

# Szyfry strumieniowe

## RC4

Paweł Burdzy  
Michał Legumina  
Sebastian Stawicki

# Szyfry strumieniowe

W kryptografii, szyfrowanie strumieniowe jest szyfrowaniem, w którym szyfrowaniu podlega na raz jeden bit (czasem jeden bajt).

Czasem nazywa się to szyfrowaniem stanowym, ponieważ szyfrowanie bitu zależy od aktualnego stanu.

# Szyfry strumieniowe

W dokumentach NSA (National Security Agency) czasami używa się terminu “combiner-type algorithms” w odniesieniu do algorytmów, które używają jakiejś funkcji do połączenia wyjścia z pseudolosowego generatora liczb (PRNG) z niezaszyfrowaną wiadomością.

# Szyfry strumieniowe

PRNG – pseudorandom number generator.  
Algorytm, który generuje ciąg liczb, którego elementy są w przybliżeniu losowe.

# Szyfry strumieniowe

Generator strumienia szyfrującego to podstawa bezpieczeństwa szyfrów strumieniowych.

Ważne jest aby generator produkował ciąg w dużym przybliżeniu losowy, nie mówiący nic o wykorzystywanym kluczu i o możliwie długim okresie, czyli liczbie iteracji, po której ciąg zaczyna się powtarzać.

# Szyfry strumieniowe

Czyli jak to działa?

# Szyfry strumieniowe

Alicja chce przesłać wiadomość do Boba.

**Alicja**

wiadomość



**Bob**

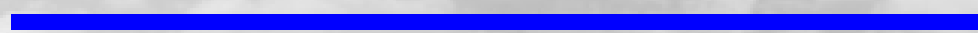
# Szyfry strumieniowe

Alicji i Bobowi znany jest klucz szyfrujący.

**Alicja**

wiadomość

klucz



**Bob**

klucz



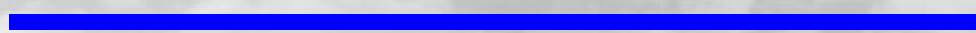
# Szyfry strumieniowe

Mają też ogólnie dostępną maszynkę do produkowania strumienia szyfrującego (PRNG).

**Alicja**

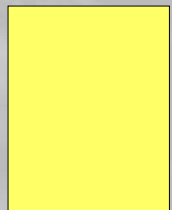
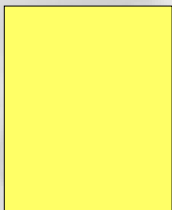
wiadomość

klucz



**Bob**

klucz

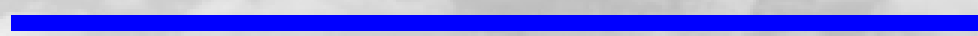


# Szyfry strumieniowe

Mają też ogólnie dostępną maszynkę do produkowania strumienia szyfrującego (PRNG).

**Alicja**

wiadomość



**Bob**

klucz

klucz

# Szyfry strumieniowe

Teraz Alicja bajt po bajcie – bierze jeden bajt z wiadomości i jeden pseudolosowy i xor'uje.



# Szyfry strumieniowe

**Alicja**

**Bob**



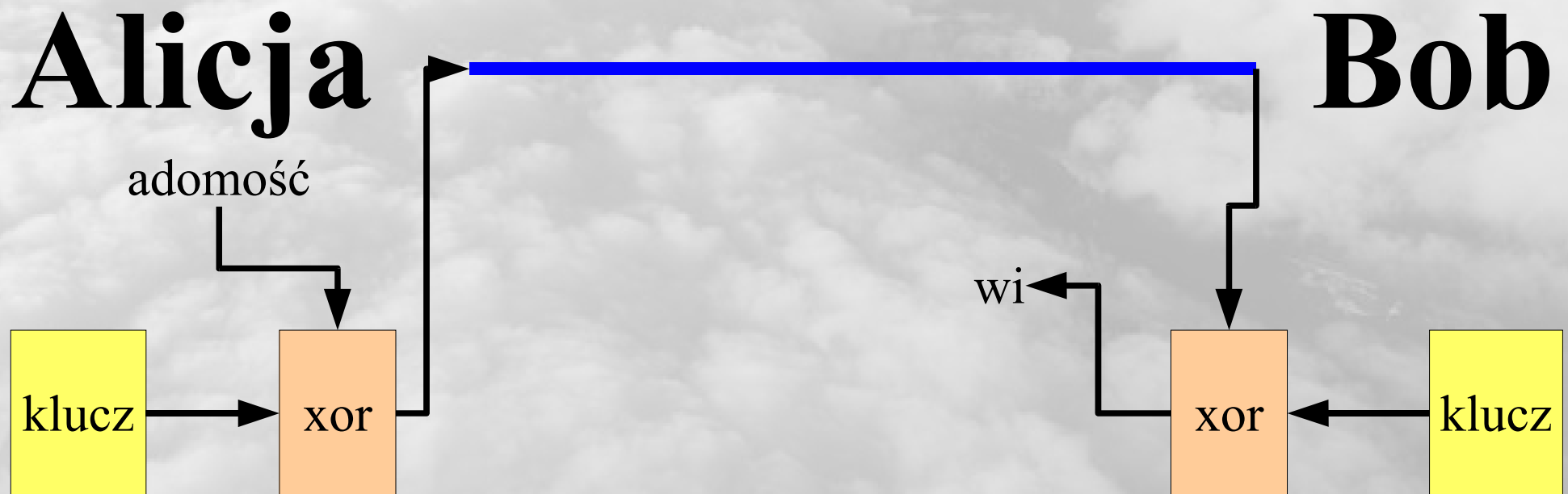
# Szyfry strumieniowe

A teraz Bob bajt po bajcie – bierze jeden bajt z otrzymanych i jeden pseudolosowy i xor'uje.



# Szyfry strumieniowe

A teraz Bob bajt po bajcie – bierze jeden bajt z otrzymanych i jeden pseudolosowy i xor'uje.



# Szyfry strumieniowe

A teraz Bob bajt po bajcie – bierze jeden bajt z otrzymanych i jeden pseudolosowy i xor'uje.



# strumieniowe vs blokowe

Szyfry strumieniowe są bardziej odpowiednie do zastosowań z ograniczoną możliwością buforowania lub tam gdzie porcje danych muszą być od razu, jak tylko się pojawią, przetwarzane.



# strumieniowe vs blokowe

W szyfrach strumieniowych nie występuje propagacja błędów lub ma ona znikomy zasięg. Dzięki tej własności szyfry strumieniowe mogą okazać się bardziej odpowiednie od blokowych w sytuacjach o dużym prawdopodobieństwie błędów transmisji.

# strumieniowe vs blokowe

Szyfry strumieniowe są używane tam, gdzie dane pojawiają się w nieznanej długości porcjach – np., w bezpiecznych połączeniach bezprzewodowych.

# strumieniowe vs blokowe

Szyfry blokowe w takich sytuacjach byłyby nieodpowiednie bo transmitowane byłyby niepotrzebne porcje danych – dopełnienia właściwych danych do pełnego bloku.

# Synchroniczne szyfry strumieniowe

Synchroniczne szyfry strumieniowe – PRNG jest inicjalizowany kluczem i generuje strumień szyfrujący niezależny od samej wiadomości i od szyfrogramu.

# Synchroniczne szyfry strumieniowe

Jest potrzebna synchronizacja tzn. nadawca i odbiorca muszą być zsynchronizowani.

# Synchroniczne szyfry strumieniowe

Nie ma propagacji błędów – pojedynczy błąd nie ma wpływu na kolejne porcje danych.

Doskonale nadaje się do sytuacji o dużym prawdopodobieństwie błędu transmisji.

# Synchroniczne szyfry strumieniowe

Niestety jest podatność na ataki zmiany bitów. Zmiana bitu w szyfrogramie powoduje zmianę bitu w rozszyfrowanej wiadomości.

# Synchroniczne szyfry strumieniowe

Podatność na ataki wstawiania i usuwania bitów  
szyfrogramu – utrata synchronizacji.



# **Samo-synchronizujące szyfry strumieniowe**

Generator strumienia szyfrującego używa kilku poprzednich bitów szyfrogramu przy generowaniu kolejnych elementów ciągu.

# **Samo-synchronizujące szyfry strumieniowe**

Pojedynczy błąd rozprzestrzenia się na kilka kolejnych elementów szyfrogramu.

Powrót do poprawnego deszyfrowania po wstawieniu lub usunięciu bitów (czyli po stracie synchronizacji) jest automatyczny. Z ustaloną ilością niemożliwych do odzyskania bitów wiadomości.

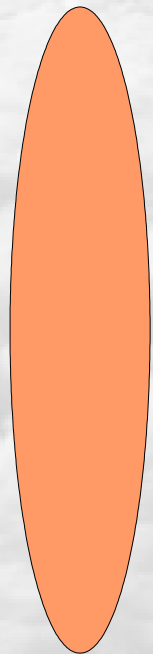
# RC4

- Historia
- Opis
- Bezpieczeństwo

# RC4 - historia

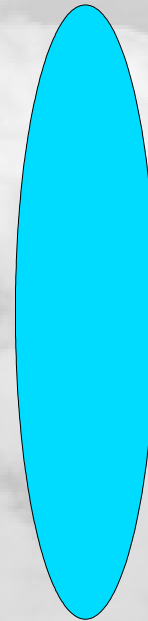
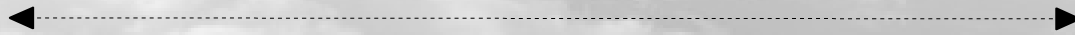
- Zaprojektowany przez Rona Rivest'a (RSA) – 1987 r.
- Początkowo tajny ale anonimowo dodany do listy dyskusyjnej – 1994 r.
- Później już szeroko omawiany chociażby na wielu stronach internetowych
- Nazwa została opatentowana, nie można jej nielegalnie używać, sam algorytm jednak może być wykorzystywany
- Unika się problemów z odnośnikami stosując powszechnie nawet ARCFOUR
- Stał się powszechnie stosowany ( WEP, WPA, SSL)

# RC4 - założenia



**Alicja**

Posiadają ten sam klucz K

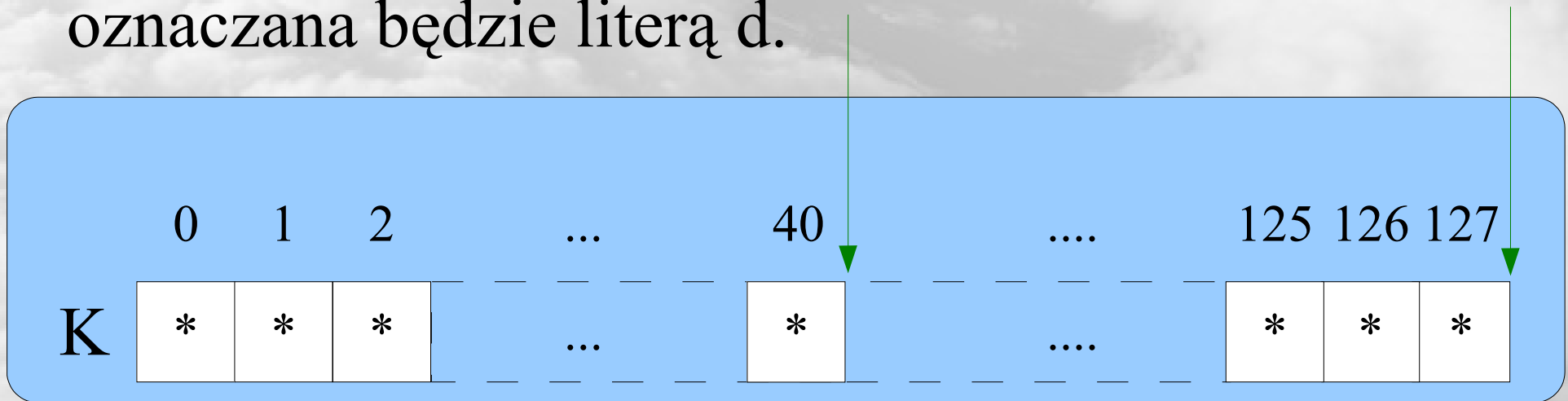


**Bob**

# RC4 - założenia

## KLUCZ:

Ciąg bajtów zmiennej długości, stosuje się od 5 do 16 bajtów na klucz. Dalej ta długość oznaczana będzie literą *d*.



# RC4 - opis

## Krótko:

- Generujemy z klucza pseudolosowy ciąg bitów
    - Przygotowanie “losowej” permutacji  $S[256]$
    - Wygenerowanie za pomocą  $S$  dowolnej długości “losowego” ciągu bitów
  - Traktujemy ciąg jako klucz w szyfrze Vernama, czyli XOR z wiadomością daje szyfrogram
- 

Odbiorca posiada ten sam klucz więc generuje ten sam ciąg bitów, który XOR razem z szyfrogramem zamieni w wiadomość.

# RC4 – szczegóły 1

Przygotowanie “losowej” permutacji  $S[256]$  (KSA):

KSA = The key-scheduling algorithm

Mamy 256 elementową, identycznościową permutację  $S$ .

```
for i from 0 to 255  
  S[i] := i
```

	0	1	2	...	253	254	255
S	0	1	2	...	253	254	255



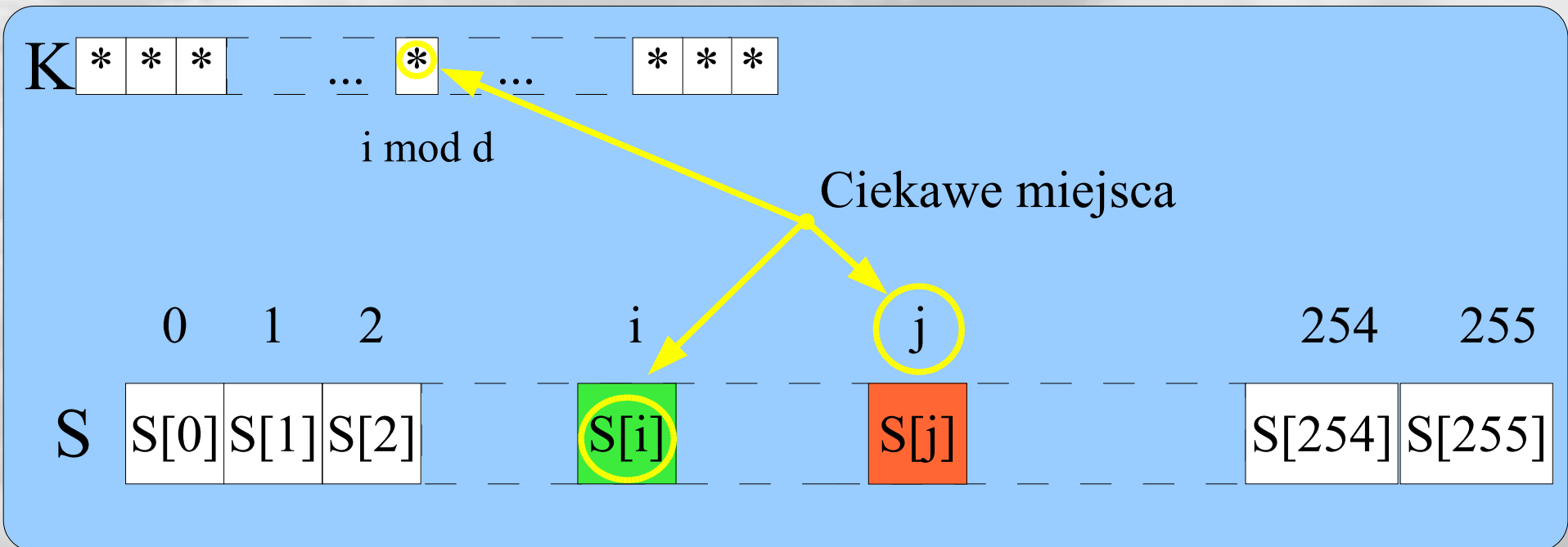
# RC4 – szczegóły 2

## The key-scheduling algorithm (KSA):

– Stworzymy alternatywną, pseudolosową permutację.

```
j := 0
for i from 0 to 255
  j := (j + S[i] + key[i mod d]) mod 256
  swap(S[i], S[j])
```

$i = 0 \dots 255$



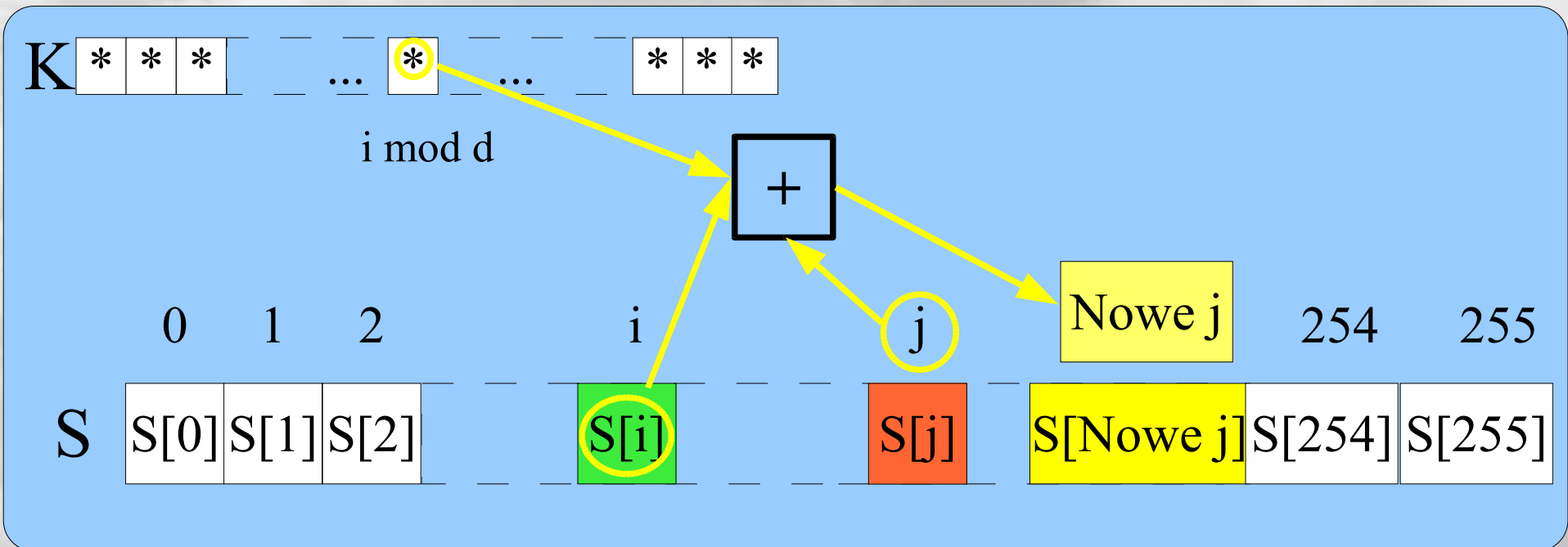
# RC4 – szczegóły 3

## The key-scheduling algorithm (KSA):

– Stworzymy alternatywną, pseudolosową permutację.

```
j := 0
for i from 0 to 255
  j := (j + S[i] + key[i mod d]) mod 256
  swap(S[i], S[j])
```

$i = 0 \dots 255$



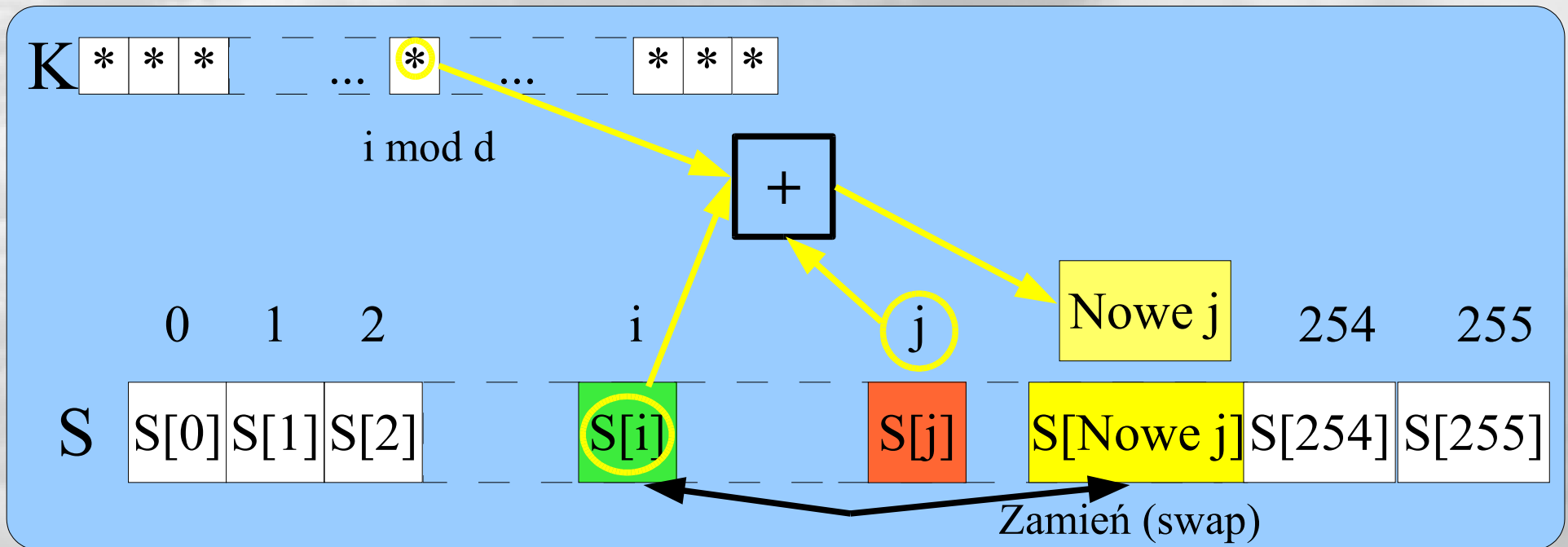
# RC4 – szczegóły 4

## The key-scheduling algorithm (KSA):

– Stworzymy alternatywną, pseudolosową permutację.

```
j := 0
for i from 0 to 255
  j := (j + S[i] + key[i mod d]) mod 256
  swap(S[i], S[j])
```

$i = 0 \dots 255$

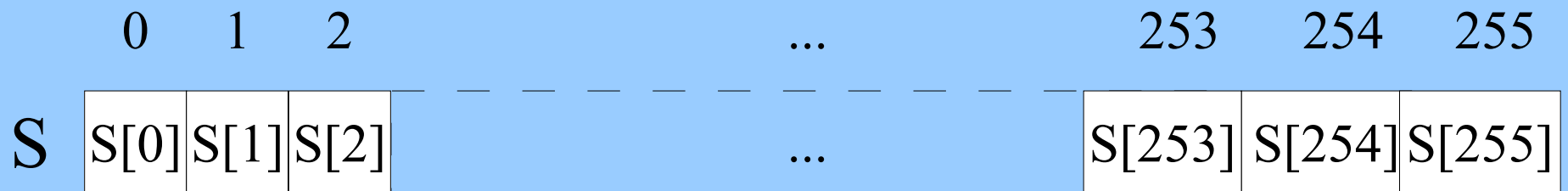


# RC4 – szczegóły 5

## The key-scheduling algorithm (KSA):

Stworzyliśmy więc z  $S$  oraz naszego klucza, alternatywną permutację.

```
j := 0
for i from 0 to 255
  j := (j + S[i] + key[i mod 1]) mod 256
  swap(S[i], S[j])
```



# RC4 – szczegóły 5

Wygenerowanie za pomocą S dowolnej długości “losowego” ciągu bitów (PRGA):

PRGA = The pseudo-random generation algorithm

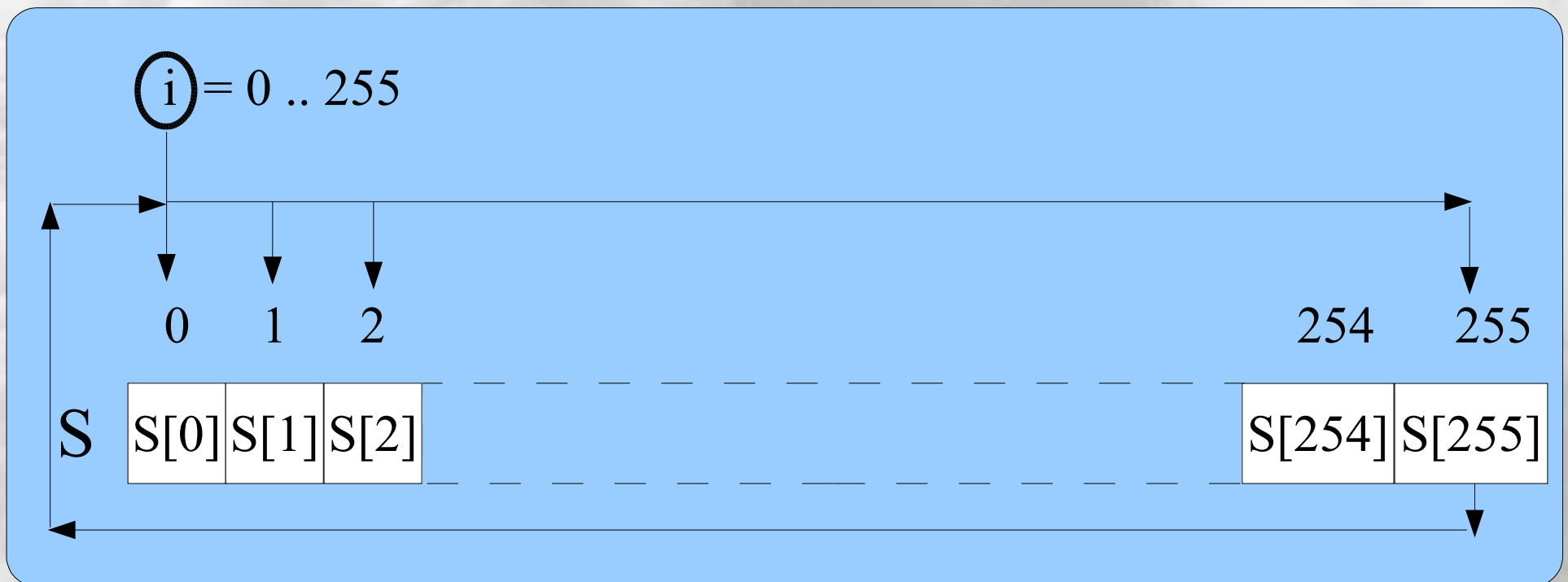
Tak długo jak zachodzi potrzeba generujemy kolejne szyfrowe bajty, którymi potem będą szyfrowane kolejne bajty wiadomości.

```
i := 0
j := 0
while GeneratingOutput:
  i := (i + 1) mod 256
  j := (j + S[i]) mod 256
  swap(S[i], S[j])
  output S[(S[i] + S[j]) mod 256]
```

# RC4 – szczegóły 6

The pseudo-random generation algorithm (PRGA):

- początkowo zmienne  $i$  oraz  $j$  ustawiamy na 0
- $i$  będzie przechodziło cyklicznie po tablicy  $S$
- $j$  będzie ostro mieszało co się da

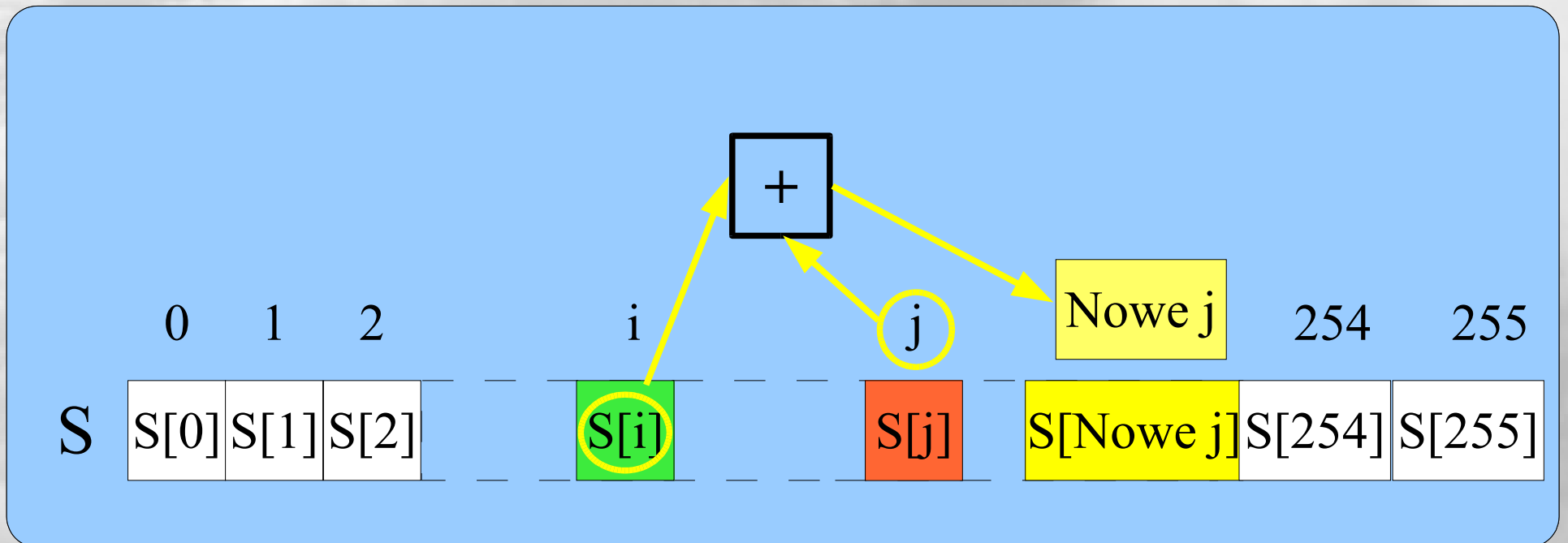


# RC4 – szczegóły 6

The pseudo-random generation algorithm (PRGA):

- nowe  $j$  tworzymy podobnie jak w **KSA**

$i = 0 .. 255$

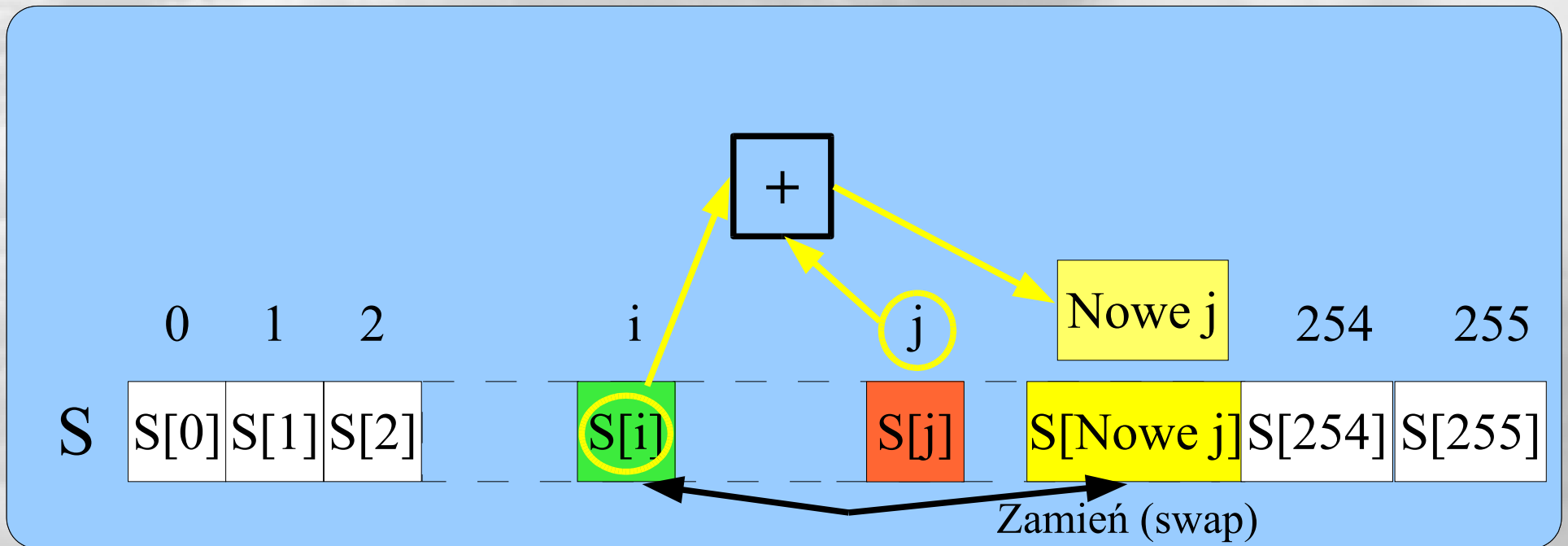


# RC4 – szczegóły 6

The pseudo-random generation algorithm (PRGA):

- nowe  $j$  tworzymy podobnie jak w **KSA**

$i = 0 .. 255$



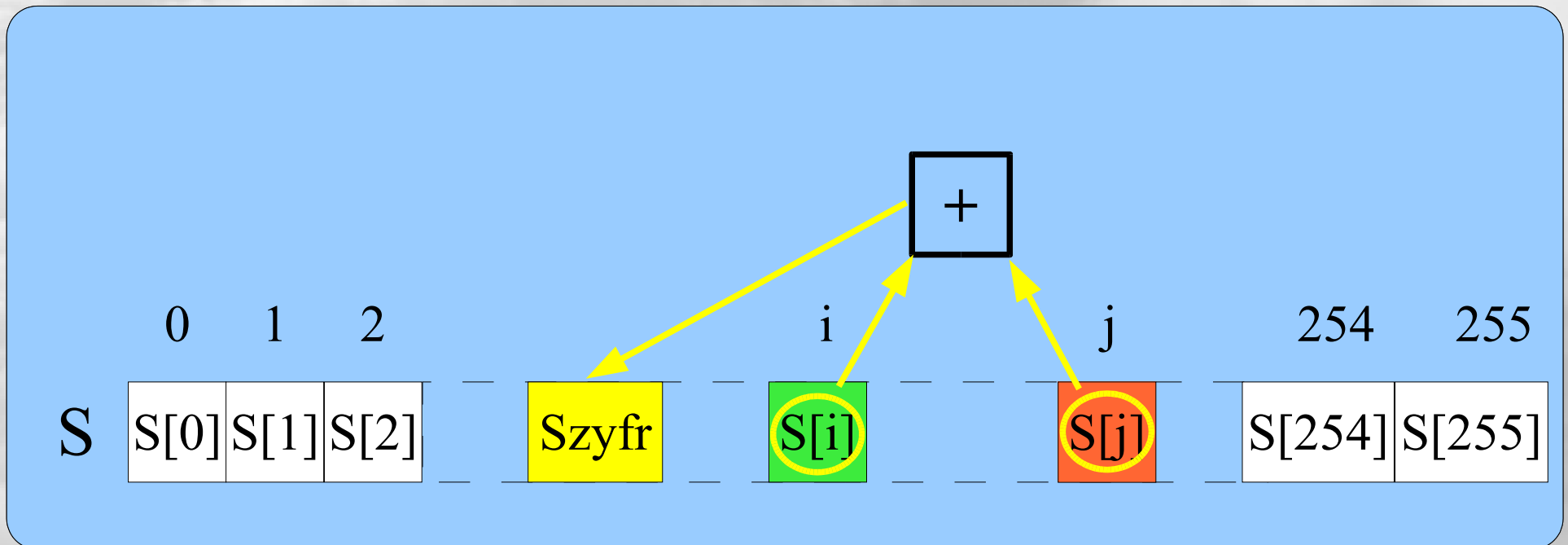


# RC4 – szczegóły 6

The pseudo-random generation algorithm (PRGA):

Dany bajt powstaje przez wzięcie  $i$  oraz  $j$  elementu z permutacji, dodanie tych liczb modulo 256 i spojrzenie co jest w permutacji na owym miejscu.

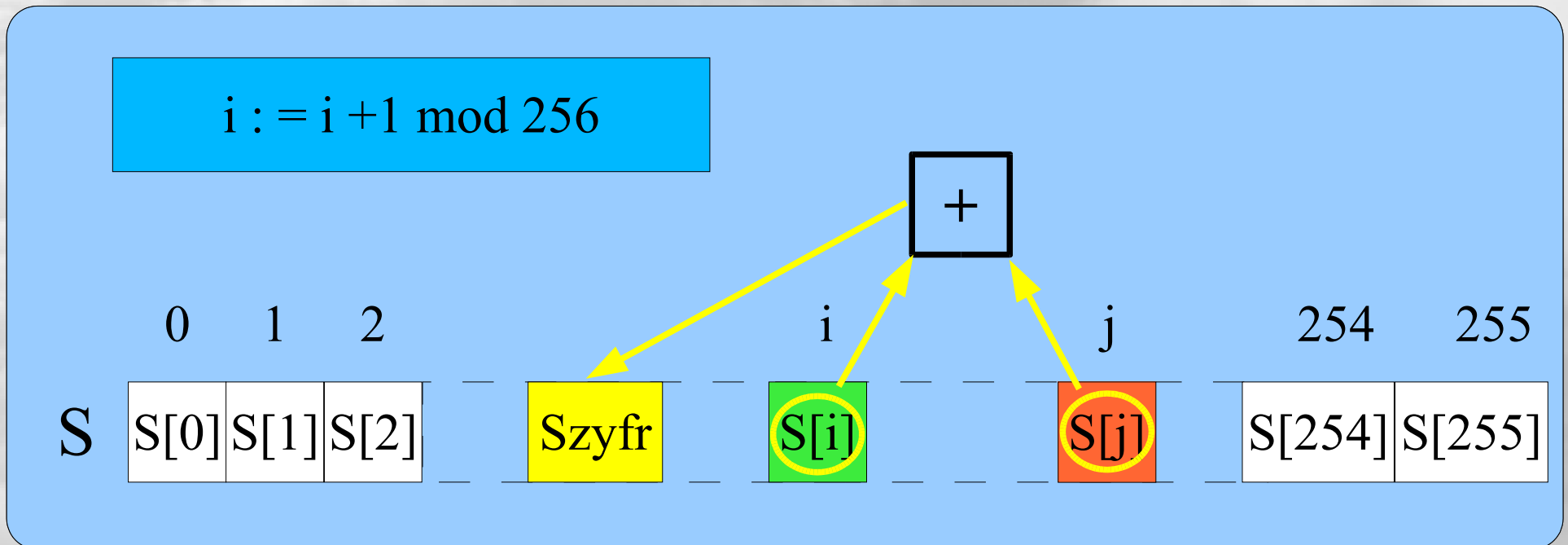
$i = 0 \dots 255$



# RC4 – szczegóły 6

The pseudo-random generation algorithm (PRGA):

Operacja ta jest kontynuowana tak długo jak “losowe” bajty są potrzebne do kolejnych znaków wiadomości.



# RC4 – implementacja

- z reguły strumieniowe oparte są o LFSR
- RC4 zaprojektowany w idealny do implementacji sposób
- wszystkie operacje dotyczą liczb  $\leq 256$
- pomocnicza permutacja S zajmuje tylko 256 B
- d bajtów zajmuje klucz
- pozostają już tylko zmienne i, j
- dodawanie modulo łatwe

# RC4 – bezpieczeństwo

- nie należy używać wielokrotnie tego samego klucza
- wiadomość tym samym kluczem będzie tak samo zakodowana
- algorytm jest w istocie deterministyczny z punktu widzenia Alicji i Boba
- istnieją algorytmy, które mając GB szyfrogramu potrafią rozróżnić RC4

An aerial photograph of a coastal city, likely San Francisco, showing a dense urban area and a large body of water. The image is semi-transparent, allowing the text to be clearly visible. The text 'Słabości RC4' is centered in a bold, black, sans-serif font.

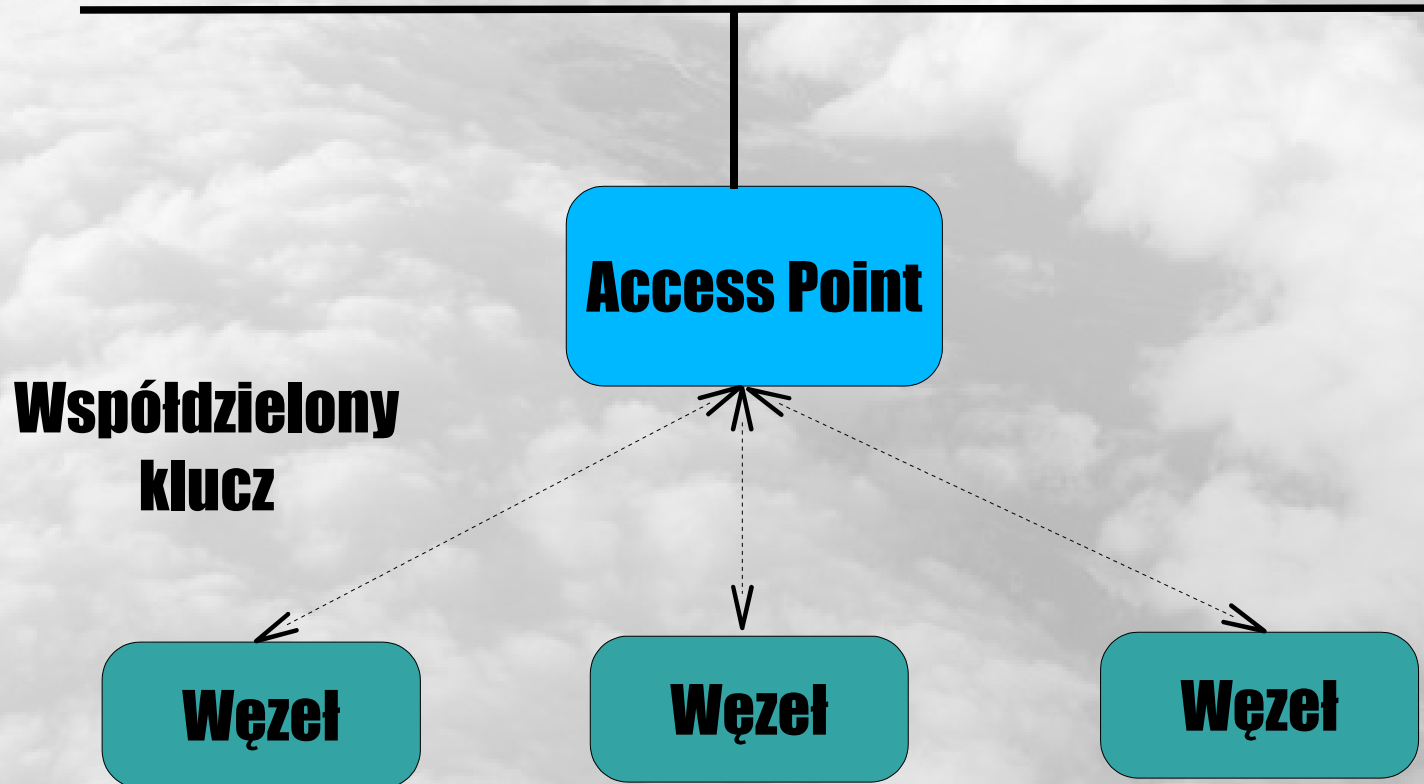
# Słabości RC4

# RC4 w WEP

- Wired Equivalent Privacy (WEP)
  - Protokół wymiany szyfrowanych pakietów transmitowanych w standardzie 802.11
  - Prosty w administracji
  - W sieci obowiązuje jedno hasło

# WEP

LAN



# Słabości WEP

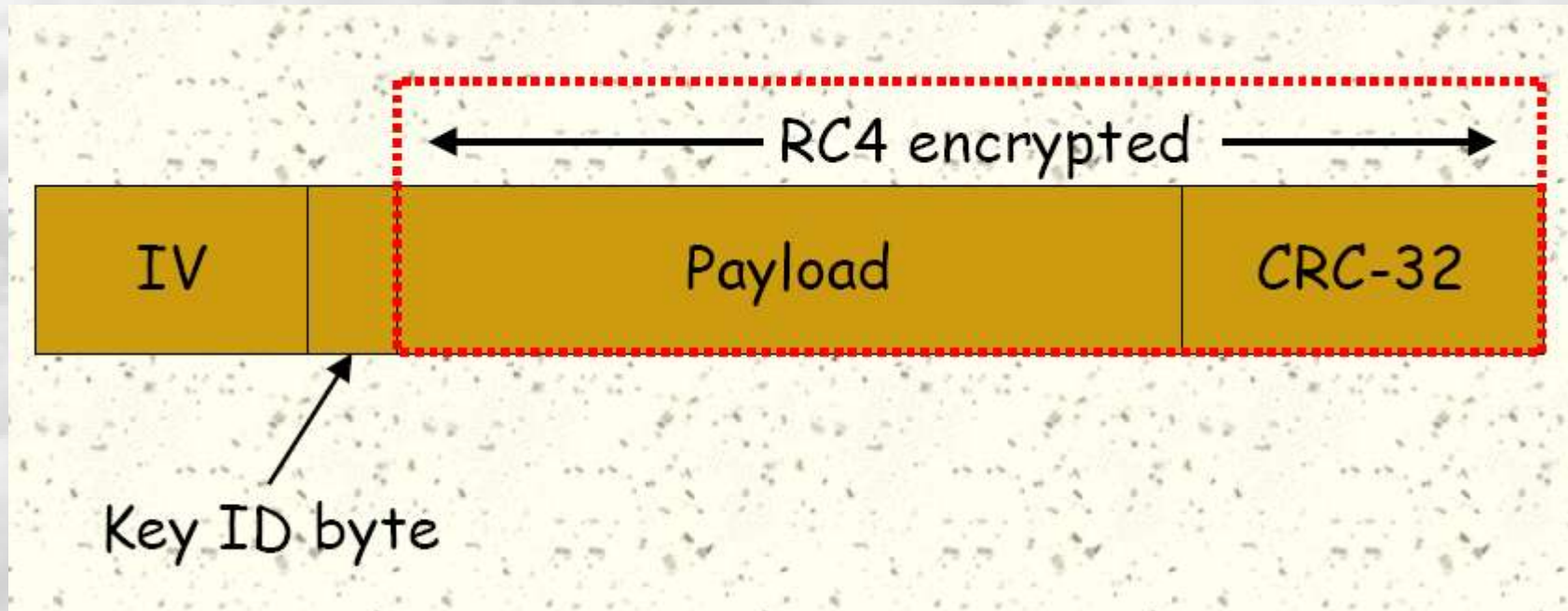
- Niektóre karty resetują IV (initialization vector) do wartości zero podczas każdej inicjalizacji
- Przestrzeń z której wybierany jest IV jest zbyt mała
- Atak na RC4 używany w WEP
  - Atak na RC4 metodą opisaną przez Fluhrera, Mantina i Shamira
    - Wykorzystanie dużej ilości szyfrogramów
    - Analiza statystyczna



# Szyfrowanie w WEP

- Dany klucz  $k$  oraz wiadomość  $M$
- Następuje konkatencja  $M$  z sumą kontrolną wiadomości  $c(M)$ ; wynik  $M*c(M)$
- Do wektora  $IV$  przyłączany jest klucz  $k$  dając klucz używany w RC4; wynik  $IV*k$
- Szyfrogram  $C=(M*c(m)) \text{ XOR } \text{RC4}(IV*k)$

# Format pakietu



# Łamanie WEPa

- Przechwytywanie pakietów
- Czekać na kolizje w wyborze IV (to zdarza się często)
- Trzeba zebrać około 6.000.000 pakietów

# Nieprawidłowe użycie RC4 w Word i Excel

- Szyfr ma być nie tylko dobry
- Musi być poprawnie użyty!!!
  - Wpadka w implementacji SSL w Netscape 1.1
  - Niedociągnięcia w obsłudze RC4 w MS Office

# Rodzaj błędu w MS Word

- Tworzymy dokument i zapisujemy go zabezpieczając hasłem
- Dokonujemy modyfikacji w dokumencie i zapisujemy do innego pliku
- ZOSTAŁ UŻYTY TEN SAM WEKTOR INICJALIZUJĄCY
- Mamy dwa szyfrogramy, które niewiele się różnią
- Można z nich wyciągać informacje o dokumencie

# Kiedy jest to niebezpieczne?



**Alicja pisze sobie  
w Wordzie**

# Kiedy jest to niebezpieczne?

Robi backup'y  
na serwerze  
w sieci lokalnej



**Alicja pisze sobie  
w Wordzie**

# Kiedy jest to niebezpieczne?

Robi backup'y  
na serwerze  
w sieci lokalnej



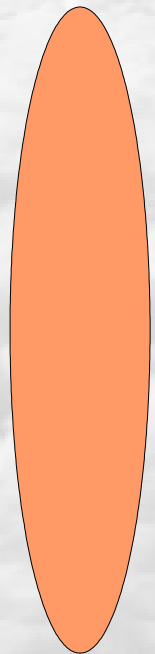
**Alicja pisze sobie  
w Wordzie**

Ewa przechwytuje  
zaszyfrowane kopie  
dokumentów



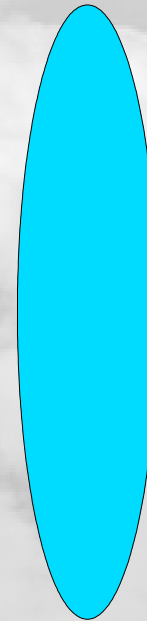
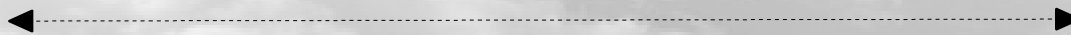


# Kiedy jest to niebezpieczne?



**Alicja**

Pracują nad jednym dokumentem



**Bob**

# Kiedy jest to niebezpieczne?



# Bezpieczeństwo w MS Office

- W MS Office 95 hasło było powielane i dokument xor'owany z tak otrzymanym ciągiem
  - To żadne zabezpieczenie
- W późniejszych wersjach pojawił się RC4
  - Na początku z 40 bitowym kluczem
  - To obecnie stosunkowo łatwo łamie się brute forcem
- Następnie RC4 z kluczem 128 bitowym
  - To już odporne na brute force

# Bezpieczeństwo w MS Office

- 128 bitowy klucz w RC4 już bezpieczny
  - Jednak źle zarządzany jest wektor inicjalizujący
- Nie jest przeszkodą ponowne używanie tego samego klucza
  - Klucz i wektor inicjalizujący są hashowane i dają w wyniku klucz używany przez RC4

# Dowody

- Używamy MS Word 2003
- Tworzymy dokument, który zawiera wiersz
  - Autorstwa Bolesława Leśmiana
  - Nie wpisujemy polskich liter

# Dowody

Śni się lasom las,  
Śnią się deszcze.  
Jawia się raz w raz  
Znikłe maje.  
I mija znów,  
I raz jeszcze...  
A ja własnych słów  
Nie poznaję

# Dowody

- Zapisujemy plik
- Patrzymy na zapis binarny

9F6	00 00 00 00 00 00 00 00 00 00 00 53 6E 69 20 73 69 65	□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □	S n i s i e
A07	20 6C 61 73 6F 6D 20 6C 61 73 2C 20 53 6E 69 61 20	l a s o m l a s , S n i a	
A18	73 69 65 20 64 65 73 7A 63 7A 65 2E 20 4A 61 77 69	s i e d e s z c z e . J a w i	
A29	61 20 73 69 65 20 72 61 7A 20 77 20 72 61 7A 20 5A	a s i e r a z w r a z Z	
A3A	6E 69 6B 6C 65 20 6D 61 6A 65 2E 20 49 20 6D 69 6A	n i k l e m a j e . I m i j	
A4B	61 6A 61 20 7A 6E 6F 77 2C 20 49 20 72 61 7A 20 6A	a j a z n o w , I r a z j	
A5C	65 73 7A 63 7A 65 2E 2E 2E 20 41 20 6A 61 20 77 6C	e s z c z e . . . A j a w l	
A6D	61 73 6E 79 63 68 20 73 6C 6F 77 20 4E 69 65 20 70	a s n y c h s l o w N i e p	
A7E	6F 7A 6E 61 6A 65 2E 0D 00 00 00 00 00 00 00 00 00	o z n a j e . □ □ □ □ □ □ □ □ □ □	

# Dowody

- Szyfrujemy plik RC4, hasło potrzebne do otwarcia dokumentu (użyłem hasła “zima”)
- Zapisujemy plik pod inną nazwą

9F6	A7	81	29	2E	EB	05	E1	E0	65	DD	6A	8A	CF	7B	17	FB	AE	\$	□	)	.	ë	□	á	ř	e	Ý	j	Š	Ď	{	□	ú	⊗
A07	E5	BA	BC	D8	DA	C1	5A	01	7C	48	FF	EB	39	96	19	31	CE	í	ξ	Ť	Ř	Ú	Á	Z	□		H	·	ë	9	-	□	l	Í
A18	AF	FD	D8	EF	8A	92	6F	6A	24	DE	9C	80	E1	82	CC	8D	58	ž	ý	Ř	ř	Š	'	o	j	ř	š	e	á	,	Ě	Ť	X	
A29	D1	D7	24	97	21	BF	E7	5F	DF	6C	DA	22	1C	C8	2F	38	8A	Ň	x	ř	-	!	ž	ř	_	š	l	Ů	"	□	č	/	8	š
A3A	66	1E	41	02	37	F5	3D	1F	05	FB	1F	93	2E	0A	A3	D9	E9	f	□	A	□	7	ó	=	□	□	ú	□	"	.	□	ł	Ů	é
A4B	96	69	69	F2	16	DC	92	B6	A7	21	6B	8D	0C	8F	02	7F	A7	-	i	i	ň	□	Ů	'	ŕ	š	!	k	ř	□	ž	□	□	š
A5C	B0	12	E7	7E	64	16	8D	81	23	B5	AA	17	6D	30	C8	68	OD	"	□	ř	~	d	□	ř	□	#	μ	š	□	m	0	č	h	□
A6D	9D	90	77	9A	FD	87	A6	DD	E8	53	B6	8B	31	F9	39	6F	CC	č	□	w	š	ý	†	!	Ý	č	š	ŕ	<	l	ů	9	o	Ě
A7E	87	84	89	C7	E2	A0	FD	DD	79	FB	A5	E1	EA	D4	77	2E	6F	†	„	ž	č	á	ý	ř	y	ú	á	ę	ô	w	.	o		



# Dowody

- Zamieniamy w dokumencie “raz w raz” na “raz po raz”
- Zapisujemy plik pod inną nazwą

9F6	A7	81	29	2E	EB	05	E1	E0	65	DD	6A	8A	CF	7B	17	FB	AE	S	□	)	.	ě	□	á	ř	e	Ý	j	Š	Ď	{	□	ú	⊗	
A07	E5	BA	BC	D8	DA	C1	5A	01	7C	48	FF	EB	39	96	19	31	CE	í	š	ř	Ú	Á	Z	□		H	´	ě	9	-	□	l	í		
A18	AF	FD	D8	EF	8A	92	6F	6A	24	DE	9C	80	E1	82	CC	8D	58	Ž	ý	Ř	ř	Š	´	o	j	š	ř	ě	á	,	ě	ř	X		
A29	D1	D7	24	97	21	BF	E7	5F	DF	6C	DD	6D	4E	DB	34	62	FO	Ň	×	š	-	!	ž	č	_	š	l	ŷ	m	Ň	Ů	4	b	ď	
A3A	52	19	43	05	3E	B0	70	13	0E	F4	54	9D	47	63	EE	DD	EA	R	□	C	□	>	"	p	□	□	ó	T	t	C	c	í	Ý	ę	
A4B	9D	62	62	B3	4C	C8	93	AE	FC	2D	02	E4	5E	9C	19	25	ED	č	b	b	i	L	Č	“	⊗	ü	-	□	ä	^	š	□	*	í	
A5C	BF	04	EE	67	7D	09	C6	81	23	BB	CB	76	27	3B	89	3F	16	ž	□	í	g	}	□	Č	□	#	»	Ě	v	'	;	ž	?	□	
A6D	90	82	6A	8D	E7	8C	EE	8E	F7	50	AE	DC	5F	DE	35	2A	9C	□	,	j	ř	č	š	í	ž	+	P	⊗	Ů	_	ř	š	*	š	
A7E	98	91	9D	C8	E9	AF	B6	F3	74	FB	A5	E1	EA	D4	77	2E	6F	□	`	č	ě	ž	ř	ó	t	ú	Ā	á	ę	ô	w	.	o		

# Dowody

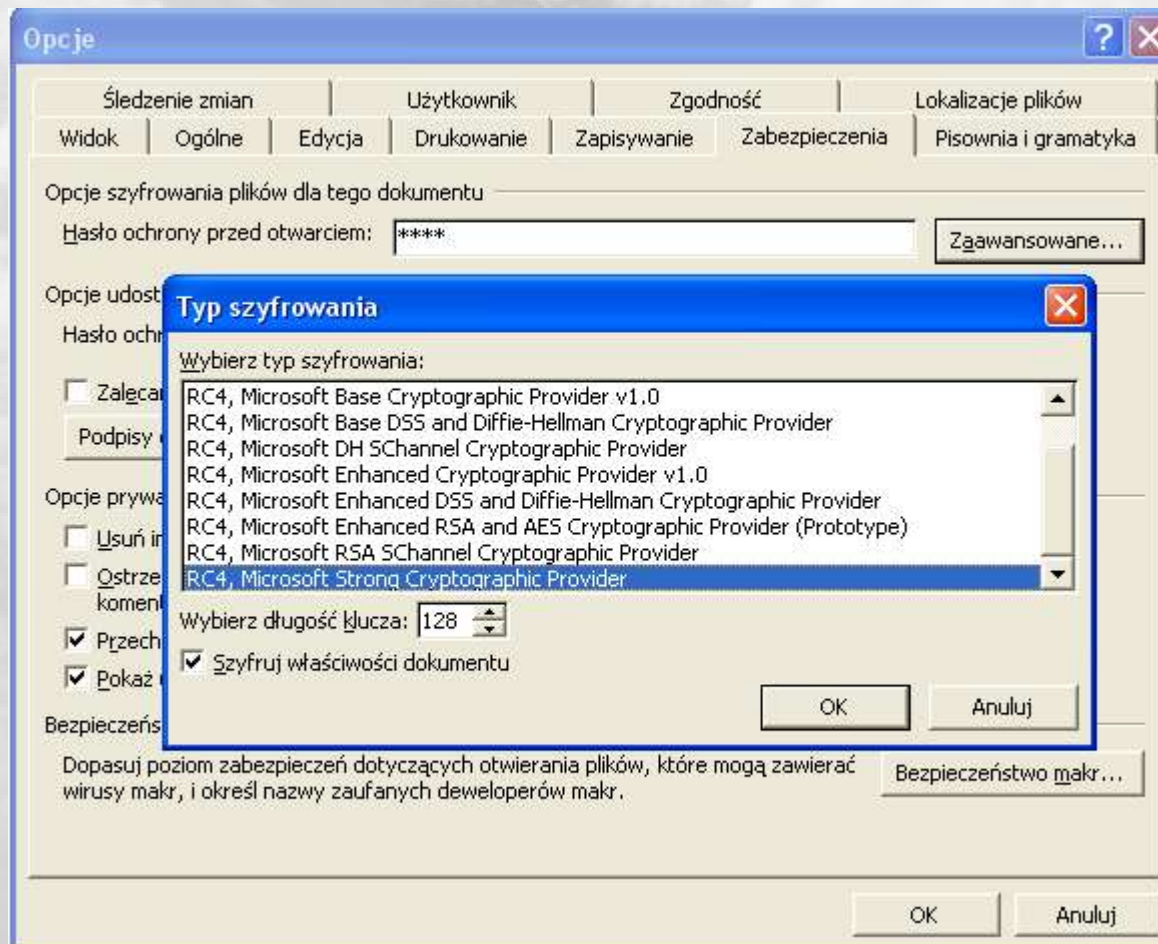
9F6	00 00 00 00 00 00 00 00 00 00 53 6E 69 20 73 69 65	□ □ □ □ □ □ □ □ □ □ □ □ Sni sie
A07	20 6C 61 73 6F 6D 20 6C 61 73 2C 20 53 6E 69 61 20	lasom las, Snia
A18	73 69 65 20 64 65 73 7A 63 7A 65 2E 20 4A 61 77 69	sie deszcze. Jawi
A29	61 20 73 69 65 20 72 61 7A 20 77 20 72 61 7A 20 5A	asie raz raz Z
A3A	6E 69 6B 6C 65 20 6D 61 6A 65 2E 20 49 20 6D 69 6A	nikle maje. Imij
A4B	61 6A 61 20 7A 6E 6F 77 2C 20 49 20 72 61 7A 20 6A	aja znów, Iraz j
A5C	65 73 7A 63 7A 65 2E 2E 2E 20 41 20 6A 61 20 77 6C	eszcze. . . Aja wl
A6D	61 73 6E 79 63 68 20 73 6C 6F 77 20 4E 69 65 20 70	asných słow Nie p
A7E	6F 7A 6E 61 6A 65 2E 0D 00 00 00 00 00 00 00 00	oznaje. □ □ □ □ □ □ □ □ □ □

9F6	A7 81 29 2E EB 05 E1 E0 65 DD 6A 8A CF 7B 17 FB AE	\$ □ ) . ë □ á ř e ý j š Ď { □ ú ☉
A07	E5 BA BC D8 DA C1 5A 01 7C 48 FF EB 39 96 19 31 CE	í § Ľ Ř Ú Á Z □   H ' ë 9 - □ l í
A18	AF FD D8 EF 8A 92 6F 6A 24 DE 9C 80 E1 82 CC 8D 58	ž ý ř ě š ' o j § Ľ é á , ě ě X
A29	D1 D7 24 97 21 BF E7 5F DF 6C DA 22 1C C8 2F 38 8A	Ń × § - ! ž ç _ B l Ů " □ Č / 8 š
A3A	66 1E 41 02 37 F5 3D 1F 05 FB 1F 93 2E 0A A3 D9 E9	f □ A □ 7 ó = □ □ ú □ " . □ Ł Ů é
A4B	96 69 69 F2 16 DC 92 B6 A7 21 6B 8D 0C 8F 02 7F A7	- i i ě □ Ů ' ¶ § ! k ě □ ž □ □ §
A5C	B0 12 E7 7E 64 16 8D 81 23 B5 AA 17 6D 30 C8 68 0D	" □ ç ~ d □ ě □ # μ § □ m 0 Č h □
A6D	9D 90 77 9A FD 87 A6 DD E8 53 B6 8B 31 F9 39 6F CC	č □ w š ý † ! ý č s ¶ < l ů 9 o ě
A7E	87 84 89 C7 E2 A0 FD D0 79 FB A5 E1 EA D4 77 2E 6F	† „ ž ç á ý ě y ú Ľ á ě Ů w . o

9F6	A7 81 29 2E EB 05 E1 E0 65 DD 6A 8A CF 7B 17 FB AE	\$ □ ) . ë □ á ř e ý j š Ď { □ ú ☉
A07	E5 BA BC D8 DA C1 5A 01 7C 48 FF EB 39 96 19 31 CE	í § Ľ Ř Ú Á Z □   H ' ë 9 - □ l í
A18	AF FD D8 EF 8A 92 6F 6A 24 DE 9C 80 E1 82 CC 8D 58	ž ý ř ě š ' o j § Ľ é á , ě ě X
A29	D1 D7 24 97 21 BF E7 5F DF 6C DD 6D 4E DB 34 62 F0	Ń × § - ! ž ç _ B l Ů " m N Ů 4 b ě
A3A	52 19 43 05 3E B0 70 13 0E F4 54 9D 47 63 EE DD EA	R □ C □ > " p □ □ ó T ě C c i ý ě
A4B	9D 62 62 B3 4C C8 93 AE FC 2D 02 E4 5E 9C 19 25 ED	č b b i L Č " ☉ ů - □ ä ^ é □ § í
A5C	BF 04 EE 67 7D 09 C6 81 23 BB CB 76 27 3B 89 3F 16	ž □ í g ) □ Č □ # » ě v ' ; ž ? □
A6D	90 82 6A 8D E7 8C EE 8E F7 50 AE DC 5F DE 35 2A 9C	□ , j ě ç š í ž + p ☉ Ů _ Ľ 5 * š
A7E	98 91 9D C8 E9 AF B6 F3 74 FB A5 E1 EA D4 77 2E 6F	□ ' ě Č é ž ¶ ó t ů Ľ á ě Ů w . o

# Szyfrowanie w MS Word

- Wybieramy Opcje z Menu Narzędzia



# Łamiemy Excel

- Tworzymy dokument postaci

	A	B	C	D
1	1a	1b	1c	1d
2	2a	2b	2c	2d
3	3a	3b	3c	3d

- Wpisujemy dane wierszami od lewej do prawej
- Zapisujemy



# Łamiemy Excel

- Wprowadzamy modyfikację zamieniając wartość “1c” na “c1”

	A	B	C	D
1	1a	1b	c1	1d
2	2a	2b	2c	2d
3	3a	3b	3c	3d



# Łamiemy Excel

- Szyfrujemy pierwszy plik (plik bez dokonanych zmian)
- Zapisujemy go dwukrotnie
  - Jest to związane z mechanizmem zapisywania Excela
- Szyfrujemy plik ze zmianami i zapisujemy
- Szukamy części wspólnej zapisów binarnych obu plików



# Przeciwdziałanie

- Należy zmieniać wektor inicjalizujący:
- Hasło i wiadomość służą nam do wygenerowania wektora inicjalizującego
  - np. za pomocą HMAC
- Znika nam w ten sposób opisany problem