Projekt KRYS

-

Szyfr blokowy SM4

Gabriela Maciejewska

Sławomir Nikiel

Aleksandra Krawczyk

Politechnika Warszawska,
Wydział Elektroniki i Technik Informacyjnych

27.11.2021

Spis treści

1. Wstęp

Zasada chaosu i zasada dyfuzji to dwie podstawowe zasady projektowania szyfrów blokowych. Dobrze zaprojektowany algorytm szyfru blokowego powinien opierać się na kryptograficznie uzasadnionej strukturze transformacji podstawowej. Kryptograficzne właściwości transformacji podstawowej decydują o efektywności wynikowej transformacji szyfrującej. Algorytm SM4 jest zbudowany na permutacji ortomorficznej. Jego transformacja rundy jest permutacją ortomorficzną, a jego właściwości kryptograficzne można wydedukować z właściwości tej permutacji.

2. Rys historyczny

Algorytm „SMS4" został wymyślony przez Shu-Wang Lu. Po raz pierwszy został opublikowany w 2003 r. jako część "Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", a następnie opublikowany niezależnie w 2006 r. przez SCA(State Cryptography Administration of China) (wówczas OSCCA - Office of State Commercial Cryptography Administration) jako SMS4 - Cryptographic Algorithm For Wireless LAN Products. Został opublikowany jako branżowy standard kryptograficzny i przemianowany na „SM4" w 2012 r. przez SCA, a ostatecznie sformalizowany w 2016 roku jako Chiński Standard Narodowy - "Information security technology – SM4 block cipher algorithm".

SM4 został pierwotnie stworzony do użytku w ochronie sieci bezprzewodowych i jest zgodny z chińskim National Standard for Wireless LAN WAPI (Wired Authentication and Privacy Infrastructure), czyli alternatywą dla mechanizmów bezpieczeństwa określonych w IEEE 802.11i. Został przedłożony Międzynarodowej Organizacji Normalizacyjnej ISO przez Chińskie Stowarzyszenie Normalizacyjne SAC.

Zarówno WAPI, jak i IEEE 802.11i zostały zaproponowane jako poprawki bezpieczeństwa do normy ISO/IEC 8802-11. Oba schematy wykorzystują dwa różne szyfry blokowe do szyfrowania danych: - WAPI wykorzystuje szyfr blokowy SMS4, podczas gdy IEEE 802.11i używa szyfru blokowego AES.

W marcu 2006 roku IEEE 802.11i został zatwierdzony jako ISO/IEC 8802-11 WLAN, natomiast WAPI został częściowo odrzucony, ze względu na nieujawniomy szyfr SMS4. Jednakże, ponieważ WAPI jest nadal oficjalnie wymagany dla chińskiego standardu krajowego, jest nadal używany w chińskiej branży WLAN i wielu międzynarodowych korporacjach, takich jak SONY, które wspierają WAPI w odpowiednich produktach.

Najnowszy standard SM4 został zaproponowany przez SCA (wówczas OSCCA), znormalizowany przez TC 260 Administracji Normalizacyjnej Chińskiej Republiki Ludowej (SAC) i został opracowany przez następujące osoby w Centrum Badań Danych i Bezpieczeństwa Komunikacji (Centrum DAS) Chińskiej Akademii Nauk, Chińskie Centrum Testowania Kryptografii Komercyjnej oraz Pekińska Akademia Informatyki i Technologii (BAIST):

— Shu-Wang Lu
— Dai-Wai Li
— Kai-Yong Deng
— Chao Zhang
— Peng Luo
— Zhong Zhang
— Fang Dong
— Ying-Ying Mao
— Zhen-Hua Liu

SM4 został również ostatecznie znormalizowany w ISO/IEC 8802-11 przez Międzynarodową Organizację Normalizacyjną w 2017 r.

3. Opis szyfru[4]

SM4 to chiński standard szyfrowania blokowego, wprowadzony do ochrony sieci bezprzewodowych i wydany w styczniu 2006 roku. Wejście, wyjście i klucz SM4 mają 128 bitów. Algorytm ma 32 rundy, z których każda modyfikuje jedno z czterech 32-bitowych słów. Szyfrowanie i deszyfrowanie ma tę samą strukturę, z wyjątkiem tego, że klucz do deszyfrowania to odwrotność klucza do szyfrowania.

3.1. Definicje

Następujące definicje są kluczowe do opisania struktury szyfru SM4.

3.1.1. Słowo i bajt

Zdefiniujmy Z_2^e jako zbiór e-bitowych wektorów. Słowo wykorzystywane w algorytmie szyfrowania SM4 jest elementem Z_2^{32}, natomiast bajt elementem Z_2^8.

3.1.2. Skrzynka podstawieniowa (ang. sbox)

Skrzynka podstawieniowa, przyjmuje 8 bitów na wejściu i zwraca 8 bitów na wyjściu.

3.1.3. Podstawowe operacje

Podstawowymi operacjami bitowymi używanymi w omawianym algorytmie są:
1. $\oplus$ — bitowy XOR dwóch wektorów z Z_2^{32}
2. $<<< i$ — rotacja w lewo o i bitów

3.1.4. Wejście, wyjście i klucz

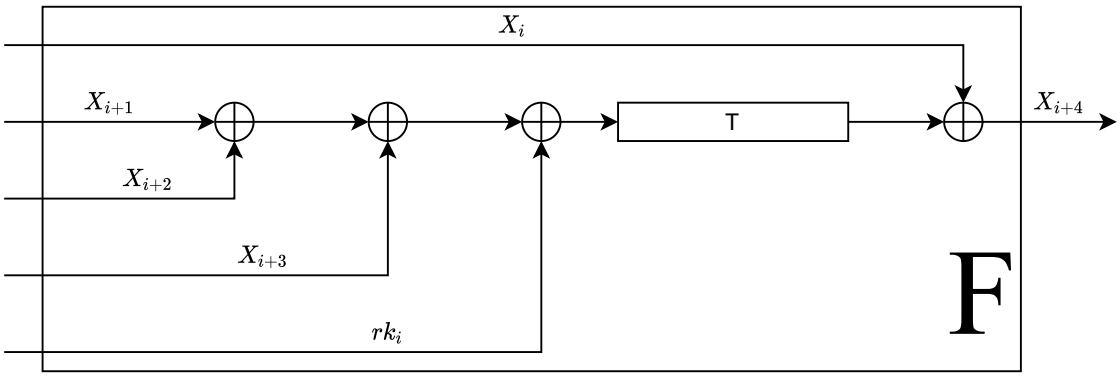
Tekst jawny to 128-bitowy blok składający się z czterech 32-bitowych słów $X = (X_0, X_1, X_2, X_3)$. Szyfrogram to analogiczny do tekstu jawnego 128-bitowy blok składający się z czterech 32-bitowych słów $Y = (Y_0, Y_1, Y_2, Y_3)$.

Klucz podawany do szyfru również zachowuje tę samą konwencję $MK = (MK_0, MK_1, MK_2, MK_3)$. Używany jest do generowania zestawu kluczy rund $(rk_0, rk_1, \ldots, rk_{31})$, gdzie $rk_i \in Z_2^{32}$.

3.2. Funkcja rundy F

Niech blok $(X_0, X_1, X_2, X_3) \in (Z_2^{32})^4$ będzie 128-bitowym wejściem oraz $rk \in Z_2^{32}$ kluczem rundy, wówczas funkcja $F : (Z_2^{32})^5 \to Z_2^{32}$:

$$F(X_0, X_1, X_2, X_3, rk) = X_0 \oplus T(X_1 \oplus X_2 \oplus X_3 \oplus rk)$$



Rys. 1. Blok F

3.2.1. Mieszane podstawienie T

$T : Z_2^{32} \to Z_2^{32}$ jest odwracalną funkcją podstawienia. Składa się z nieliniowego podstawienia τ oraz liniowego L.

$$T(\ldots) = L(\tau(\ldots))$$

Nieliniowe podstawienie τ

Niech A będzie 32-bitowym słowem wejściowym, czyli $A = (a_0, a_1, a_2, a_3) \in (Z_2^8)^4$. Wówczas $\tau : (Z_2^8)^4 \rightarrow (Z_2^8)^4$ definiujemy następująco:

$$\tau(A) = (Sbox(a_0), Sbox(a_1), Sbox(a_2), Sbox(a_3))$$

Liniowe podstawienie L

Niech B, będzie wynikiem otrzymanym z funkcji τ. Wówczas funkcja $L : Z_2^{32} \rightarrow Z_2^{32}$ definiowana jest następująco:

$$L(B) = B \oplus (B <<< 2) \oplus (B <<< 10) \oplus (B <<< 18) \oplus (B <<< 24)$$

3.2.2. Skrzynka podstawieniowa

Działanie skrzynki podstawieniowej opisuje tabela 1. Wartości przedstawione są w systemie heksadecymalnym.

Tabela 1. Skrzynka podstawieniowa

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	d6	90	e9	fe	cc	e1	3d	b7	16	b6	14	c2	28	fb	2c	05
1	2b	67	9a	76	2a	be	04	c3	aa	44	13	26	49	86	06	99
2	9c	42	50	f4	91	ef	98	7a	33	54	0b	43	ed	cf	ac	62
3	e4	b3	1c	a9	c9	08	e8	95	80	df	94	fa	75	8f	3f	a6
4	47	07	a7	fc	f3	73	17	ba	83	59	3c	19	e6	85	4f	a8
5	68	6b	81	b2	71	64	da	8b	f8	eb	0f	4b	70	56	9d	35
6	1e	24	0e	5e	63	58	d1	a2	25	22	7c	3b	01	21	78	87
7	d4	00	46	57	9f	d3	27	52	4c	36	02	e7	a0	c4	c8	9e
8	ea	bf	8a	d2	40	c7	38	b5	a3	f7	f2	ce	f9	61	15	a1
9	e0	ae	5d	a4	9b	34	1a	55	ad	93	32	30	f5	8c	b1	e3
a	1d	f6	e2	2e	82	66	ca	60	c0	29	23	ab	0d	53	4e	6f
b	d5	db	37	45	de	fd	8e	2f	03	ff	6a	72	6d	6c	5b	51
c	8d	1b	af	92	bb	dd	bc	7f	11	d9	5c	41	1f	10	5a	d8
d	0a	c1	31	88	a5	cd	7b	bd	2d	74	d0	12	b8	e5	b4	b0
e	89	69	97	4a	0c	96	77	7e	65	b9	f1	09	c5	6e	c6	84
f	18	f0	7d	ec	3a	dc	4d	20	79	ee	5f	3e	d7	cb	39	48

Przykładowo dla wejścia '3f' odczytujemy 3-ci rząd i f-ą kolumnę: $Sbox('3f') = a6$. Skrzynka podstawieniowa bazuje na operacji odwracania nad ciałem $GF(2^8)$ — Ciało Galois.

3.3. Szyfrowanie i deszyfrowanie

Niech R będzie odwrotnym podstawieniem:

$$R(A_0, A_1, A_2, A_3) = (A_3, A_2, A_1, A_0), A_i \in Z_2^{32}, i = 0, 1, 2, 3$$

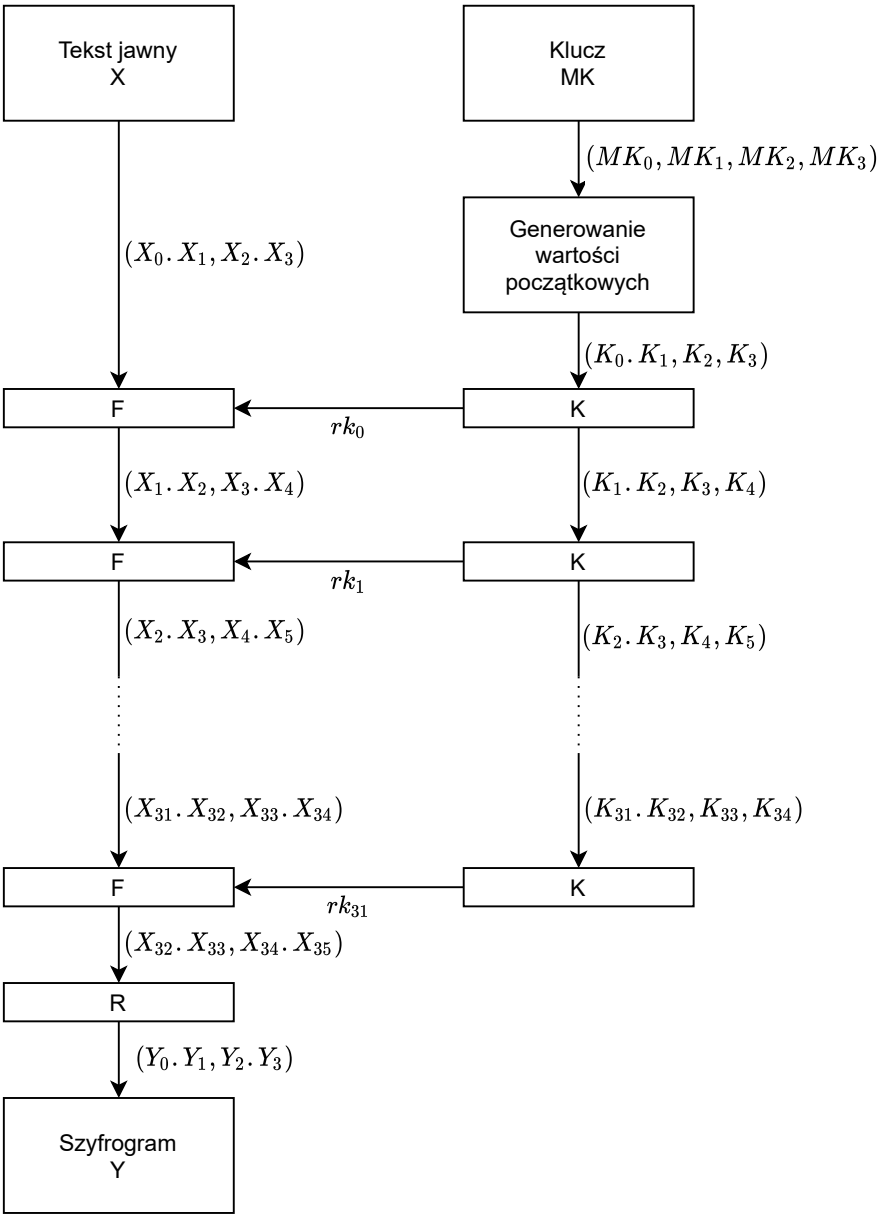
Weźmy tekst jawny $(X_0, X_1, X_2, X_3) \in (Z_2^{32})^4$, oznaczmy jego szyfrogram jako $(Y_0, Y_1, Y_2, Y_3) \in (Z_2^{32})^4$. Załóżmy, że klucze poszczególnych rund zostały wygenerowane z klucza 128-bitowego i są definiowane jak następuje: $rk_i \in Z_2^{32}, i = 0, 1, \ldots, 31$. Szczegóły generowania kluczy rund opisane są w sekcji 3.4.

Szyfrowanie przebiega w następujący sposób:

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i) = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i), i = 0, 1, \ldots, 31$$

$$(Y_0, Y_1, Y_2, Y_3) = R(X_{32}, X_{33}, X_{34}, X_{35})$$

Deszyfrowanie odbywa się za pomocą tego samego algorytmu, jedyną różnicą jest odwrotna kolejność kluczy rund. Klucze rund szyfrowania $(rk_0, rk_1, \ldots, rk_{31})$, klucze rund deszyfrowania $(rk_{31}, rk_{30}, \ldots, rk_0)$



Rys. 2. Schemat algorytmu szyfrowania SM4

Bloki F i K opisane są kolejno na Rysunku 1 i Rysunku 3.

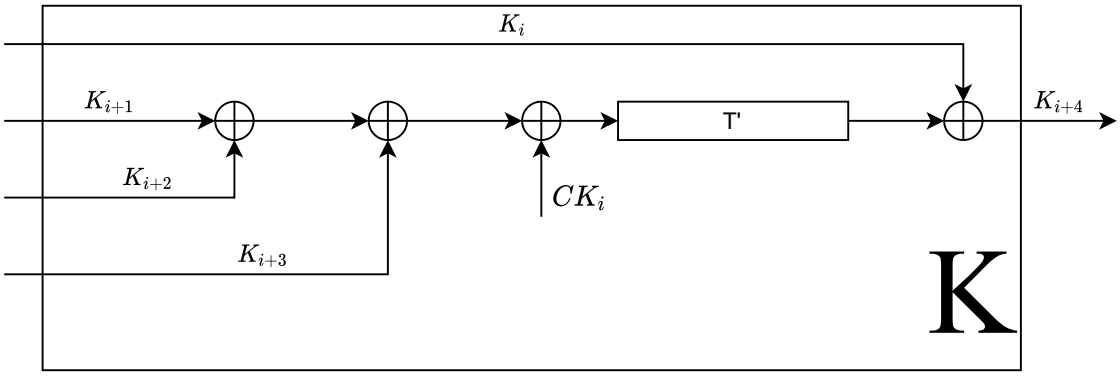
3.4. Generowanie kluczy rund

Klucze rund generowane są z klucza 128-bitowego. Niech $MK = (MK_0, MK_1, MK_2, MK_3), MK_i \in Z_2^{32}, i = 0, 1, 2, 3$ będzie głównym kluczem 128-bitowym. Wówczas klucze rund $rk_i \in Z_2^{32}, i = 0, 1, \ldots, 31$ otrzymywane są w następujący sposób.

$$(K_0, K_1, K_2, K_3) = (MK_0 \oplus FK_0, MK_1 \oplus FK_1, MK_2 \oplus FK_2, MK_3 \oplus FK_3)$$

Wówczas dla $i = 0, 1, \ldots, 31$:

$$rk_i = K_{i+1} = K_i \oplus T'(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i)$$



Rys. 3. Blok K

Mieszane podstawienie T'

Podstawienie T' jest defniowane tak samo jak podstawienie T z tą różnicą, że zamianst funkcji L używana jest następująca funkcja L':

$$L'(B) = B \oplus (B <<< 13) \oplus (B <<< 23)$$

Parametr FK

Przytaczane parametry FK zdefiniowane są w notacji heksadecymalnej:

$$FK_0 = (a3b1bac6), FK_1 = (56aa3350), FK_2 = (677d9197), FK_3 = (b27022dc)$$

Parametr CK

Niech $ck_{i,j}$ będzie j-tym bajtem $CK_i = (ck_{i,0}, ck_{i,1}, ck_{i,2}, ck_{i,3}) \in (Z_2^8)^4$, gdzie $ck_{i,j} = (4i + j) \times 7 \pmod{256}$. Z tego wynika, że parametry CK są stałymi i nie ma potrzeby każdorazowego obliczania ich.

4. Analiza bezpieczeństwa

Od czasu pierwszej publikacji, SM4 został poddany wielu kryptoanalizom wykonanym przez międzynarodowych badaczy. Obecnie, nie są jednak znane żadne praktyczne ataki na pełny szyfr SM4. Jedyne pojawiające się obawy związane z kanałami bocznymi [**1**], gdy algorytm używany jest w implementacji sprzętowej.

SM4 został przeanalizowany pod względem następujących typów ataków:

— liniowe - kryptoanaliza liniowa jest jedną z najważniejszych technik analizy kryptograficznej z kluczem symetrycznym. Kryptoanaliza liniowa skupia się na liniowości przybliżenia między tekstem jawnym, tekstem zaszyfrowanym i kluczem. Jeśli szyfr zachowuje się inaczej niż losowa permutacja można zbudować wyróżnik lub nawet atak odzyskiwania klucza poprzez dodanie kilku rund. Podklucze dołączonych rund są odgadywane, a szyfrogramy są odszyfrowywane i/lub jawne teksty są szyfrowane przy użyciu tych podkluczy do obliczenia stanu pośredniego na końcach wyróżnika.

— różnicowe - kryptoanaliza różnicowa, jest jednym z najpotężniejszych ataków na wybrany tekst jawny (lub wybrany tekst szyfrujący) w kryptografii symetryczno-kluczowej (tzn. w szyfrach blokowych, szyfrach strumieniowych, funkcjach haszujących i algorytmach MAC). Po wprowadzeniu tego ataku, został on skutecznie zastosowany do wielu znanych szyfrów, a także zaproponowano różne warianty tego ataku (atak na obciętą różnicę, atak na kwadrat, atak na różniczkę, atak na niemożliwą różnicę, atak na bumerang).

— liniowe wielowymiarowe - Biryukov et al. [**2**] zaproponował podejście, które może wykorzystywać wiele aproksymacji liniowych z różnymi bitami klucza. Jednakże, metody te zakładają, że aproksymacje liniowe są statystycznie niezależne.

— niemożliwe różnicowe - w porównaniu z normalnymi atakami różnicowymi, niemożliwy atak różnicowy jest bardziej efektywny w przypadku niektórych szyfrów blokowych, takich jak np. AES. Zastosowanie w SMS4 jest również kolejnym skutecznym sposobem atakowania tuż po atakach różnicowych.

— algebraiczne - Kryptoanaliza algebraiczna polega na znalezieniu i rozwiązaniu układu równań wielomianowych wielowartościowych w skończonym polu.
— liniowe o zerowej korelacji,
— integralne,
— macierzowe.

Porównanie najsilniejszych ataków na algorytm SM4 znajduje się w tabeli 2. Obecnie nie są powszechnie znane żadne skuteczne ataki powyżej 24 rundy.

Tabela 2. Najsilniejsze ataki na SM4

Metoda	Rundy	Złożoność czasu	złożoność danych	złożoność pamięci
$Linear$	24	$2^{122.6}$	$2^{122.6}$	2^{85}
$Multi-dimensional\ Linear$	23	$2^{122.7}$	$2^{122.6}$	$2^{120.6}$
$Differential$	23	$2^{126.7}$	2^{117}	$2^{120.7}$
$Matrix$	18	$2^{110.77}$	2^{127}	2^{130}
$Impossible\ Differential$	17	2^{132}	2^{117}	–
$Zero-correlation\ Linear$	14	$2^{120.7}$	$2^{123.5}$	2^{73}
$Integral$	14	$2^{96.5}$	2^{32}	–
$Bruteforce$	11	2^{128}	2^{128}	–

Na podstawie danych zamieszczonych w tabeli możemy stwierdzić, że atak brutalny nie jest skutecznym sposobem ataku w przypadku algorytmu SM4. Umożliwia on dotarcie tylko do 11 rundy przy bardzo wysokich wartościach złożoności czasowej i danych.

Kwestie bezpieczeństwa są w Chinach niezwykle istotne. Produkty i usługi wykorzystujące kryptografię są regulowane przez SCA [**3**] - muszą one być wyraźnie zatwierdzone lub certyfikowane przez SCA zanim zostaną dopuszczone do sprzedaży lub użytkowania w Chinach. Algorytm SM4 jest uważany tam za alternatywę dla AES-128.

4.1. Atak liniowy [6]

Aby przeprowadzić atak liniowy (atak na odzyskiwanie klucza) na 24-rundowe SM4 dodajemy dwie rundy odpowiednio na dole i na górze 20-rundowej aproksymacji liniowej w tabeli 3. Technika sum częściowych [**5**] jest używana w procedurach częściowego szyfrowania i deszyfrowania. Cały proces przedstawiony jest na rysunku 4. Zgodnie z aproksymacją liniową oznaczamy, że $\Gamma_2^0 = \Gamma_2^3 = \Gamma_{22}^0 = \Gamma_{22}^3 = 0x8808A288, \Gamma_2^1 = 0x8808C228, \Gamma_2^2 = \Gamma_{22}^1 = 0x88080828, \Gamma_{22}^2 = 0x88086828, \Lambda_1 = \Lambda_{22} = 0x00008200, \Lambda_{1,2} = \Lambda_{22,2} = 0x82$.
Z liniowego przybliżenia wynika, że

$$\Gamma_2^0 \cdot X_2^0 \oplus \Gamma_2^1 \cdot X_2^1 \oplus \Gamma_2^2 \cdot X_2^2 \oplus \Gamma_2^3 \cdot X_2^3 \oplus \Gamma_{22}^0 \cdot X_{22}^0 \oplus \Gamma_{22}^1 \cdot X_{22}^1 \oplus \Gamma_{22}^2 \cdot X_{22}^2 \oplus \Gamma_{22}^3 \cdot X_{22}^3 = K$$

Rozważmy częściowe szyfrowanie i deszyfrowanie, wtedy lewą stronę powyższego równania można zapisać w następujący sposób:

$$\Gamma_2^0 \cdot P_2 \oplus \Gamma_2^1 \cdot P_3 \oplus \Gamma_2^2 \cdot (P_0 \oplus XX_0) \oplus \Gamma_2^0 \cdot (P_1 \oplus XX_1) \oplus \Gamma_2^0 \cdot (C_2 \oplus XX_{22}) \oplus \Gamma_2^2 \cdot (C_3 \oplus XX_{23}) \oplus \Gamma_{22}^2 \cdot C_0 \oplus \Gamma_2^0 \cdot C_1 =$$

$$\Gamma_2^0 \cdot P_2 \oplus \Gamma_2^1 \cdot P_3 \oplus \Gamma_2^2 \cdot P_0 \oplus \Gamma_2^0 \cdot P_1 \oplus \Gamma_2^0 \cdot C_2 \oplus \Gamma_2^2 \cdot C_3 \oplus \Gamma_{22}^2 \cdot C_0 \oplus \Gamma_2^0 \cdot C_1 \oplus \Gamma_2^2 \cdot XX_0 \oplus \Gamma_2^0 \cdot XX_1 \oplus \Gamma_2^0 \cdot XX_{22} \oplus \Gamma_2^2 \cdot XX_{23},$$

gdzie XX_r jest stanem po transformacji T, a XS_r jest stanem po warstwie S w r-tej rundzie.

$$\Theta = \Gamma_2^0 \cdot P_2 \oplus \Gamma_2^1 \cdot P_3 \oplus \Gamma_2^2 \cdot P_0 \oplus \Gamma_2^0 \cdot P_1 \oplus \Gamma_2^0 \cdot C_2 \oplus \Gamma_2^2 \cdot C_3 \oplus \Gamma_{22}^2 \cdot C_0 \oplus \Gamma_2^0 \cdot C_1.$$

Atak odzyskiwania klucza przebiega w następujący sposób:

1. Zbierz N par tekstu jawnego i odpowiadającego mu szyfrogramu.

2. Inicjalizuj 2^{82} liczników $V_0[0]$, ..., $V_0[2^{81}-1]$ do zera.

3. Dla każdej pary tekstu jawnego i odpowiadającego mu szyfrogramu oblicz:

$$w = \Theta||(P_1 \oplus P_2 \oplus P_3)||(C_3 \oplus C_0 \oplus C_1)||C_0 \oplus C_1 \oplus C_2||P_1 \oplus P_2 \oplus P_3.$$

Następnie zwiększ licznik $V_0[w]$ o jeden.

4. Odgadnij 32-bitowe k_{23}.Zaalokuj 2^{49} liczników $V_1[0]$, ..., $V_1[2^{49}-1]$ na zero.

5. Dla każdego $0 \leqslant w \leqslant 2^{81}-1$ oblicz $\Theta \Leftarrow \Theta \oplus \Gamma_2^2 \cdot T(C_0 \oplus C_1 \oplus C_2 \oplus k_{23}), XX_{23}$ oraz $x = \Theta||(P_0 \oplus P_2 \oplus P_3)||(C_3 \oplus C_0 \oplus C_1) \oplus XX_{23}||P_1 \oplus P_2 \oplus P_3.V_1[x]+ = V_0[w]$.

6. Odgadnij 8-bitowe k_{22}. Zainicjuj 2^{41} liczników $V_2[0]$, ..., $V_2[2^{41}-1]$ na zero.

7. Dla każdego $0 \leqslant x \leqslant 2^{49}-1$ oblicz $\Theta \Leftarrow \Theta \oplus \Lambda_{1,2} \cdot S((C_3 \oplus C_0 \oplus C_1) \oplus XX_{23} \oplus k_{22})$ oraz $y = \Theta||(P_0 \oplus P_2 \oplus P_3)||P_1 \oplus P_2 \oplus P_3).V_2[y]+ = V_1[x]$.

8. Odgadnij 32-bitowe k_0. Zainicjuj 2^9 liczników $V_3[0]$, ..., $V_3[2^9-1]$ na zero.

9. Dla każdego $0 \leqslant y \leqslant 2^{41}-1$ oblicz $\Theta \Leftarrow \Theta \oplus \Gamma_2^2 \cdot T(P_1 \oplus P_2 \oplus P_3 \oplus k_0), XX_0]$ oraz $z = \Theta||(P_0 \oplus P_2 \oplus P_3) \oplus XX_0.V_3[z]+ = V_2[y]$.

10. Odgadnij 8-bitowe k_1. Zainicjuj 2^{80} liczników $V_1[0]$, ..., $V_1[2^{80}-1]$ na zero.

11. Dla każdego $0 \leqslant z \leqslant 2^9-1$ oblicz $\Theta \Leftarrow \Theta \oplus \Lambda_{1,2} \cdot S((P_1 \oplus P_2 \oplus P_3) \oplus XX_0 \oplus k_1)$. Jeśli $\Theta = 0$ zwiększ $V_{key}[k_0||k_1||k_{22}||k_{23}]$ o $V_3[z]$, w przeciwnym przypadku zmniejsz go o $V_3[z]$.

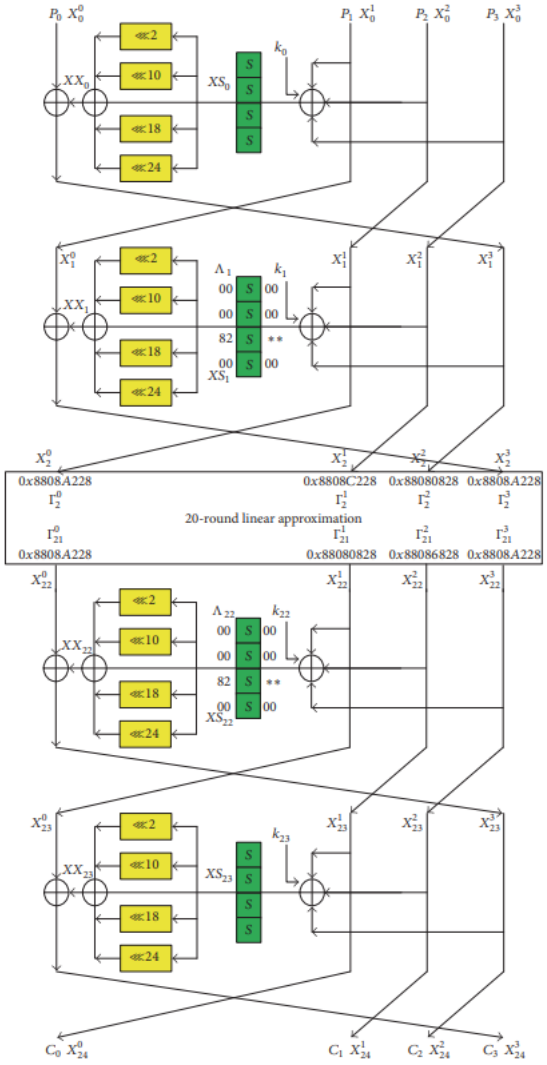
12. Ustaw przewagę α na 47, co oznacza, że zachowane są 2^{33} największe wartości bezwzględne w V_{key}. Dla każdej zachowanej wartości podklucza, zgadujemy pozostałe 88 bitów z $k_1[0-15, 24-31]||k_2||k_3$ (klucz główny można uzyskać z harmonogramu kluczy) i przetestować klucz przez ścieżkę szyfrowania.

Tabela 3. Aproksymacja liniowa dla 20-rundowego SM4

Rundy	i	Γ_i^0	Λ_i	Γ_i	tendencyjność	Γ_i^1	Γ_i^2	Γ_i^3
1	0	8808A228	00008200	00006000	2^{-4}	8808C228	88080828	8808A228
2	1	8808A228	00008200	0000CA00	2^{-4}	88086828	8808C228	8808A228
3	2	8808A228	00008200	0000AA00	2^{-4}	88080828	88086828	8808A228
4	3	8808A228	00008200	00006000	2^{-4}	8808C228	88080828	8808A228
5	4	8808A228	00008200	0000CA00	2^{-4}	88086828	8808C228	8808A228
6	5	8808A228	00008200	0000AA00	2^{-4}	88080828	88086828	8808A228
7	6	8808A228	00008200	00006000	2^{-4}	8808C228	88080828	8808A228
8	7	8808A228	00008200	0000CA00	2^{-4}	88086828	8808C228	8808A228
9	8	8808A228	00008200	0000AA00	2^{-4}	88080828	88086828	8808A228
10	9	8808A228	00008200	00006000	2^{-4}	8808C228	88080828	8808A228
11	10	8808A228	00008200	0000CA00	2^{-4}	88086828	8808C228	8808A228
12	11	8808A228	00008200	0000AA00	2^{-4}	88080828	88086828	8808A228
13	12	8808A228	00008200	00006000	2^{-4}	8808C228	88080828	8808A228
14	13	8808A228	00008200	0000CA00	2^{-4}	88086828	8808C228	8808A228
15	14	8808A228	00008200	0000AA00	2^{-4}	88080828	88086828	8808A228
16	15	8808A228	00008200	00006000	2^{-4}	8808C228	88080828	8808A228
17	16	8808A228	00008200	0000CA00	2^{-4}	88086828	8808C228	8808A228
18	17	8808A228	00008200	0000AA00	2^{-4}	88080828	88086828	8808A228
19	18	8808A228	00008200	00006000	2^{-4}	8808C228	88080828	8808A228
20	19	8808A228	00008200	0000CA00	2^{-4}	88086828	8808C228	8808A228
21	20	8808A228	00008200	*	*	88080828	88086828	8808A228

Złożoność czasowa kroku (3) wynosi około $2^{126.6}$ operacji, co jest równoważne $2^{126.6}/24 = 2^{122.0}$ 24-rundom szyfrowania. Kroki (5) i (9) wymagają 2^{113} jednokrotnych deszyfracji lub szyfrowania. Kroki (7) i (11) wymagają

2^{89} jednokrotnych odszyfrowań lub zaszyfrowań. Złożoność kroku (12) wynosi 2^{121} 24-rund szyfrowań. Zatem całkowita złożoność czasowa wynosi około $2^{126.6}$ szyfrowań. Złożoność pamięciowa kroku (2) wynosi około $2^{81} \times 8 = 2^{84}$ bajtów, a licznik $Vkey$ wymaga $2^{80} \times 16 = 2^{84}$ bajtów, więc całkowita złożoność pamięciowa wynosi około $2^{84} + 2^{84} = 2^{85}$ bajtów.
Przy złożoności danych $2^{126.6}$, prawdopodobieństwo sukcesu tego ataku wynosi 76.1%.



Rys. 4. Atak odzyskiwania klucza na 24-rundowy SM4

5. Analiza efektywności

Efektywność algorytmu jest jedną z jego podstawowych cech. Możemy wyróżnić złożoność czasową i pamięciową algorytmu i na tej podstawie wybrać, który algorytm jest dla nas bardziej korzystny ze względu na czas, albo na złożoność pamięciową.

5.1. Złożoność czasowa

Złożoność czasowa – to ilość czasu potrzebnego do wykonania zadania, wyrażona jako funkcja ilości danych, aby ją określić należy rozważyć złożoność poszczególnych operacji oraz ilość ich powtórzeń. Wynik powinien być sprawdzony dla każdej możliwej kombinacji wejściowej. Funkcja złożoności czasowej: t: 0, 1* → N nazywamy

złożonością czasową algorytmu A, jeżeli dla każdego x na wejściu algorytm zatrzymuje się po x krokach dokładnie w t(x). Jeśli funkcja otrzymuje skończoną liczbę danych wejściowych, to jej złożoność czasowa wynosi O(1), zależy więc od rozmiaru danych.

Złożoność czasowa funkcji SM4:
— funkcja rundy F - funkcja otrzymuje pięć 32-bitowych wektorów i zwraca jeden 32 bitowy wektor, podczas tej funkcji wykonywane są 4 operacje XOR dwóch 32 bitowych wektorów oraz mieszane podstawienie T.
— mieszanie podstawienie T - jest to odwracalna funkcja podstawienia, składa się z nieliniowego podstawienia oraz liniowego L.
— nieliniowe podstawienie - funkcja otrzymuje na wejście 4 bajty i przy pomocy S box podstawia dane, jest to więc funkcja jednokierunkowa.
— liniowe podstawienie L - funkcja przyjmująca 32-bitowy wektor i zwraca jeden 32 bitowy wektor, wykonuje 4 operacje XOR oraz 4 razy rotacja w lewo o 2, 10, 18 i 24 bity

Złożoność czasowa wszystkich tych funkcji wynosi O(1), wynika więc z tego, że algorytm SM4 posiada liniową złożoność czasową równą O(n).

5.2. Złożoność pamięciowa

Złożoność pamięciowa jest miarą ilości pamięci wykorzystanej podczas wykonywania zadania obliczeniowego.

Złożoność pamięciowa funkcji:
— S-Box - mapuje 32-bitowe wejście na 32-bitowe wyjście i używa skończonej liczby parametrów pamięci tymczasowej do wykonania swojego zadania, wówczas jego złożoność pamięciowa wynosi O(1).
— liniowe podstawienie L - funkcja przyjmująca 32-bitowy wektor i zwraca jeden 32 bitowy wektor, wykonuje 4 operacje XOR oraz 4 razy rotacja w lewo o 2, 10, 18 i 24 bity również jego złożoność pamięciowa wynosi O(1).
— nieliniowe podstawienie - funkcja otrzymuje na wejście 4 bajty i przy pomocy S box podstawia dane, jest to więc funkcja jednokierunkowa, którego również złożoność pamięciowa wynosi O(1).
— mieszanie podstawienie T - składa się z nieliniowego podstawienia oraz liniowego L, których złożoność pamięciowa wynosi O(1), więc złożoność podstawienie T również wynosi O(1).
— funkcja rundy F - ze względu na to, że wszystkie operacje wykonywane w tej funkcji mają złożoność pamięciową O(1), to cała funkcja posiada również taką złożoność.

Po obliczeniu złożoności pamięciowej poszczególnych funkcji możemy stwierdzić, że złożoność pamięciowa całego algorytmu SM4 wynosi O(1).

Naszym planem analizy efektywności jest porównanie złożoności poszczególnych operacji z innymi algorytmem blokowym, ze względu na popularność wybraliśmy algorytm AES. Skupimy się na porównaniu ilości niezbędnych operacji to zakodowania takiego samego ciągu znaków tj XOR, czy przesunięcia. Dzięki dostępności gotowych implementacji algorytmu AES, będziemy mogli również po zaimplementowaniu porównać czasy potrzebne dla szyfrowania tekstu dla algorytmu SM4 oraz AES.

6. Implementacja

Do implementacja naszego algorytmu zdecydowaliśmy się wykorzystać język programowania Python.

6.1. Wykorzystywane biblioteki

Wykorzystano następujące biblioteki języka Python:

— termcolor - do pokazywania logów w konsoli w kolorze zielonym - kiedy test przebiegł pomyślnie oraz w kolorze czerwonym - kiedy pojawił się błąd
— tqdm - wyświetla progres wykonywania zadania w naszych przypadku, na jakim poziomie jest obecnie wykonywany test
— Crypto.Cipher - tej biblioteki użyliśmy do szyfrowania szyfrem AES

6.2. Uruchomienie symulacji

Do uruchomienia programu realizującego zadane symulacje wymagany jest **Python 3** oraz zainstalowanie powyższych zależności. Zostały one zawarte w pliku *requirements.txt.* Należy je zainstalować za pomocą polecenia: ***pip install -r requirements.txt*** Aby uruchomić symulację należy wewnątrz folderu wywołać w konsoli polecenie ***python tests.py.***

6.3. Implementacja

Implementacja naszego algorytmu składa się z 3 plików:

— SM4.py
— tests.py
— bytesHelpers.py
W pliku bytesHelpers umieszczone zostały funkcje pomocnicze tj. long_to_bytes, bytes_to_long, xor oraz leftShift.

Plik SM4.py opisuję główną klasę algorytmu, składa się z następujących elementów, które realizują schemat algorytmu przedstawiony w punkcie 3:

— **konstruktora** - tworzy obiekt klasy SM4, składający się z klucza (*key*), oraz zmiennej *roundedKeys*, która przechowuje 32 klucze rund oraz zmiennej *recentOutputs*, która przetrzymuje wartość wyjściową z danej rundy i przekazuje ją na wejście kolejnej rundy

— **stałych**:
  ◇ *sboxTable* - skrzynka podstawieniowa
  ◇ *FK* - parametr FK
  ◇ *CK* - parametr CK

— **funkcji**:
  ◇ *getRoundKeys* - pobiera wektor kluczy
  ◇ *getRecentOutputs* - pobiera wartość *recentOutputs*
  ◇ *encrypt* - funkcja szyfrująca, wywołuje funkcje *encryptDecrypt* z przekazywanym tekstem do zaszyfrowania
  ◇ *decrypt* - funkcja deszyfrująca, wywołuje funkcje *encryptDecrypt* z przekazywanym tekstem do odszyfrowania oraz drugim parametrem *isDecryption* ustawionym na *true*

- ◇ *encryptDecrypt* - w zależność od typu operacji: szyfrowanie lub deszyfrowanie ustawia odpowiednią kolejność kluczy, dzieli tekst na 4 słowa i wywołuje 32 razy funkcję rundy*roundFunction*
- ◇ *roundFunction* - wykonuje operacje XOR z wyjściem funkcji *mixerSubstitution*
- ◇ *mixerSubstitution* - wywołuje liniowanie przekształcenie (*linearSubstitution*) podając jako parametr wyjście funkcji nieliniowego przekształcenia (*nonlinearSubstitution*)
- ◇ *nonlinearSubstitution* - generuj wektor z 4 słów, które zostały stworzone przy pomocy skrzynki podstawieniowej
- ◇ *linearSubstitution* - wykonuje serie operacji XOR z wektorami po rotacji w lewo, odpowiednio o 2, 10, 18 i 24 bity.
- ◇ *keyExpansion* - generuje 32 klucze dla każdej rundy poprzez XORowanie poszczególnych słów klucza z odpowiednimi elementami *parametru FK*, następnie w pętli dodaje je do listy jednocześnie XORując wcześniejszy klucz z wyjściem funkcji *mixerSubstitution2*
- ◇ *mixerSubstitution2* - wywołuje liniowanie przekształcenie (*linearSubstitution2*) podając jako parametr wyjście funkcji nieliniowego przekształcenia (*nonlinearSubstitution*)
- ◇ *linearSubstitution2* - XORuje wektor z wektorami po rotacji w lewo, odpowiednio o 13 i 23 bity.
- ◇ *sbox* - zwraca podstawiony wektory przy pomocy skrzynki podstawieniowej - *sboxTable*

Wszelkie powyższe operacje są dokładną implementacją operacji występujących w algorytmie SM4.

6.4. Testy

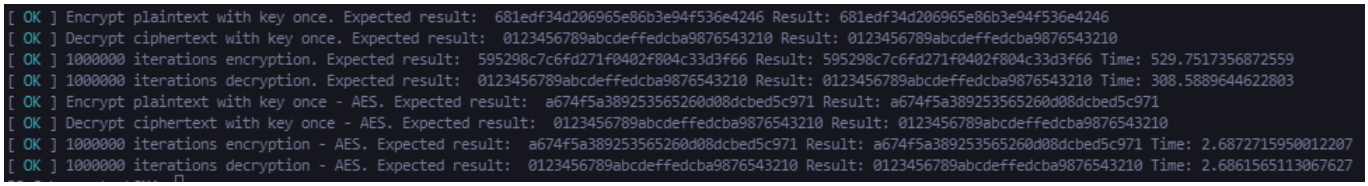
Testy zrealizowany zostały przy pomocy pliku *tests.py*. Przykłady scenariuszy testowy:

1. Sprawdzenie, czy klucze wygenerowały się poprawnie i są takie jak oczekiwaliśmy.
2. Zaszyfrowanie jednego słowa i porównanie, czy wyniki jest zgodny z oczekiwanym.
3. Odszyfrowanie jednego słowa i porównanie, czy wyniki jest zgodny z oczekiwanym.
4. Testy wykonane w pętli dla 1000000 iteracji. Po każdej iteracji sprawdzenie, czy tekst został zaszyfrowany poprawnie.
5. Testy wykonane w pętli dla 1000000 iteracji. Po każdej iteracji sprawdzenie, czy tekst został zdeszyfrowany poprawnie.

Dodatkowo, aby przeprowadzić porównanie wydajnościowe z algorytmem AES, przeprowadziliśmy dodatkowe testy.

1. Zaszyfrowanie jednego słowa i porównanie, czy wyniki jest zgodny z oczekiwanym dla algorytmu AES.
2. Odszyfrowanie jednego słowa i porównanie, czy wyniki jest zgodny z oczekiwanym dla algorytmu AES.
3. Testy wykonane w pętli dla 1000000 iteracji. Po każdej iteracji sprawdzenie, czy tekst został zaszyfrowany poprawnie dla algorytmu AES.
4. Testy wykonane w pętli dla 1000000 iteracji. Po każdej iteracji sprawdzenie, czy tekst został zdeszyfrowany poprawnie dla algorytmu AES.

Wszystkie testy przebiegły pomyślnie, co potwierdza poniższy zrzut ekranu.



Rys. 5. Testy algorytmu szyfrowania SM4

6.5. Porównanie wydajności

Planem analizy wydajności, było przeprowadzanie takie samej liczby testów, z tym samym słowem i kluczem, dla algorytmu SM4 oraz AES, ponieważ oba te algorytmy to szyfry blokowe. Aby to zrobić zmierzyliśmy czas wykonywania wszystkich iteracji dla każdego algorytmu i wyliczyliśmy średnia przepływność tych algorytmów. Otrzymane wyniki zaprezentowaliśmy w tabeli 4.

Tabela 4. Porównanie przepływności

Metoda	Algorytm SM4 [MB/s]	Algorytm AES [MB/s]
$Szyfrowanie$	0,03	5,68
$Deszyfrowanie$	0,05	5,68

Algorytm AES jest zdecydowanie szybszym algorytmem niż algorytm SM4, ponad 100-krotnie. Fakt, że algorytm SM4 nie został jeszcze złamany, świadczy o jego atrakcyjności ze względu na poziom bezpieczeństwa, jednak pod względem wydajności, w przypadku naszej implementacji, zdecydowanie lepiej prezentuje się algorytm AES. Tak duża różnica w wydajności może być spowodowana implementacją SM4 w języku Python, który jest językiem interpretowanym. Z dużym prawdopodobieństwem lepsze rezultaty uzyskałaby implementacja w języku kompilowanym takim jak np. C.

Bibliografia

[1] Lei, Q., Wu, L., Zhang, S., Zhang, X., Li, X., Pan, L. and Z. Dong: *Software Hardware Co-design for Side-Channel Analysis Platform on Security Chips*, December 2015

[2] Biryukov, A., De Canniere, C., Quisquater, M.: *On Multiple Linear Approximations*, 2004

[3] State Cryptography Administration of China: *State Cryptography Administration of Chinas*, December 2017

[4] Translated and typeset by Whitfield Diffie of Sun Microsystems and George Ledin of Sonoma State University: *SMS4 Encryption Algorithm for Wireless Networks*, 15 May 2008

[5] N. Ferguson, J. Kelsey, S. Lucks et al., Fast Software Encryption, vol. 1978 of Lecture Notes in Computer Science, pp. 213–230, Springer: *Improved Cryptanalysis of Rijndael*, 2000

[6] Yu Liu, Huicong Liang, Wei Wang, Meiqin Wang: *New Linear Cryptanalysis of Chinese Commercial Block Cipher Standard SM4*, 6 September 2017