.

⋆ **Alfabety kwantowe**

Mając dwie bazy możemy stworzyć dwa kwantowe alfabety przypisując dwóm ortogonalnym stanom bazy wartości binarne 0 i 1.

$$\begin{aligned} |\rightarrow\rangle &\equiv 0 \\ |\uparrow\rangle &\equiv 1 \end{aligned}$$

$$\begin{aligned} |\nearrow\rangle &\equiv 0 \\ |\nwarrow\rangle &\equiv 1 \end{aligned}$$

⋆ **Etapy BB84**

1. Alicja wybiera losowo jedną z dwóch baz i jedną z dwóch ortogonalnych polaryzacji w wybranej bazie, co oznacza wybór jednej z czterech możliwych polaryzacji i wysyła do Bolka foton o takiej polaryzacji. Zgodnie z przyjętymi alfabetami oznacza to odpowiadający wybranym polaryzacjom ciąg bitów.
2. Bolek losowo wybiera bazę prostą lub ukośną i wykonuje pomiar polaryzacji fotonu, który otrzymał od Alicji
3. Bolek notuje wyniki pomiarów zachowując je w tajemnicy
4. Bolek publicznie informuje Alicję jakiej bazy używał, zaś Alicja informuje go czy była to baza właściwa czy nie.
5. Alicja i Bolek przechowują wyniki, dla których Bolek użył właściwej bazy. Przypisując tym wynikom

wartości binarne 0 i 1 zgodnie z przyjętymi alfabetami oboje otrzymują taki sam ciąg zer i jedynek (losowy), który może służyć jako klucz kryptograficzny.

Przykład:

Alicja	+	×	+	×	×	+	×	×	×	+	+	+	×
	↑	↗	→	↗	↖	→	↖	↖	↗	↑	↑	↑	↗
	1	0	0	0	1	0	1	1	0	1	1	1	0
Bolek	+	+	×	+	×	×	×	+	×	+	+	×	×
	↑	→	↗	↑	↖	↗	↖	→	↗	↑	↑	↖	↗
	1	0	0	1	1	0	1	0	0	1	1	1	0
A/B	√				√		√		√	√	√		√
klucz	1				1		1		0	1	1		0

Porównując bity wysłane przez Alicję z bitami zarejestrowanymi przez Bolka możemy podzielić bity zarejestrowane przez Bolka na trzy kategorie: bity pewne (średnio 50 %) — te dla których Bolek wybrał prawidłową bazę i które mogą być traktowane jako klucz kryptograficzny; bity prawidłowe pomimo złego wyboru bazy (średnio 25 %); bity nieprawidłowe (średnio 25 %). Prawdopodobieństwo wyboru jednej z dwóch możliwych baz wynosi 1/2, prawdopodobieństwo zarejestrowania prawidłowej polaryzacji przy prawidłowym wyborze bazy wynosi 1, prawdopodobieństwo pomiaru

prawidłowej polaryzacji przy nieprawidłowo wybranej bazie wynosi 1/2, zatem prawdopodobieństwo tego, że zarejestrowany bit będzie prawidłowy (taki sam jak bit wysłany) jest równe $\frac{1}{2}\cdot 1+\frac{1}{2}\cdot\frac{1}{2}=\frac{3}{4}$. Prawdopodobieństwo zarejestrowania bitu nieprawidłowego (błędnego) wynosi więc $1-\frac{3}{4}=\frac{1}{4}$.

Alicja	+	×	+	×	×	+	×	×	×	+	+	+	×
	↑	↗	→	↗	↖	→	↖	↖	↗	↑	↑	↑	↗
	1	0	0	0	1	0	1	1	0	1	1	1	0
Bolek	+	+	×	+	×	×	×	+	×	+	+	×	×
	↑	→	↗	↑	↖	↗	↖	→	↗	↑	↑	↖	↗
	1	0	0	1	1	0	1	0	0	1	1	1	0
pewne	1				1		1		0	1	1		0
dobre		0	0			0						1	
złe				1				0					

Jeśli Ewa podsłuchuje stosując strategię tzw. nieprzeźroczystego podsłuchu, to wybiera losowo bazę prostą lub ukośną, dokonuje pomiaru polaryzacji w tej bazie i następnie przesyła do Bolka foton o takiej polaryzacji jaką zmierzyła. Dokonywane przez Ewę pomiary muszą wprowadzić błędy, które Alicja i Bolek mogą wykryć przy uzgadnianiu klucza. W podanym niżej przykładzie ostatni bit został zmieniony. Średnio 25 % bitów klucza zostanie zmienionych. Takie błędy Alicja i Bolek mogą

wykryć wybierając losowo pewną liczbę bitów klucza i porównując publicznym kanałem ich wartości. Te bity oczywiście następnie się wyrzuca. Jeśli liczba błędów przekracza założony poziom to uznaje się, że kanał był podsłuchiwany i procedurę uzgadniania klucza rozpoczyna się od nowa.

Mechanika kwantowa nie dopuszcza możliwości pasywnego podsłuchu. Bezpieczeństwo kwantowego systemu kryptograficznego gwarantowane jest przez prawa fizyki!

Alicja	+	×	+	×	×	+	×	×	×	+	+	+	×
	↑	↗	→	↗	↖	→	↖	↖	↗	↑	↑	↑	↗
	1	0	0	0	1	0	1	1	0	1	1	1	0
Ewa	+	+	+	×	+	×	+	×	+	+	+	×	+
	↑	→	→	↗	→	↗	↑	↖	↑	↑	↑	↖	→
	1	0	0	0	0	0	1	1	1	1	1	1	0
Bolek	+	+	×	+	×	×	×	+	×	+	+	×	×
	↑	→	↗	↑	↖	↗	↖	→	↗	↑	↑	↖	↖
	1	0	0	1	1	0	1	0	0	1	1	1	1
klucz	1				1		1		0	1	1		1

- **Protokół B92** (Bennett, 1992)
  W 1992 r. Charles Bennett zaproponował protokół wymiany klucza oparty na dwóch nieortogonalnych stanach kwantowych. Niech takimi stanami będą $\{|\rightarrow\rangle, |\nearrow\rangle\}$. Bolek wykonuje pomiary polaryzacji w stanach ortogonalnych do $\{|\rightarrow\rangle, |\nearrow\rangle\}$, tzn. w stanach $\{|\uparrow\rangle, |\nwarrow\rangle\}$.

  ⋆ **Alfabet kwantowy**

  Alicja przygotowuje fotony o polaryzacji horyzontalnej $|\rightarrow\rangle$ lub polaryzacji 45° $|\nearrow\rangle$ przypisując im wartości binarne

$$\begin{aligned} |\rightarrow\rangle &\equiv 0 \\ |\nearrow\rangle &\equiv 1 \end{aligned}$$

  ⋆ **Etapy B92**

1. Alicja wybiera losowo jedną z dwóch polaryzacji $\{|\rightarrow\rangle, |\nearrow\rangle\}$ i przesyła do Bolka foton o takiej polaryzacji. Powtarzając tę procedurę, Alicja wysyła do Bolka losowy ciąg zer i jedynek.
2. Bolek losowo wybiera jeden ze stanów $\{|\uparrow\rangle, |\nwarrow\rangle\}$ i mierzy polaryzację w takim stanie. Jeśli wybrał polaryzację ortogonalną do polaryzacji wybranej przez Alicję, to nie zarejestruje fotonu. W przeciwnym razie z prawdopodobieństwem 1/2 zarejestruje foton. Jeśli zarejestrował foton o polaryzacji $|\uparrow\rangle$ to przypisuje mu wartość binarną 1, zaś fotonowi o polaryzacji $|\nwarrow\rangle$ przypisuje wartość binarną 0.
3. Bolek przekazuje Alicji publicznym kanałem informację dla których fotonów uzyskał wynik pozytywny

(T), czyli zarejestrował foton, ale nie zdradza jaką polaryzację zmierzył.

4. Alicja i Bolek przechowują ciąg bitów, dla których Bolek zarejestrował foton. Ciąg ten stanowi klucz kryptograficzny.

Przykład:

Alicja	↗	→	→	→	↗	→	↗	↗	→	↗	↗	↗	→
	1	0	0	0	1	0	1	1	0	1	1	1	0
Bolek	↖	↖	↑	↖	↑	↑	↑	↖	↑	↖	↖	↑	↑
	N	T	N	T	T	N	N	N	N	N	N	T	N
		0		0	1							1	
A/B		√		√	√							√	
klucz		0		0	1							1	

Podobnie jak w przypadku protokołu BB84 obecność Ewy spowoduje błędy w kluczu, które Alicja i Bolek mogą wykryć.

Kryptografia kwantowa szybko się rozwija. Tutaj przedstawiłem tylko najprostsze protokoły. Istnieją inne protokoły kwantowe uzgadniania klucza, np. protokół zaproponowany prze Ekerta w 1991 r oparty na zjawisku EPR. Do kodowania można używać np. fazy fotonu, a nie polaryzacji.
Kryptografia kwantowa jest już faktem!.

Grupa prof. Gisina w Genewie przeprowadziła udane eksperymenty z kwantową dystrybucją klucza na odległości 67 km, używając komercyjnych światłowodów. Trwają intensywne prace nad kwantową dystrybucją klucza w otwartej przestrzeni.

Mechanika kwantowa, która z jednej strony może spowodować, że klasyczne algorytmy kryptograficzne staną się bezużyteczne, z drugiej strony daje możliwość wykorzystania jej praw do bezpiecznego przekazywania klucza kryptograficznego.