Kryptografia

z elementami kryptografii kwantowej

Ryszard Tanaś

http://zon8.physd.amu.edu.pl/~tanas

Wykład 8

Spis treści

13 Szyfrowanie strumieniowe i generatory ciągów pseudolosowych

13.1 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.

13 Szyfrowanie strumieniowe i generatory ciągów pseudolosowych

13.1 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.

13 Szyfrowanie strumieniowe i generatory ciągów pseudolosowych

13.1 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.

13 Szyfrowanie strumieniowe i generatory ciągów pseudolosowych

13.1 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.

13 Szyfrowanie strumieniowe i generatory ciągów pseudolosowych

13.1 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.

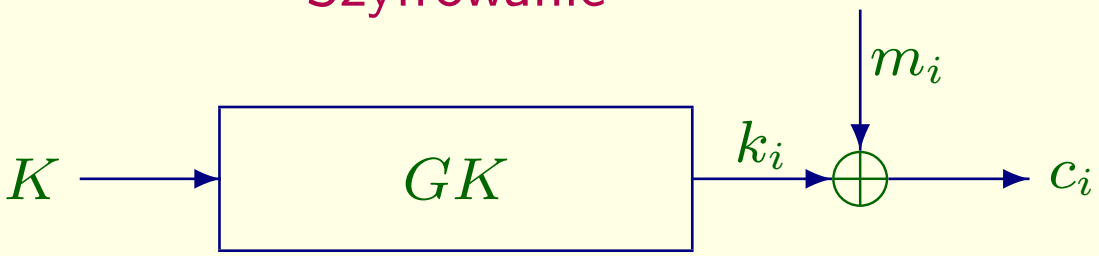
Szyfrowanie

Szyfrowanie

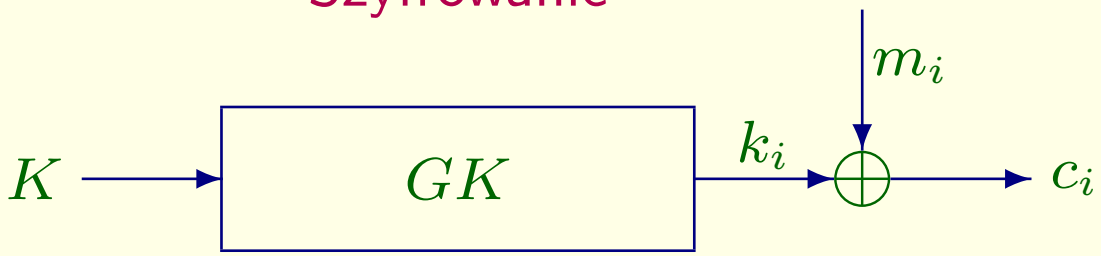
Szyfrowanie

Szyfrowanie

Szyfrowanie

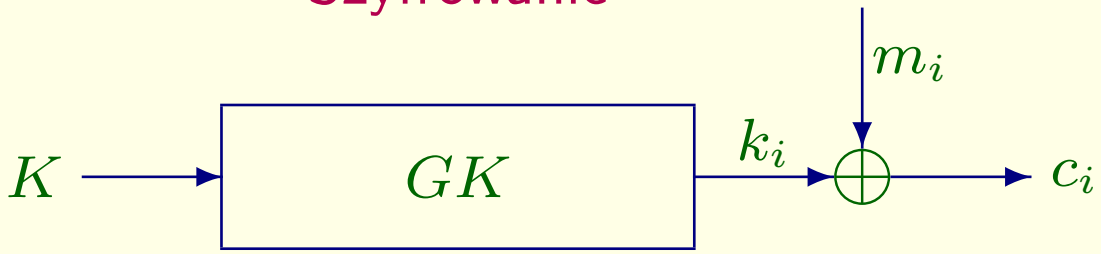


Szyfrowanie



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 $c_i = m_i \oplus k_i$

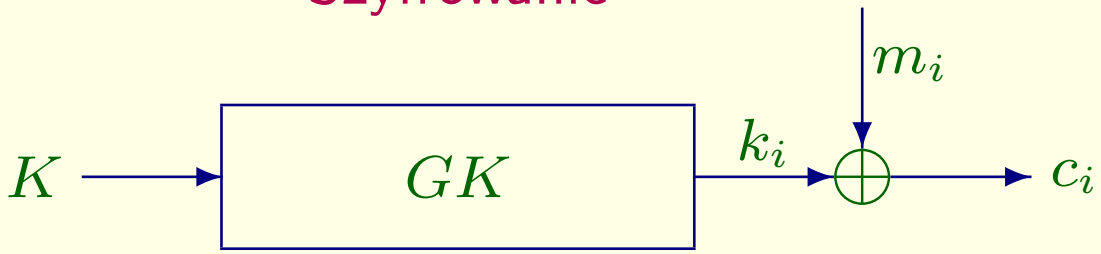
Szyfrowanie



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 $c_i = m_i \oplus k_i$

Deszyfrowanie

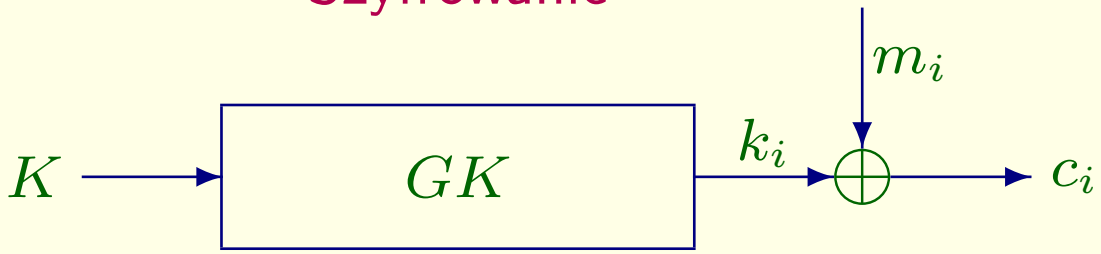
Szyfrowanie



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 $c_i = m_i \oplus k_i$

Deszyfrowanie

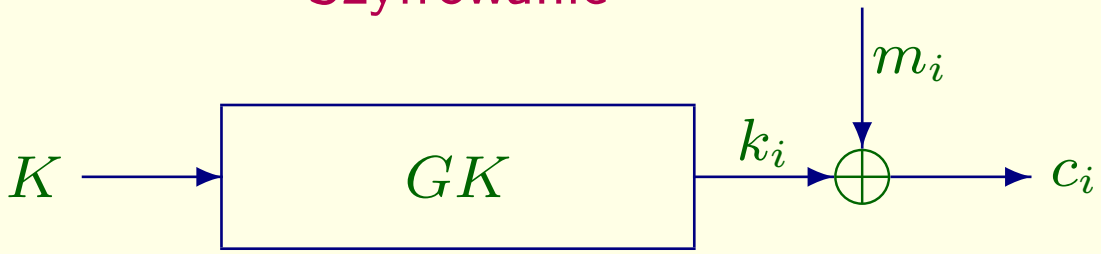
Szyfrowanie



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 $c_i = m_i \oplus k_i$

Deszyfrowanie

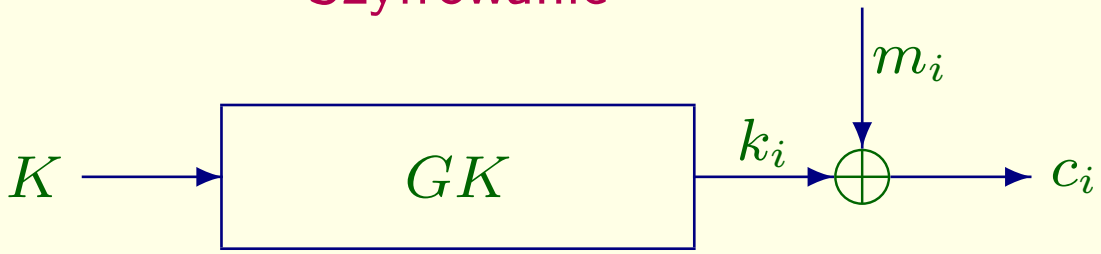
Szyfrowanie



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 $c_i = m_i \oplus k_i$

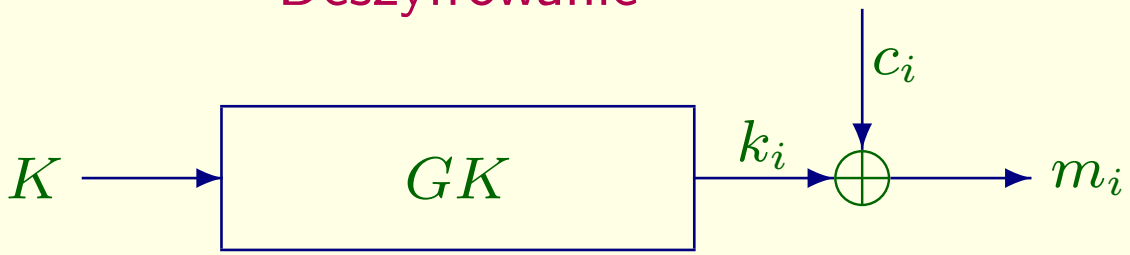
Deszyfrowanie

Szyfrowanie



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 $c_i = m_i \oplus k_i$

Deszyfrowanie



13.2 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).

13.2 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).

13.2 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).

13.2 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).

13.2 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).

13.2 Samosynchronizujące (asynchroniczne) szyfrowanie strumieniowe

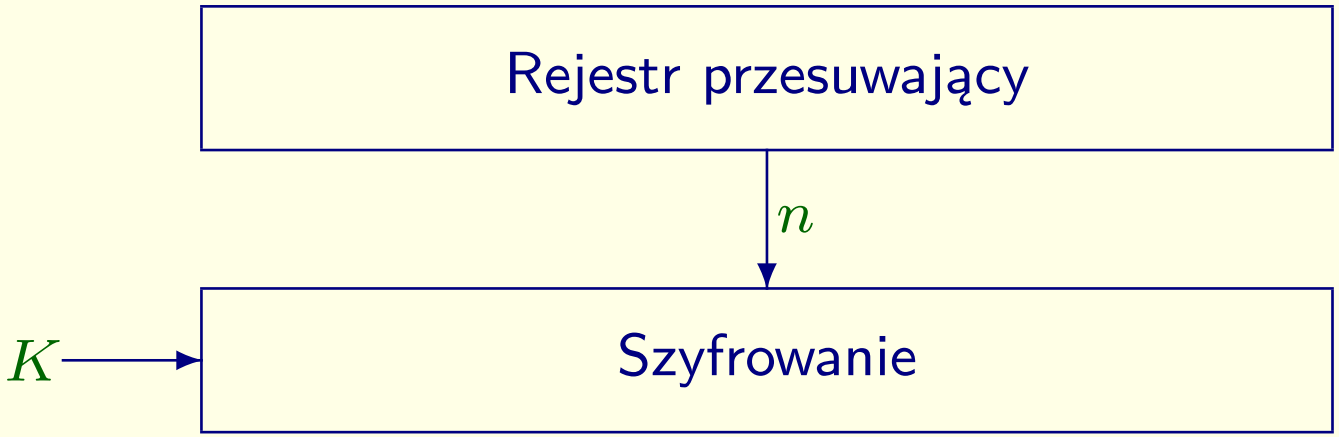
- CFB (Cipher Feedback) z blokiem jednobitowym
- Utrata lub dodanie bitu w kryptogramie powoduje utratę tylko kawałka wiadomości — samosynchronizacja.
- Ograniczona propagacja błędów.
- Zmiana bitu kryptogramu powoduje, że kilka innych bitów będzie deszyfrowanych błędnie — łatwiej wykryć taką zmianę.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

Szyfrowanie

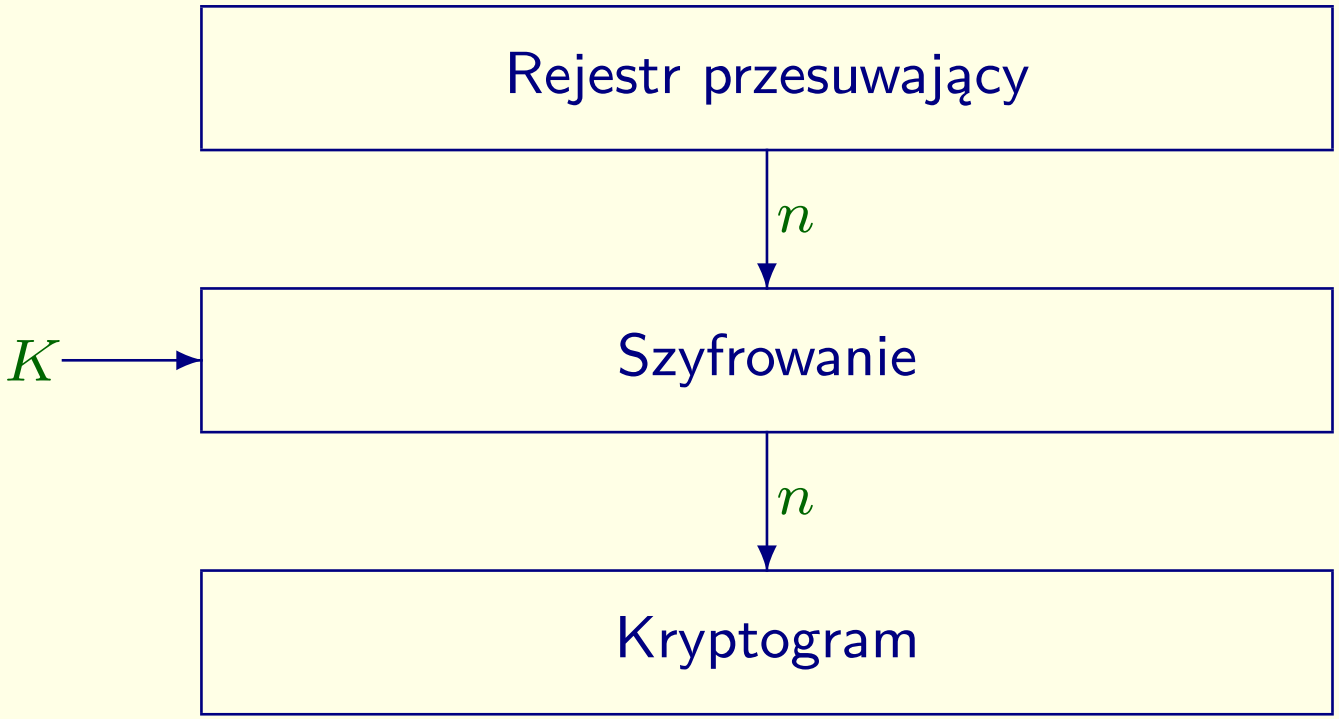
Szyfrowanie

Rejestr przesuwający

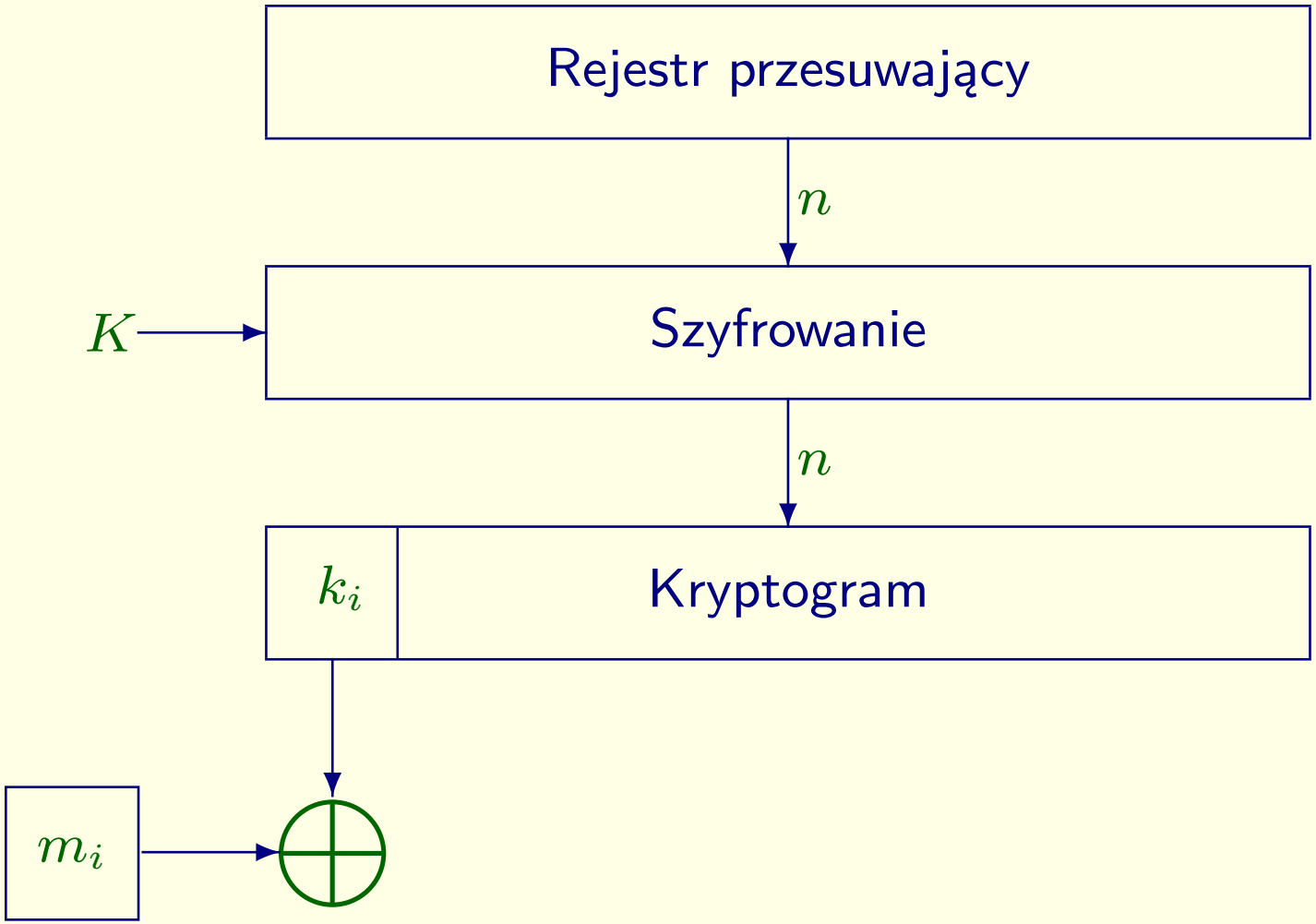
Szyfrowanie



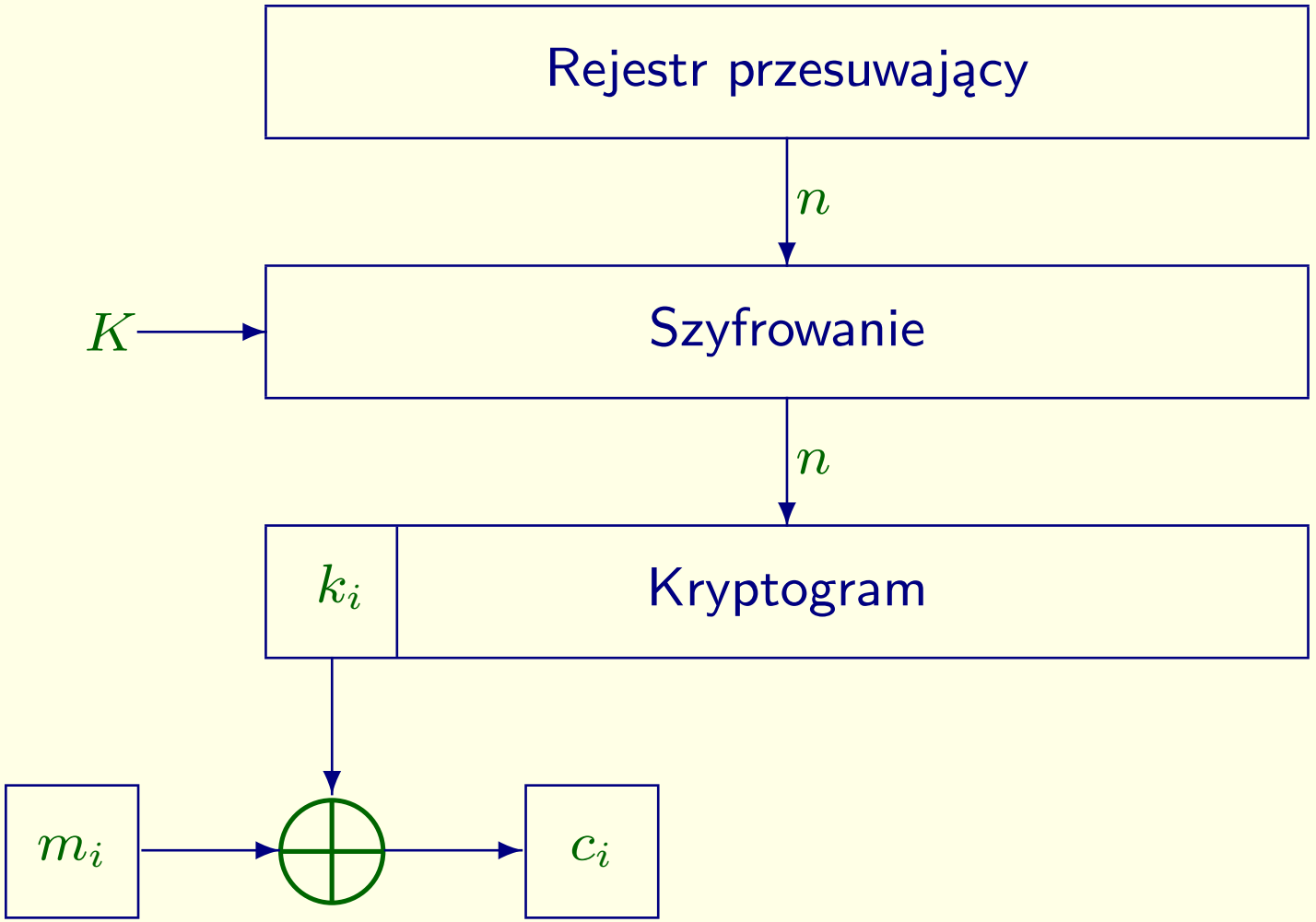
Szyfrowanie



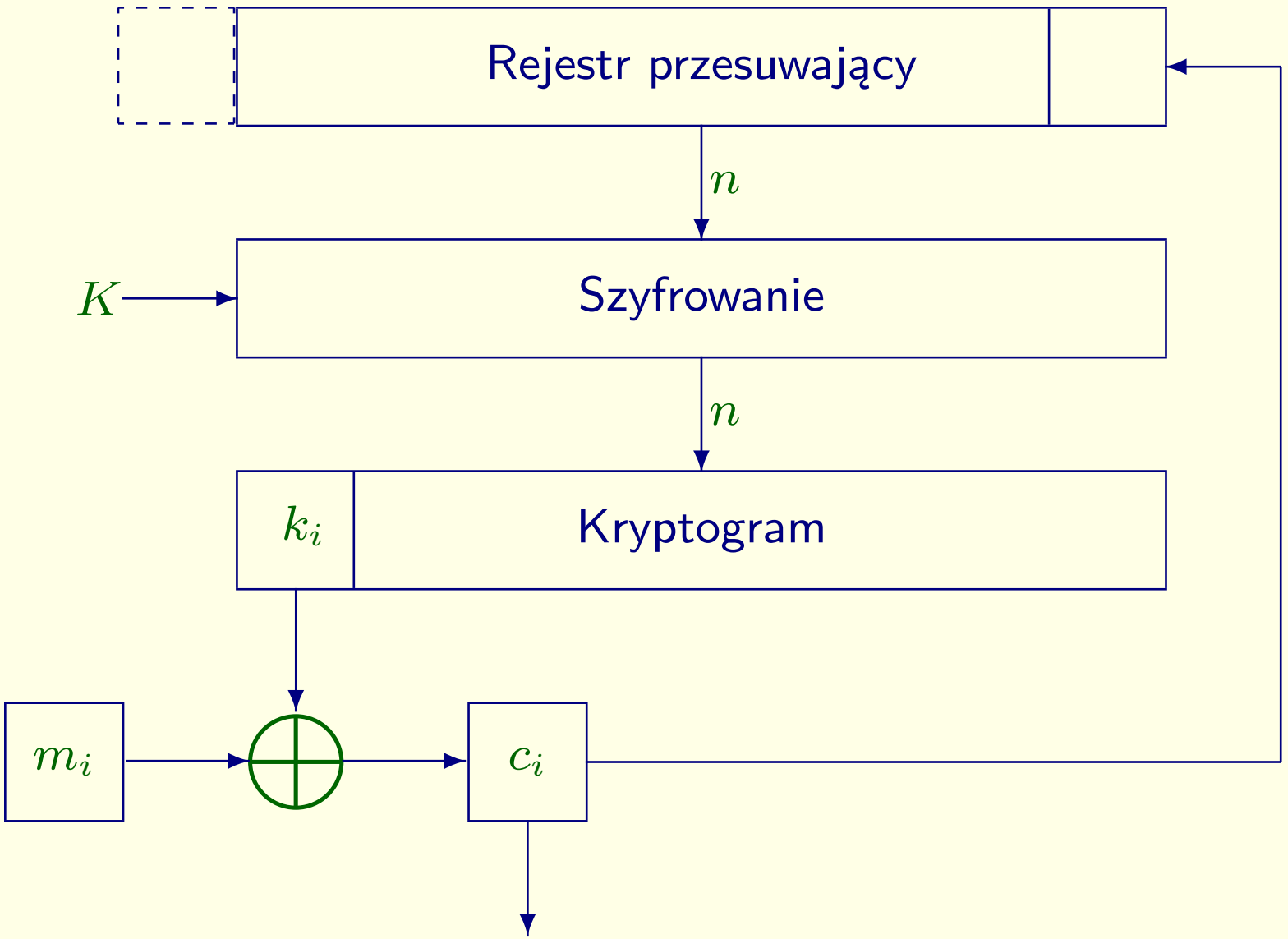
Szyfrowanie



Szyfrowanie



Szyfrowanie

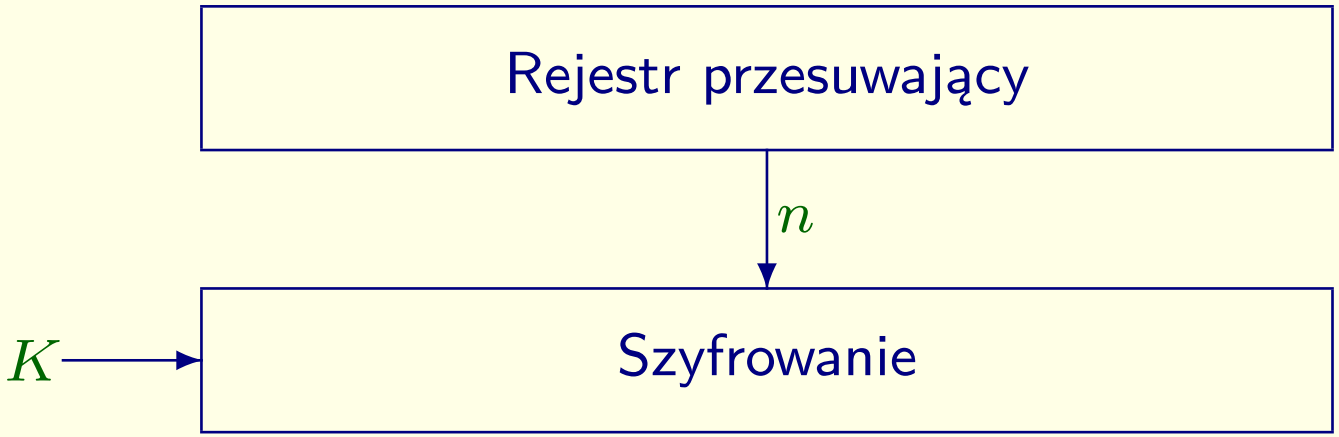


Deszyfrowanie

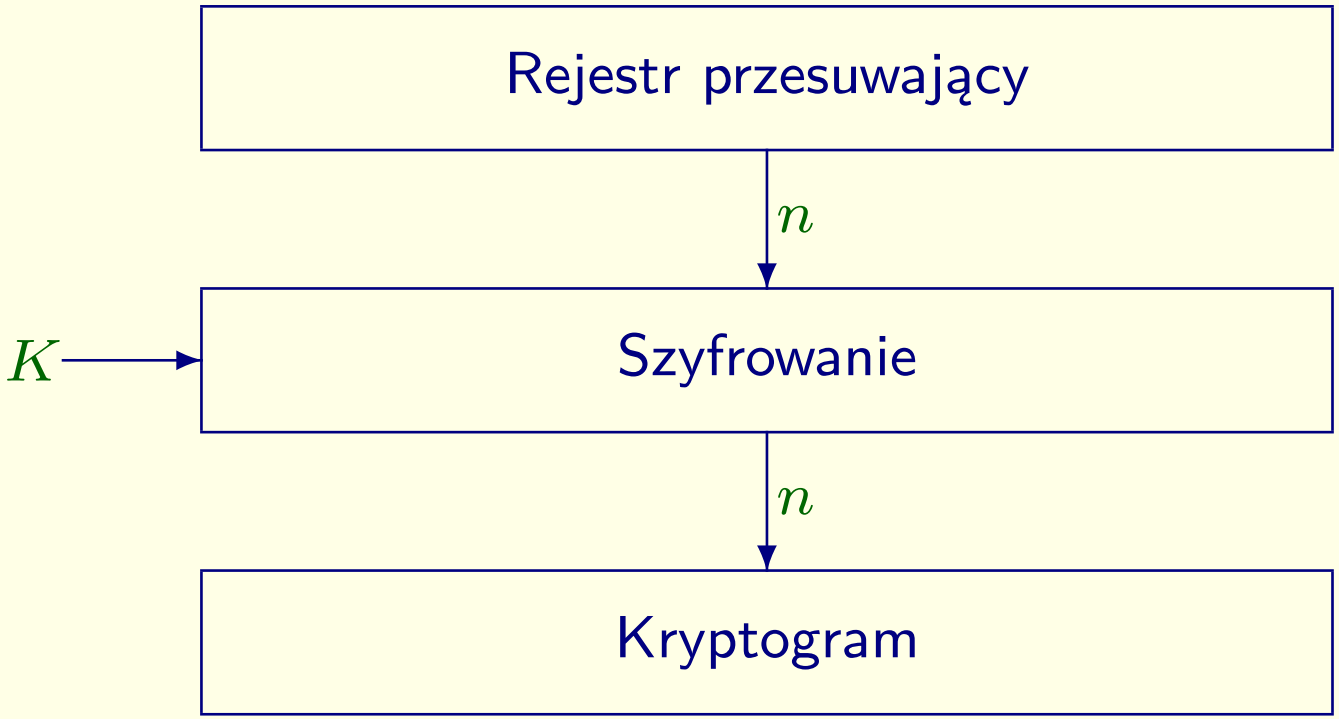
Deszyfrowanie

Rejestr przesuwający

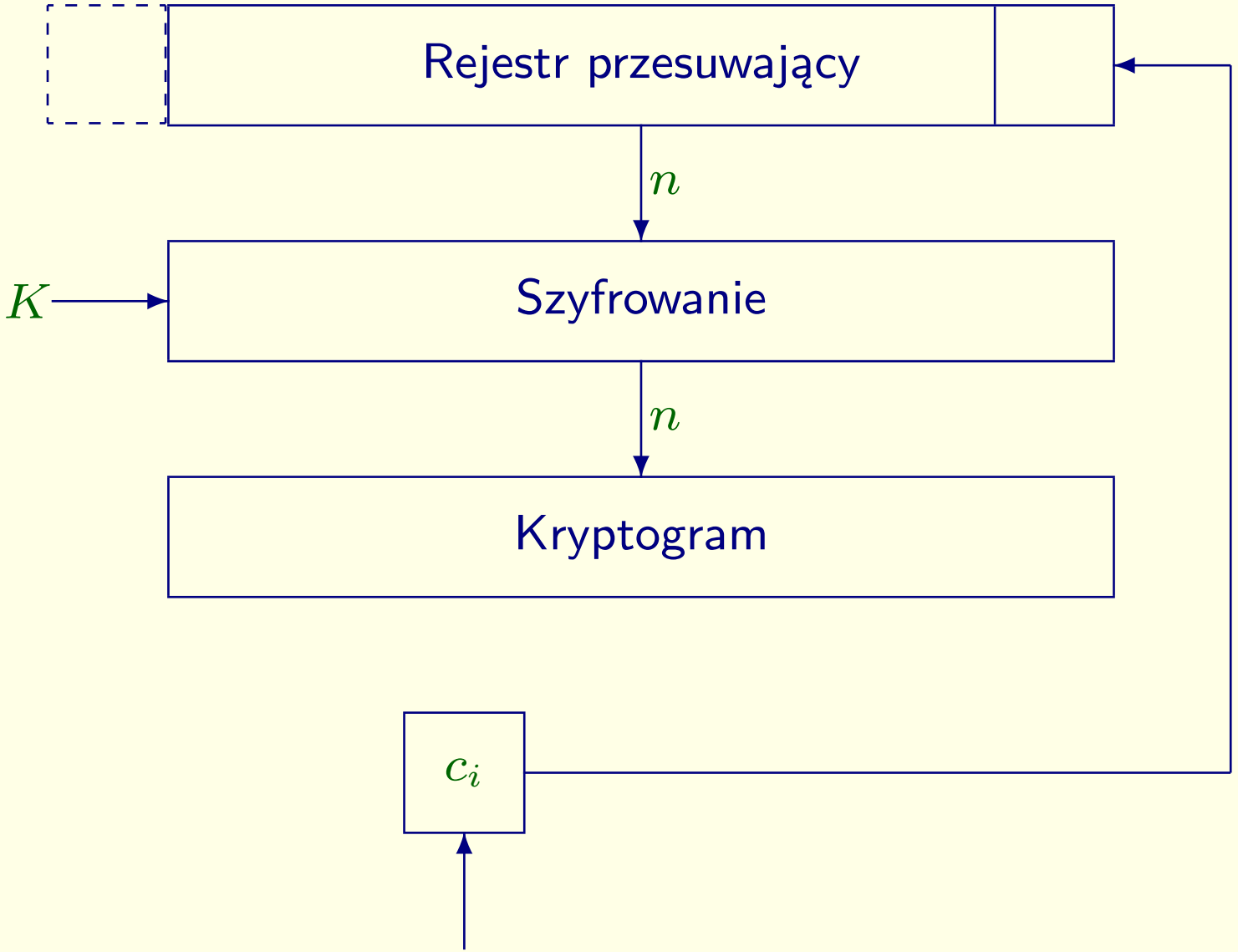
Deszyfrowanie



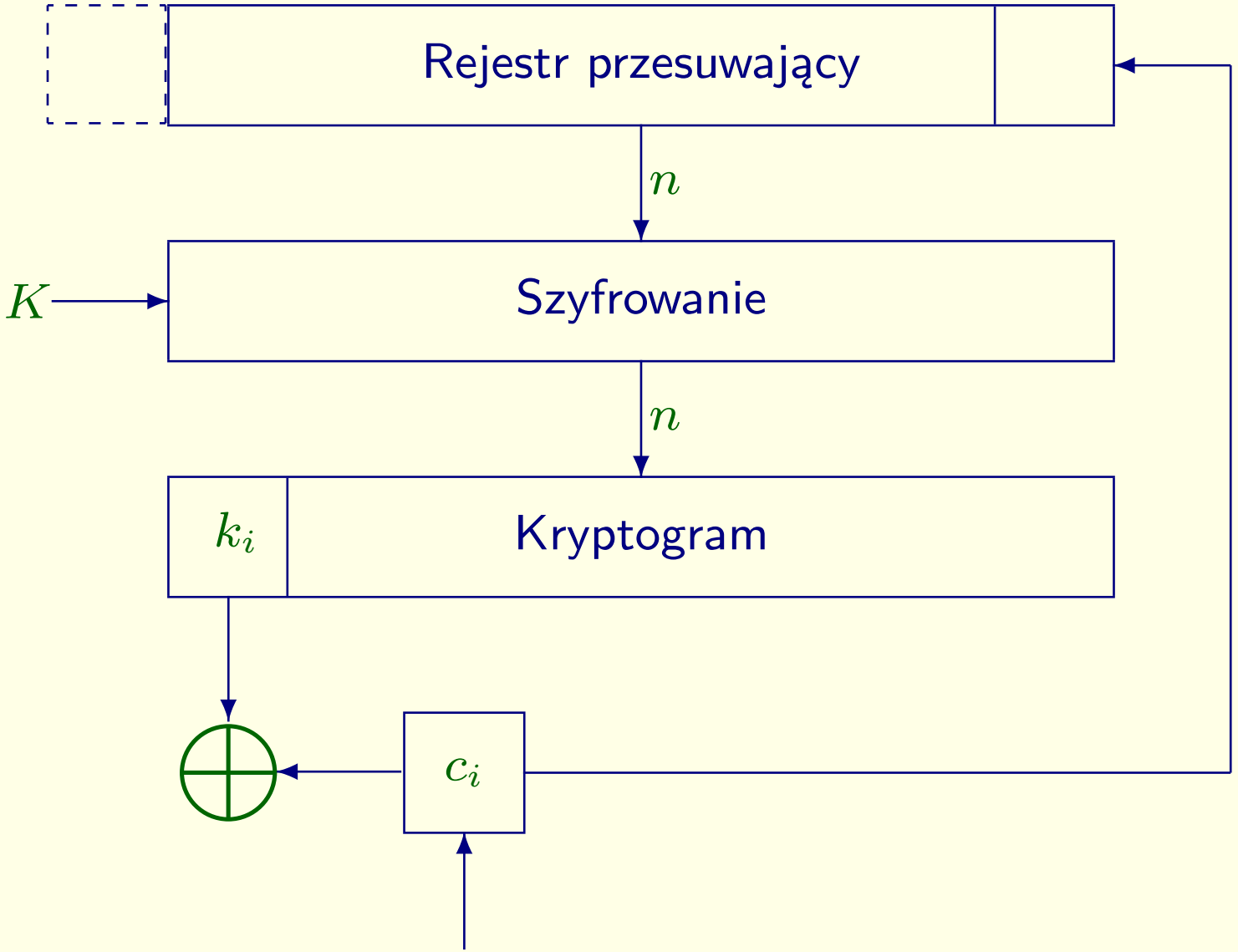
Deszyfrowanie



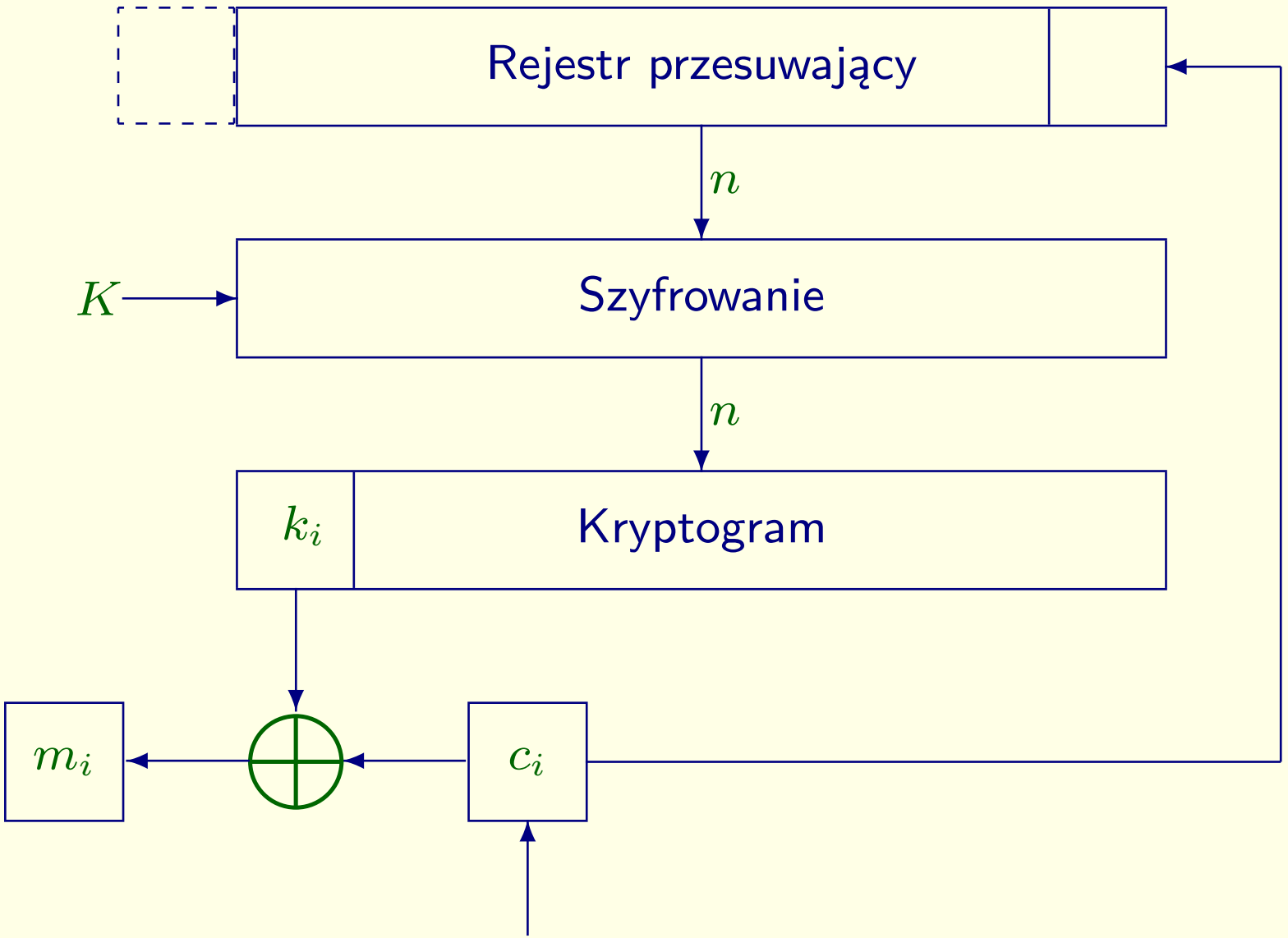
Deszyfrowanie



Deszyfrowanie



Deszyfrowanie



13.3 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.

13.3 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.

13.3 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.

13.4 LFSR — Linear Feedback Shift Register

(Rejestr przesuwający z liniowym sprzężeniem zwrotnym)

- 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.

13.4 LFSR — Linear Feedback Shift Register

(Rejestr przesuwający z liniowym sprzężeniem zwrotnym)

- 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.

13.4 LFSR — Linear Feedback Shift Register

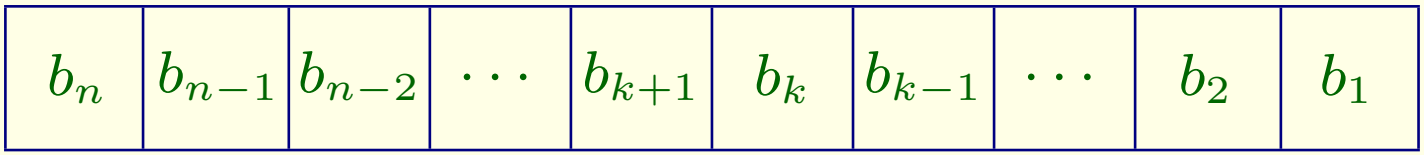
(Rejestr przesuwający z liniowym sprzężeniem zwrotnym)

- 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.

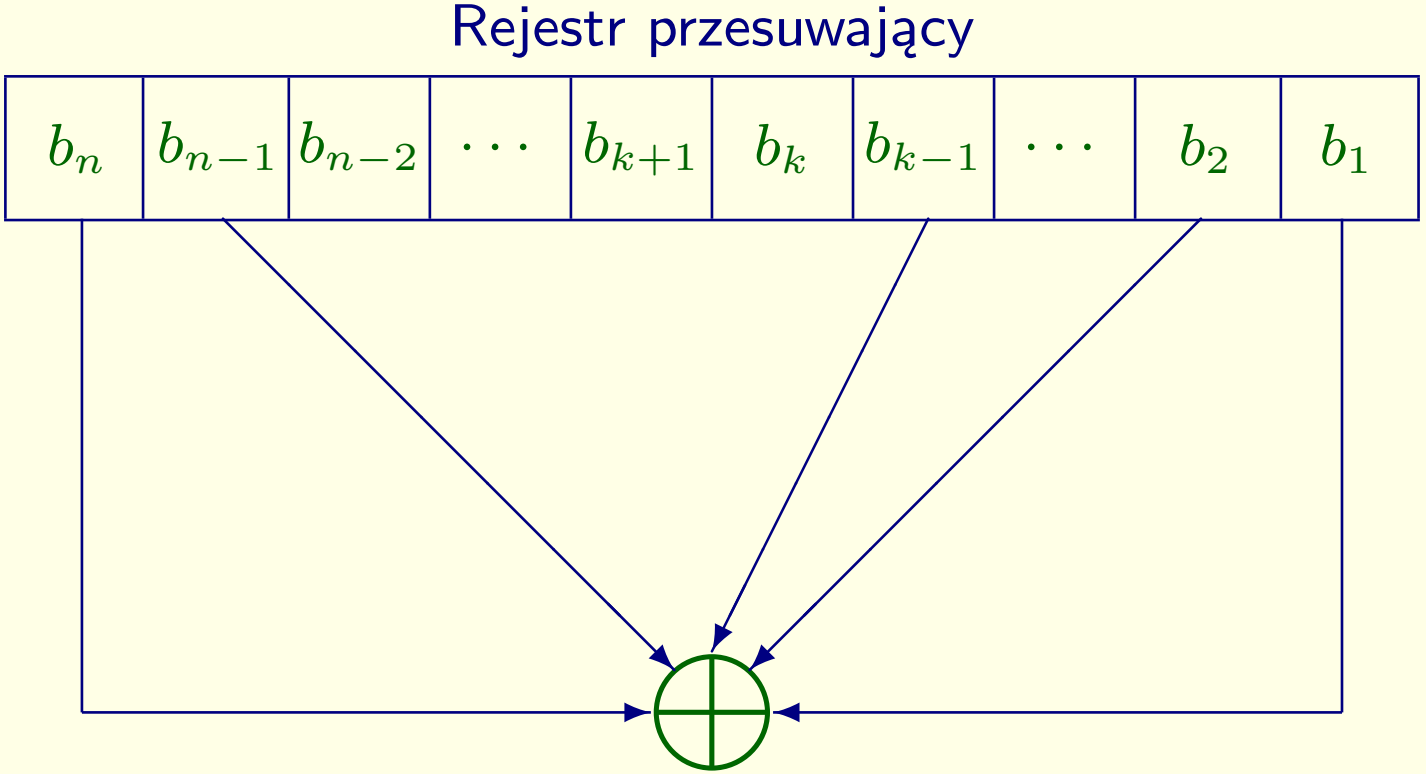
LFSR — Linear Feedback Shift Register

LFSR — Linear Feedback Shift Register

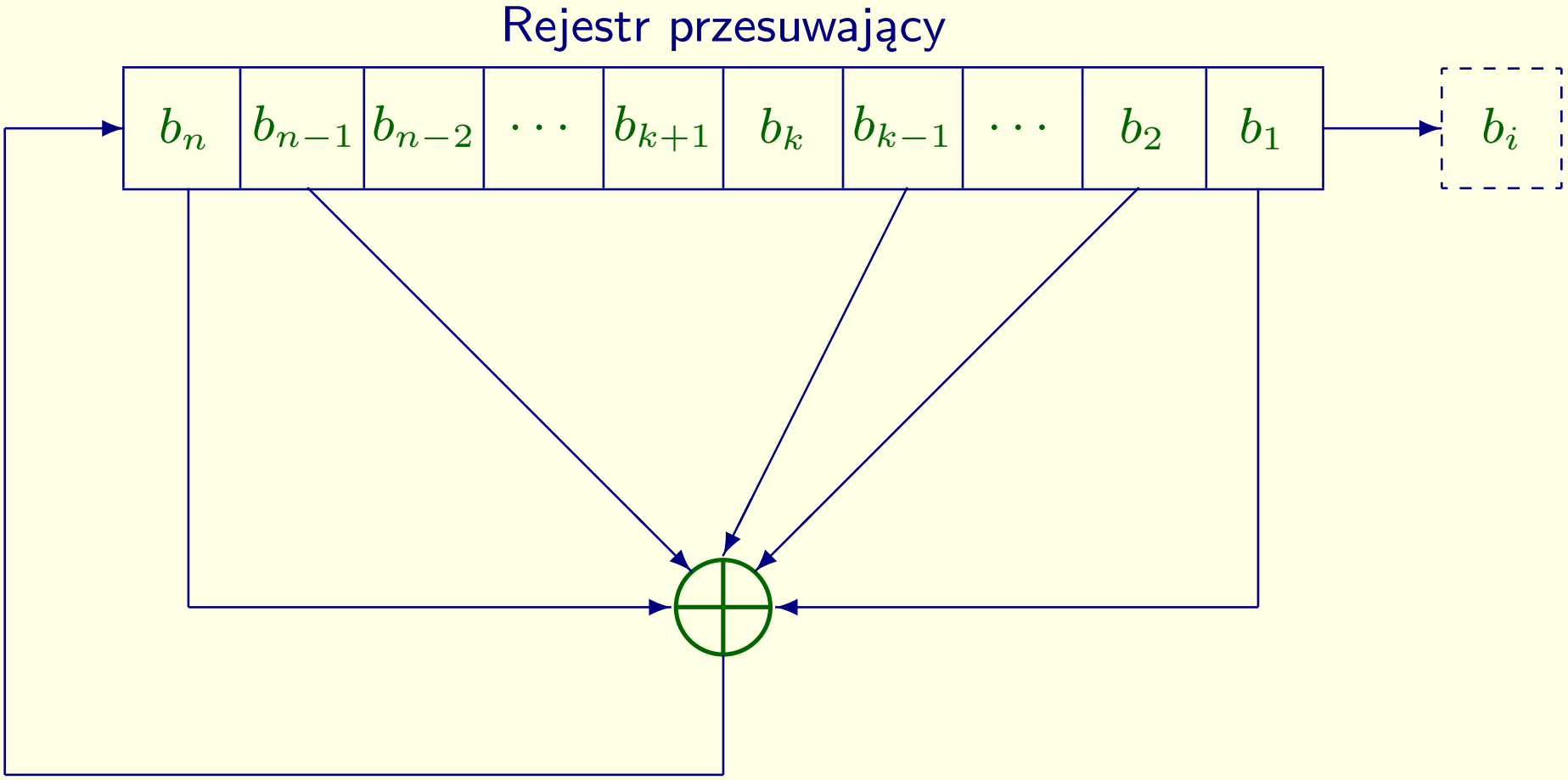
Rejestr przesuwający



LFSR — Linear Feedback Shift Register



LFSR — Linear Feedback Shift Register



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:

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:

$$
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0
\end{array}
$$

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 . . .

czyli:

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

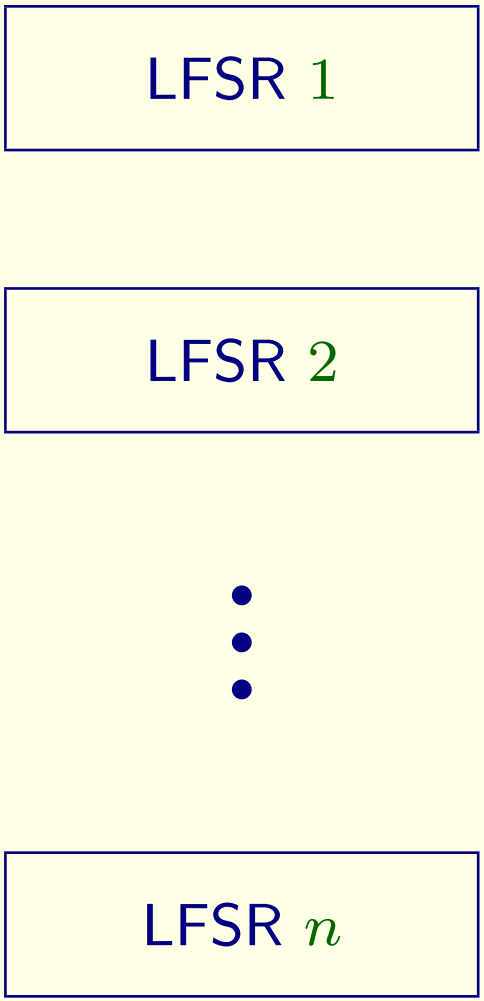
Ponieważ n-bitowy rejestr może znaleźć się w jednym z $2^n - 1$ stanów, więc teoretycznie może on generować ciąg o długości $2^n - 1$ bitów. Potem ciąg się powtarza. (Wykluczamy ciąg samych zer, który daje niekończący się ciąg zer)

- 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.

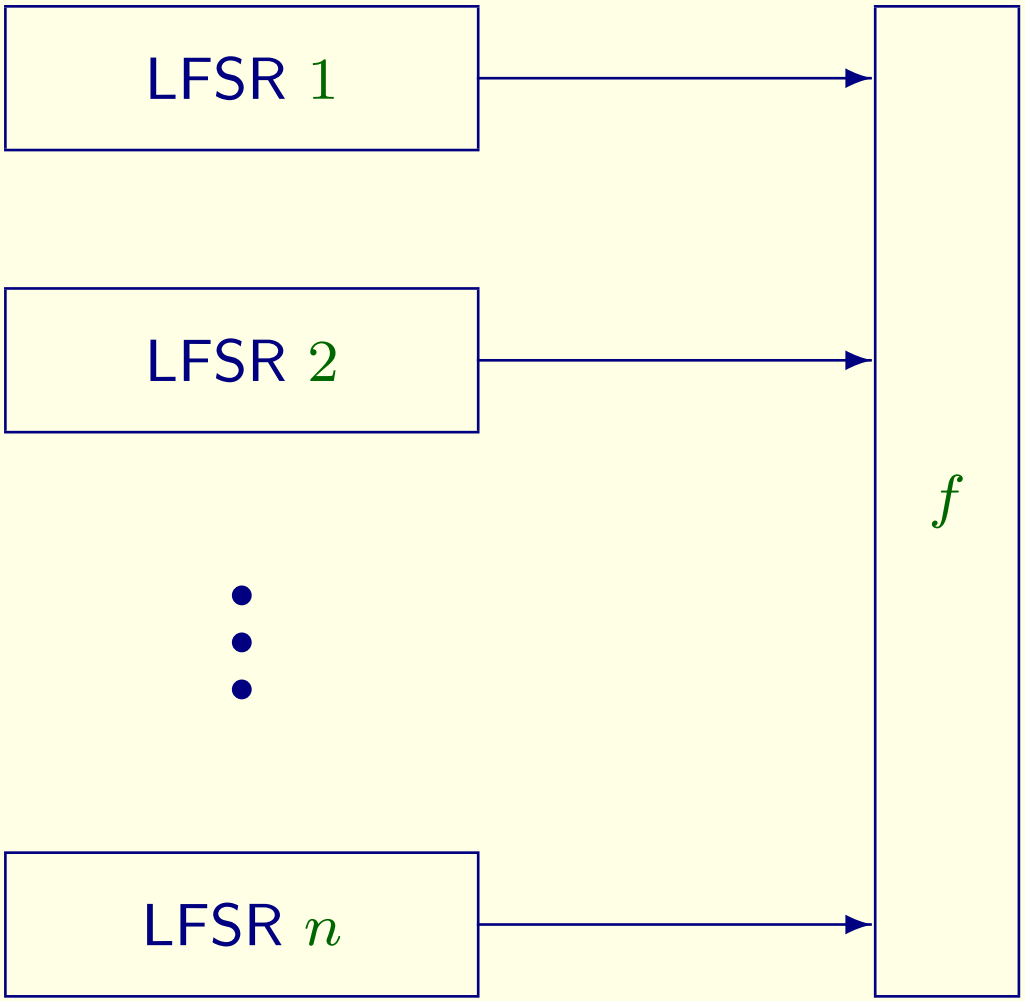
- LFSR ma słabą wartość kryptograficzną gdyż znajomość $2n$ kolejnych bitów ciągu pozwala na znalezienie wartości generowanych od tego miejsca.
- LFSR działa jednak bardzo szybko, zwłaszcza jeśli jest to układ hardware'owy, i stąd jest on bardzo atrakcyjny w praktycznych zastosowaniach. Można konstruować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

Układ kilku LFSR i nieliniowej funkcji f

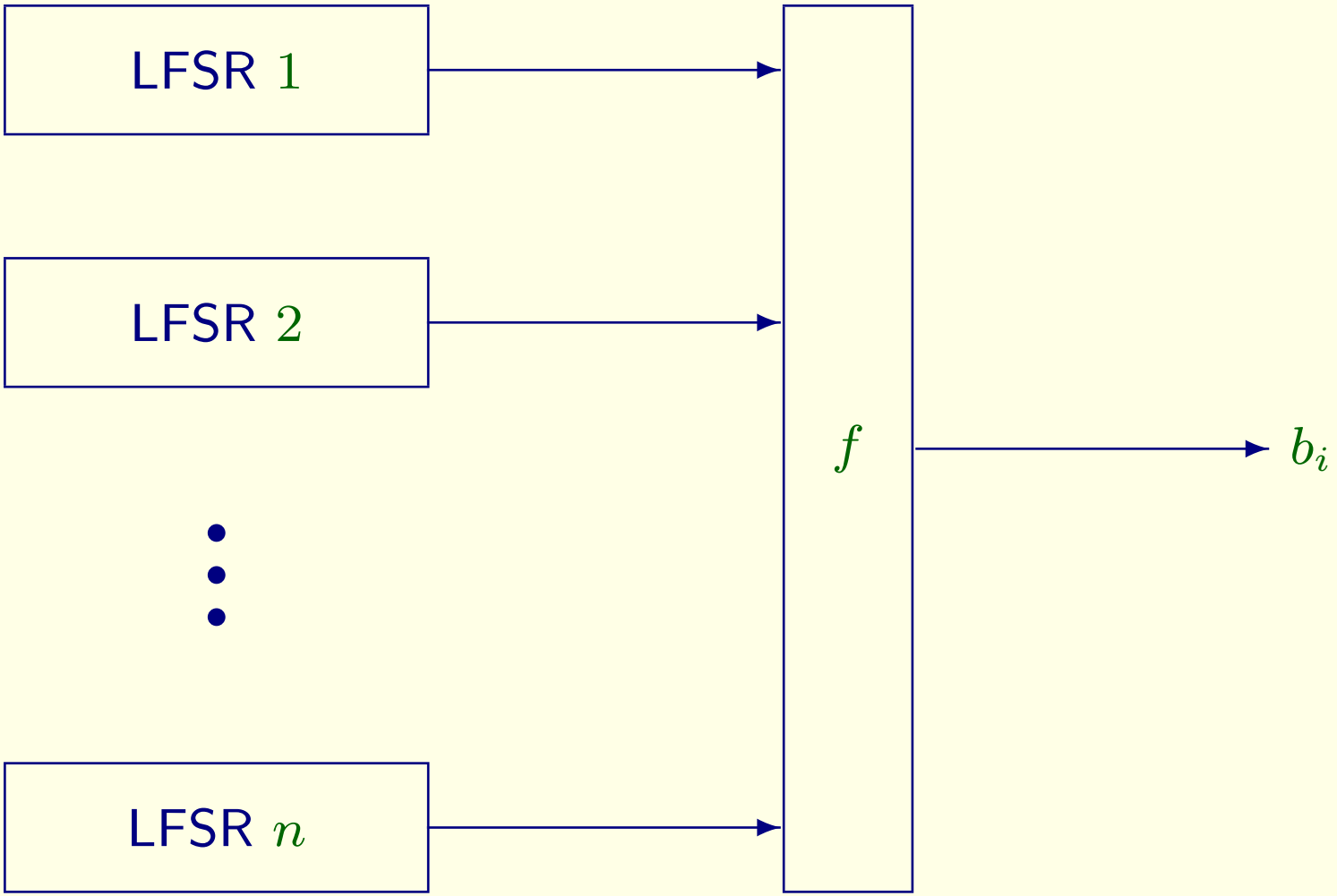
Układ kilku LFSR i nieliniowej funkcji f



Układ kilku LFSR i nieliniowej funkcji f

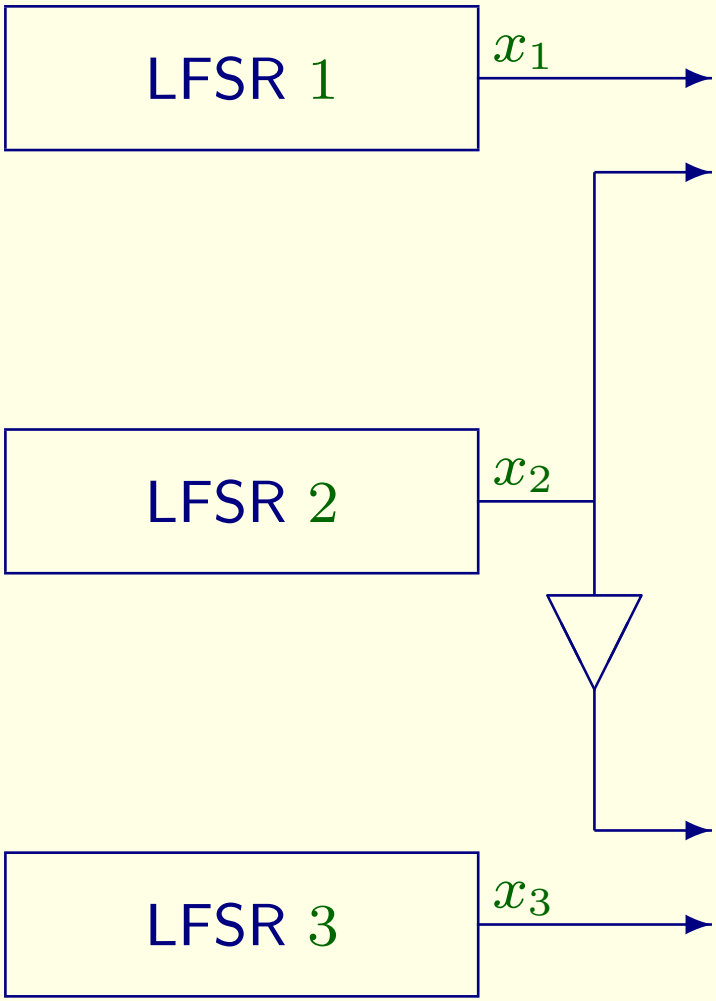


Układ kilku LFSR i nieliniowej funkcji f

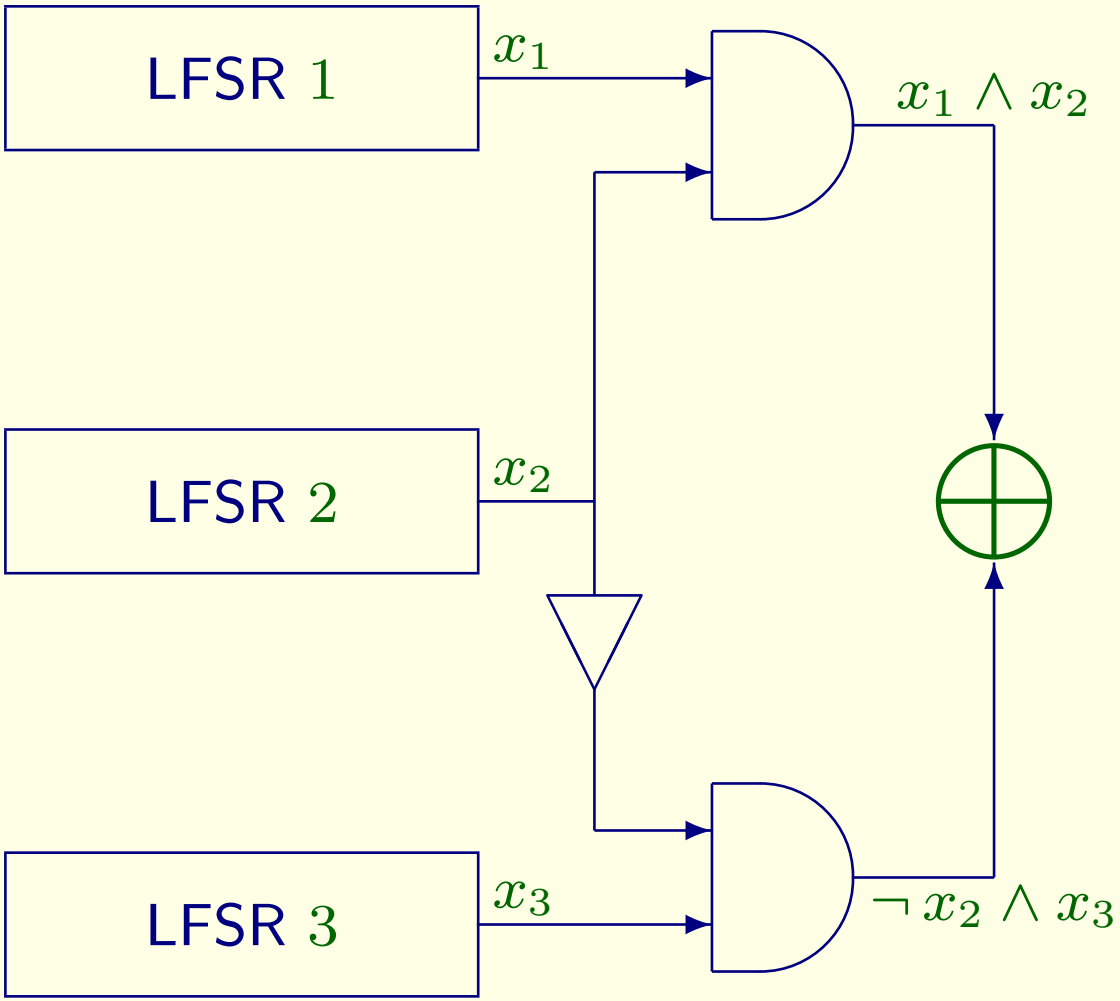


Przykład: Generator Geffe

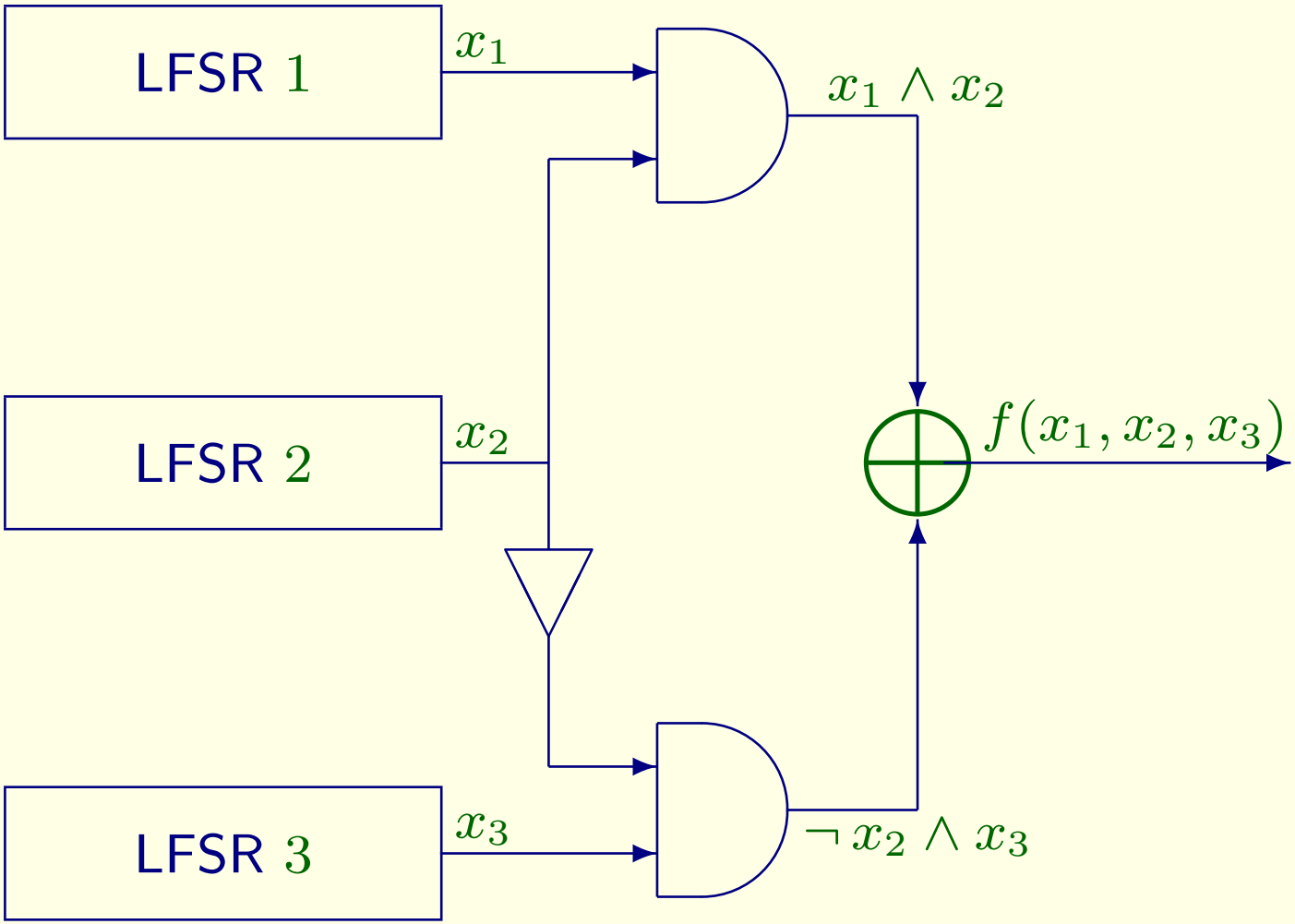
Przykład: Generator Geffe



Przykład: Generator Geffe



Przykład: 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

- 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 $x_3(t)$.

13.5 Generatory sterowane zegarem

Generator o zmiennym kroku, przemienny Stop-and-Go

alternating step generator, Stop-and-Go

13.5 Generatory sterowane zegarem

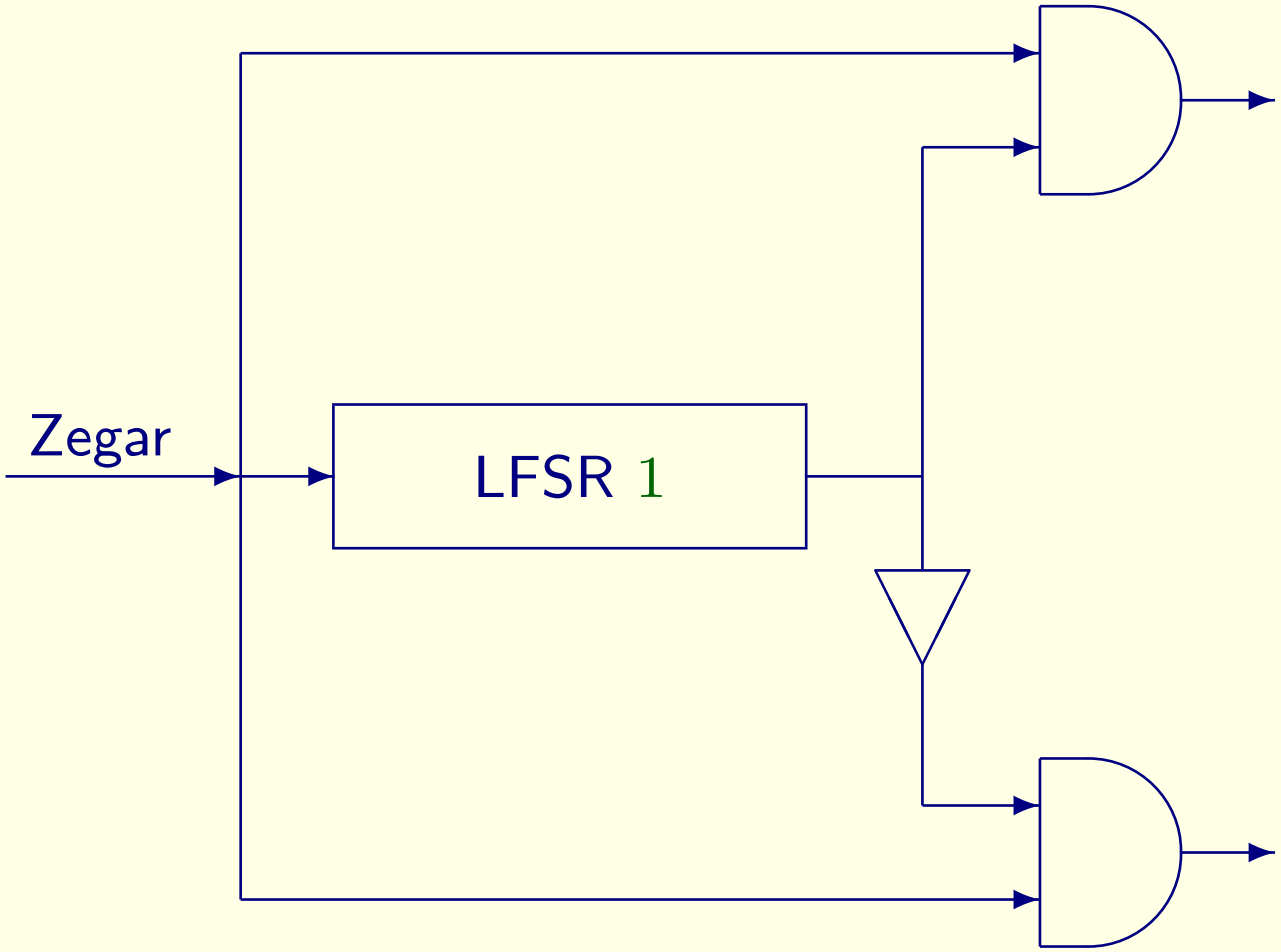
Generator o zmiennym kroku, przemienny Stop-and-Go

alternating step generator, Stop-and-Go

13.5 Generatory sterowane zegarem

Generator o zmiennym kroku, przemienny Stop-and-Go

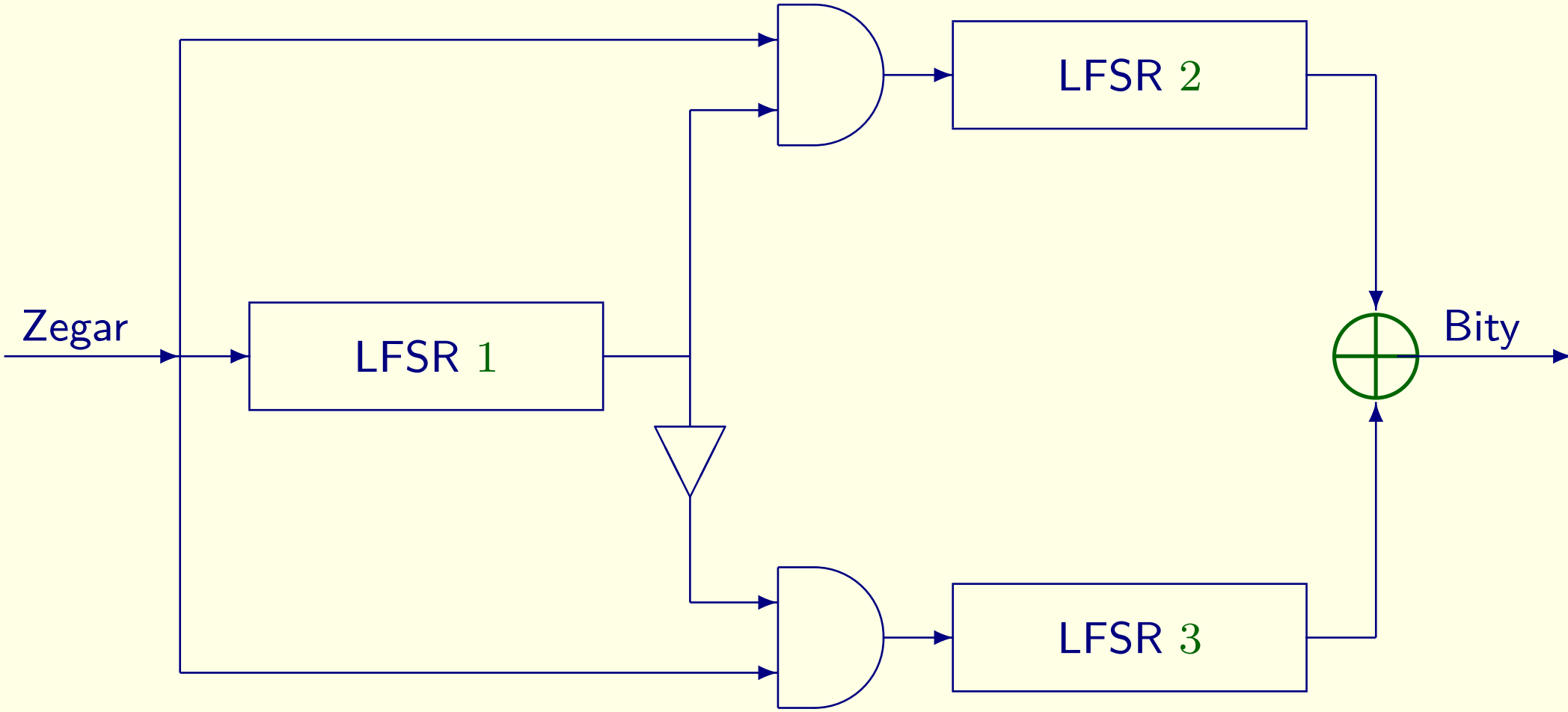
alternating step generator, Stop-and-Go



13.5 Generatory sterowane zegarem

Generator o zmiennym kroku, przemienny Stop-and-Go

alternating step generator, Stop-and-Go



- 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.

- 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.

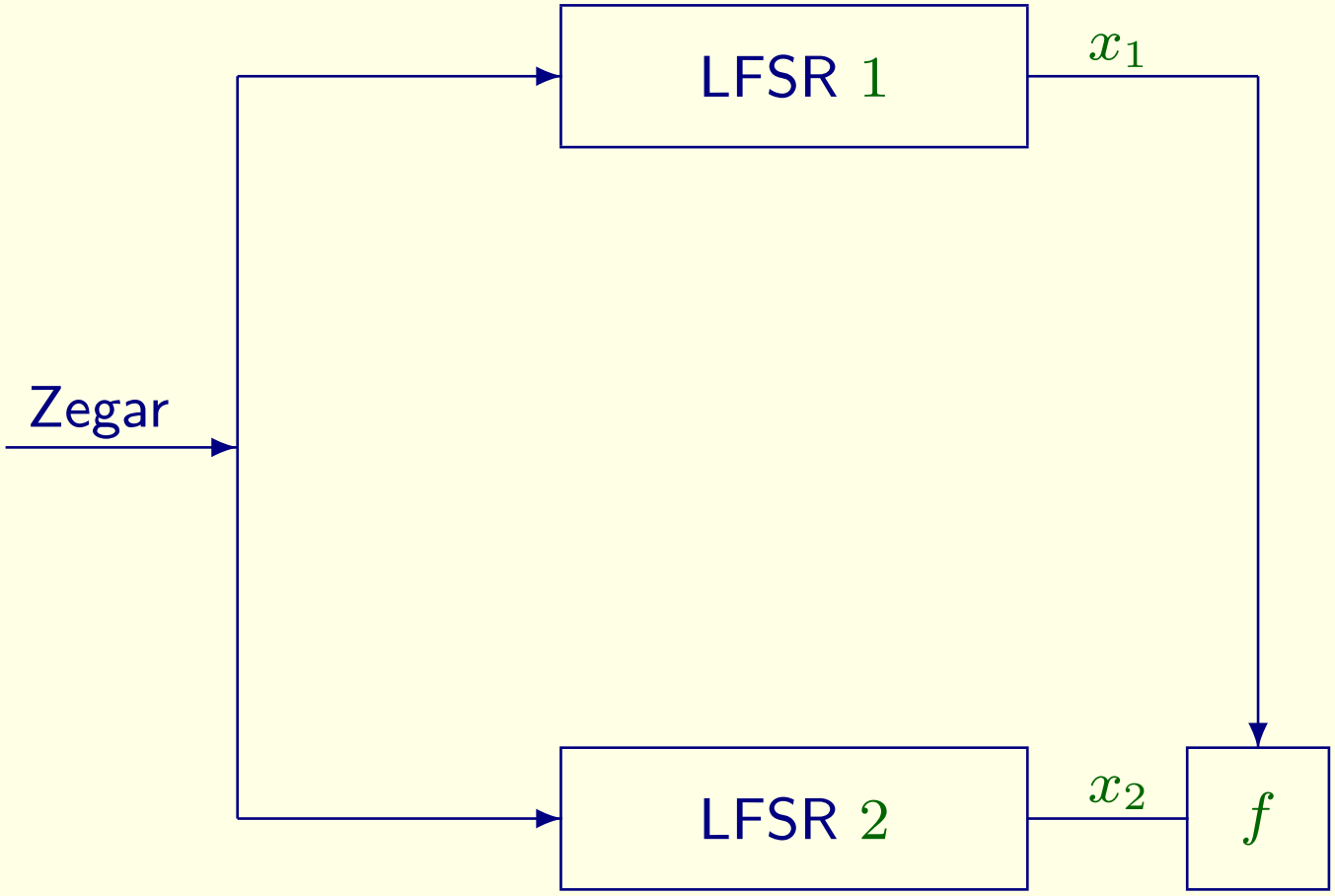
- LFSR 1 jest przesuwany w każdym takcie zegara.
- Jeśli na wyjściu LFSR 1 jest 1 to LFSR 2 jest przesuwany; LFSR 3 nie jest przesuwany (poprzedni bit jest powtarzany).
- Jeśli na wyjściu LFSR 1 jest 0 to LFSR 3 jest przesuwany; LFSR 2 nie jest przesuwany (poprzedni bit jest powtarzany).
- Wyjściowe bity LFSR 2 i LFSR 3 są dodawane modulo 2 ($\oplus$) dając kolejny bit generowanego ciągu.

- 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.

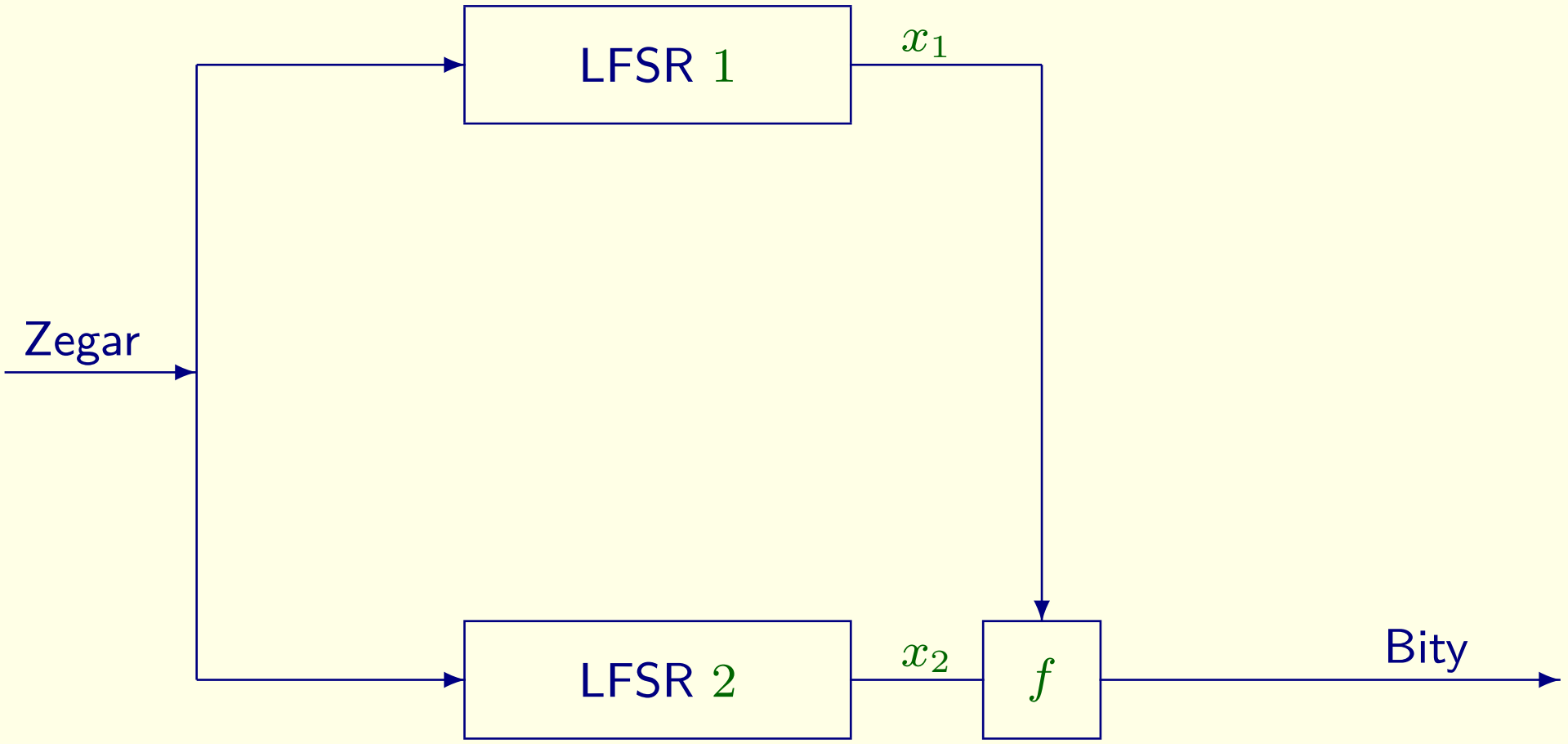
Generator obcinający (shrinking generator)

Generator obcinający (shrinking generator)

Generator obcinający (shrinking generator)



Generator obcinający (shrinking generator)



$$f = \begin{cases} \text{wyślij } x_2 \text{ jeśli } x_1 = 1 \\ \text{opuść } x_2 \text{ jeśli } x_1 = 0 \end{cases}$$

13.6 Generatory, których bezpieczeństwo oparte jest na trudnościach obliczeniowych

13.6.1 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

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

- Pseudolosowy ciąg bitów tworzymy w następujący sposób:

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

13.6 Generatory, których bezpieczeństwo oparte jest na trudnościach obliczeniowych

13.6.1 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

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

- Pseudolosowy ciąg bitów tworzymy w następujący sposób:

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

13.6 Generatory, których bezpieczeństwo oparte jest na trudnościach obliczeniowych

13.6.1 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

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

- Pseudolosowy ciąg bitów tworzymy w następujący sposób:

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

13.6.2 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

13.6.2 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

13.6.2 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

13.6.2 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

13.6.3 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.
- Liczymy

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

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

13.6.3 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.
- Liczymy

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

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

13.6.3 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.
- Liczymy

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

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

13.6.3 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.

- Liczymy

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

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

13.6.3 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.
- Liczymy

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

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

13.7 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.

13.7 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.

13.7 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.

13.7 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

- 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

- 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.

- 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.

- 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.