Funkcja jednokierunkowa VMPC i system uwierzytelnionego szyfrowania VMPC-Tail-MAC

Bartosz Żółtak

http://www.vmpcfunction.com
bzoltak@vmpcfunction.com

Przedmiotem pracy są opracowane przez autora funkcja jednokierunkowa oraz szyfr strumieniowy VMPC, zaprezentowane na międzynarodowej konferencji Fast Software Encryption 2004, Delhi, Indie, 5-7 lutego 2004, oraz schemat uwierzytelnionego szyfrowania oparty na szyfrze VMPC i koncepcji Tail-MAC obliczania kodów uwierzytelniających wiadomość.

Funkcja jednokierunkowa VMPC charakteryzuje się niezwykłą prostotą – obliczenie jej wartości wymaga trzech podstawowych operacji na permutacjach na bajt, które ponadto mogą być zaimplementowane trzema podstawowymi instrukcjami „MOV" procesora. Jednocześnie najszybsza znana autorowi metoda odwracania funkcji wymaga średnio 2^{260} operacji.

Szyfr strumieniowy VMPC został zaprojektowany jako praktyczne zastosowanie funkcji VMPC w kryptografii. Podstawowymi założeniami projektowymi było uzyskanie algorytmu bezpiecznego o jak najprostszej konstrukcji, o wysokiej wydajności w implementacji programowej oraz wolnego od szeregu wad, jakie są charakterystyczne dla popularnego szyfru RC4.

Koncepcja Tail-MAC jest propozycją bardzo prostego i wydajnego algorytmu obliczania kodów uwierzytelniających wiadomość (MAC) dla szyfrów strumieniowych. Jej podstawowym założeniem konstrukcyjnym jest wykorzystanie dla celów obliczenia MACa części operacji, wykonanych już przez algorytm szyfrowania, co pozwala na osiągnięcie dużej wydajności dzięki minimalizowaniu dodatkowych – poza procesem szyfrowania – operacji wykonywanych przez kryptosystem.

1. Wstęp

Funkcja VMPC (Variably Modified Permutation Composition – zmiennie modyfikowane złożenie permutacji) jest kombinacją elementarnych operacji na permutacjach oraz liczbach całkowitych. Podczas gdy wyliczenie wartości najprostszego wariantu funkcji (VMPC 1. stopnia) wymaga jedynie trzech operacji „MOV" procesora 80x86 (które na procesorze 80486 i Pentium zajmują jeden cykl zegarowy) na bajt, odwrócenie funkcji, gdy zostanie zastosowana na permutacjach 256-elementowych (dogodny rozmiar w zastosowaniach kryptograficznych), wymaga średnio 2^{260} operacji. Co więcej, odporność funkcji VMPC na odwrócenie jest w bardzo prosty sposób skalowalna, np. funkcja VMPC 2. stopnia, która jest możliwa do zaimplementowania w czterech instrukcjach „MOV", wymaga średniego nakładu 2^{420} operacji do jej odwrócenia.

Prostota funkcji rodzić może pytanie – czy nie jest możliwe udowodnienie, że funkcji tej w istocie nie da się odwrócić szybciej niż średnio w X operacji, gdzie X byłoby liczbą obliczalnie nieosiągalną. Waga takiego dowodu byłaby ogromna, zwłaszcza dla teorii złożoności, gdyż pozwoliłby on rozstrzygnąć problem P ?= NP, stwierdzając, że P ≠ NP, co wynika bezpośrednio z istnienia funkcji (praktycznie) jednokierunkowej. Jest to aktualnie otwarty problem i możliwy przedmiot przyszłych badań.

Własności funkcji VMPC – bardzo niski nakład, jakim można uzyskać praktyczną jednokierunkowość – sprawia, że funkcja ta staje się interesującym kandydatem do zastosowań kryptograficznych. Propozycją autora zastosowania funkcji w kryptografii symetrycznej jest szyfr strumieniowy VMPC. Szyfr ten jest bardzo wydajny w implementacji programowej, charakteryzuje się dużą prostotą konstrukcji oraz eliminuje szereg wad statystycznych charakterystycznych dla szyfru RC4 oraz wad w algorytmie inicjowania klucza wewnętrznego RC4 (Key Scheduling Algorithm, KSA).

Ponadto zastosowana dla szyfru VMPC konstrukcja KSA pozwala na zbudowanie schematu szyfrowania dowolnej liczby wiadomości tym samym kluczem w sposób udowadnialnie bezpieczny po przyjęciu prawdziwości pewnych założeń – w sensie, że złamanie szyfru możliwe jest tylko metodą przeszukania wszystkich (średnio połowy) możliwych wartości klucza (tzw. metoda brute-force).

Potrzeba uwierzytelniania szyfrowanej wiadomości (mesage authentication), a więc kontroli, czy wiadomość po deszyfracji jest dokładnie tą samą wiadomością, która została zaszyfrowana, jest istotnym wymaganiem dla praktycznych systemów kryptograficznych. Opracowana przez autora koncepcja Tail-MAC jest ogólną propozycją prostego i wydajnego algorytmu obliczania kodów uwierzytelniających wiadomość (Message Authentication Code, MAC) dla szyfrów strumieniowych. Tail-MAC wykorzystuje część danych klucza wewnętrznego (internal state) algorytmu szyfrującego, co pozwala na minimalizowanie liczby operacji dodatkowych, poza samym procesem szyfrowania. W pracy niniejszej omówiony zostanie przykład zastosowania Tail-MACa z szyfrem strumieniowym VMPC wraz z oszacowaniem złożoności złamania MACa (sfałszowania wiadomości) z wykorzystaniem najefektywniejszego znanego autorowi ataku.

2. Definicja funkcji VMPC

Notacja:

n, P, Q : P i Q: n-elementowe permutacje; $A \rightarrow A$, gdzie $A = \{0,1,\ldots,n\text{-}1\}$

k : Stopień funkcji; $k < n$

$+$: dodawanie modulo n

Definicja:

Funkcją VMPC k-tego stopnia, określaną jako VMPC_k, jest takie przekształcenie P w Q, gdzie:

$$Q[x] = P[P_k[P_{k-1}[\ldots[P_1[P[x]]]\ldots]]],$$

$x \in \{0,1,\ldots,n\text{-}1\}$,

P_i jest n-elementową permutacją, taką że $P_i[x] = f_i(P[x])$, gdzie

f_i jest dowolną funkcją taką, że $P_i[x] \neq P[x] \neq P_j[x]$ dla $i \in \{1,2,...,k\}$, $j \in \{1,2,...,k\}$, $i \neq j$.

Dla prostoty przyszłych odwołań f_i jest przyjęta jako $f_i(x) = x + i$

<u>*Przykład:*</u> $Q = \text{VMPC}_1(P)$: $Q[x] = P[P[P[x]]+1]$

$Q = \text{VMPC}_2(P)$: $Q[x] = P[P[P[P[x]]+1]+2]$

3. Istota funkcji VMPC

Istotę funkcji VMPC można łatwo zobrazować na przykładzie funkcji 1. stopnia, gdzie $Q[x] = P[P[P[x]]+1]$. Operacja arytmetyczna (+1) wykonana na wartości permutacji P podczas obliczania jej potrójnego złożenia odgrywa tu kluczową rolę. W wyniku tej operacji struktura cykli permutacji P zostaje zakłócona, co w dalszych analizach okaże się mieć zasadnicze konsekwencje dla trudności odwrócenia funkcji, a więc znalezienia permutacji P na podstawie permutacji Q.

Zauważmy, że proste potrójne złożenie permutacji zachowuje strukturę cykli składanej permutacji. Potrójne złożenie powodować może co najwyżej skrócenie cykli, ale mając daną permutację Q, gdzie $Q[x]=P[P[P[x]]]$ (proste złożenie), możemy w bardzo prosty sposób odtworzyć postać P. Mając $Q[X]=Y$ możemy stwierdzić o permutacji P, że następujące elementy P należą do jednego cyklu permutacji P: $P[X]=a$, $P[a]=b$, $P[b]=Y$, lub, stosując notację cykli: (X, a, b, Y, ...), gdzie X i Y są znane. Zauważmy, że mając dane takie części cykli permutacji P, jak (X, a, b, Y, ...), dla wszystkich elementów Q (dla $X \in \{0,1,...,n\text{-}1\}$), możemy je w prosty sposób poskładać na siebie tak, aby znane wartości pokryły się z nieznanymi i odczytać postać całej permutacji P.

Operacja dodania liczby 1 do wartości permutacji P na drugim kroku jej składania w funkcji VMPC ($Q[x] = P[P[P[x]]+1]$) zakłóca tę regularność. Sprawia ona, że do wyliczenia jednego elementu permutacji Q użyte mogą być elementy permutacji P pochodzące z różnych cykli. Zauważmy, co możemy powiedzieć o permutacji P na podstawie danego elementu $Q[X]=Y$, stosując analogiczne rozumowanie, jak dla prostego złożenia permutacji: $P[X]=a$, $P[a]=b$, $P[b+1]=Y$. Stosując notację cykli jest to jednoznaczne z (X, a, b, ...) oraz (b+1, Y, ...). Takich fragmentów cykli, których regularność zakłócona jest operacją dodawania, nie możemy już na siebie w prosty sposób poskładać, odtwarzając pierwotną strukturę cykli permutacji P. Operacja dodawania wykonana na wartości danego elementu permutacji P miesza bowiem występujące w P cykle i odtworzenie ich oryginalnej struktury nie jest już możliwe w sposób analogiczny, jak dla prostego złożenia permutacji. Przybliżenie problemu odwracania funkcji VMPC zawarte jest w rozdziale 4.

4. Trudność odwrócenia funkcji VMPC

n-elementowa permutacja P ma być odtworzona z n-elementowej permutacji Q, gdzie $Q=\mathrm{VMPC}_k(P)$.

Z definicji każdy element Q jest wyliczony z wykorzystaniem k+2, zwykle różnych od siebie, elementów P. Dany element Q może być uzyskany z wielu możliwych konfiguracji elementów P (np. dla $Q=\mathrm{VMPC}_1(P)$: $Q[X]=Y$ może być uzyskany z $P[X]=a$, $P[a]=b$, $P[b+1]=Y$ dla dowolnej rozsądnej kombinacji wartości zmiennych a i b).

Wszystkie z możliwych konfiguracji są równie prawdopodobnie prawidłowe. Jeśli którakolwiek konfiguracja zostałaby wybrana, musi zostać zweryfikowana z wykorzystaniem wszystkich tych elementów permutacji Q, do których wyliczenia został wykorzystany którykolwiek element P z wybranej konfiguracji.

Każdy element P jest zwykle użyty do wyliczenia k+2 różnych elementów Q, więc zwykle (k+2) × (k+1) nowych elementów permutacji Q musi zostać odwróconych (wszystkie k+2 elementy P użyte do wyliczenia każdego z tych elementów Q muszą zostać ujawnione), aby zweryfikować k+2 elementów P zawartych w wybranej konfiguracji.

Ponieważ struktura cykli P jest zakłócona operacją dodawania, zwykle niemożliwe jest znalezienie dwóch różnych elementów Q, które mają k+1 wspólnych elementów P, przy użyciu których te elementy Q zostały wyliczone. Zamiast tego, zwykle taki element Q może zostać znaleziony ($Q[x]$), który ma tylko jeden wspólny element P z innym elementem Q. Innymi słowy trudno jest znaleźć takie elementy permutacji Q: $Q[x]$, $Q[y]$, $x \neq y$, do wyliczenia których wykorzystanych by zostało mniej niż ((k+2)×2–1) różnych elementów permutacji P.

To sprawia, że aby odwrócić jakikolwiek nowy element permutacji Q: $Q[x]$, k elementów P, użytych do wyliczenia $Q[x]$, musi zwykle zostać zgadniętych.

Jednakże z każdym zgadniętym elementem P pojawia się zwykle k+1 nowych elementów Q, do których wyliczenia dany element P został wykorzystany, i które muszą być odwrócone w celu zweryfikowania poprawności zgadniętego elementu P.

Algorytm wpada w pętlę, gdzie na każdym kroku zwykle *k* nowych elementów *P* musi zostać zgadniętych, aby umożliwić kontynuację weryfikacji elementów *P* zgadniętych wcześniej. W konsekwencji *k*+2 elementy *P* zgadnięte na początku procesu (wybrana konfiguracja) pośrednio zależą od wszystkich *n* elementów *Q*.

Opisana sytuacja ma miejsce zwykle. W sprzyjających okolicznościach oraz ze wzrostem liczby elementów *P* już ujawnionych, rośnie prawdopodobieństwo, że proces weryfikacji zostanie uproszczony poprzez wykorzystanie zbiegów okoliczności, gdzie np. jest możliwe znalezienie dwóch elementów *Q*, które mają więcej niż 1 wspólnych elementów *P* (np. dla Q=VMPC$_1$(P): Q[2]=3: P[2]=4, P[4]=8, P[9]=3 oraz Q[1]=8: P[1]=9, P[9]=3, P[4]=8) lub np. gdy ujawnione już elementy *P* pochodzące z różnych elementów *Q* mogą zostać dopasowane jako więcej niż 1 element *P* użyty do wyliczenie nowego, nieodwróconego jeszcze, elementu *Q*.

Opracowany algorytm odwracania funkcji VMPC został zoptymalizowany tak, aby wykorzystywać możliwe do zaistnienia zbiegi okoliczności. Średnia liczba elementów *P*, które muszą być zgadnięte dla funkcji VMPC 1. stopnia i n=256 została zredukowana (z około 128, co wynikałoby z ekstrapolacji sytuacji opisanej powyżej na ujawnienie wszystkich elementów *P*) do około 34. Dla VMPC$_2$ do około 57, dla VMPC$_3$ do około 77, a dla funkcji VMPC 4. stopnia – do około 92 zgadniętych elementów *P*.

Przeszukanie średnio połowy zgadywanych elementów *P* przez algorytm odwracania funkcji VMPC wymaga średnio około 2^{260} operacji dla funkcji VMPC 1. stopnia, około 2^{420} dla VMPC$_2$, około 2^{550} dla VMPC$_3$ oraz około 2^{660} operacji dla funkcji VMPC 4. stopnia.

Dokładny opis algorytmu odwracania funkcji VMPC znajduje się w [16] oraz w [19].

5. Implementacja funkcji VMPC w 3 instrukcjach procesora

Tabela 1 zawiera implementację assemblerową funkcji VMPC 1. stopnia, gdzie $Q[x]=P[P[P[x]]+1]$ dla 256-elementowych permutacji.

Założenia:

- *Pm* jest 257-elementową tablicą indeksowaną liczbami od 0 do 256. Permutacja *P* jest umieszczona w tablicy *Pm* w następujący sposób: *Pm* [0...255]=*P* oraz *Pm* [256]=*P*[0]
- 32-bitowy rejestr EAX określa, który element permutacji *Q* ma być wyliczony (AL zawsze oznacza 8 najmniej znaczących bitów EAX. W rozpatrywanej sytuacji zawsze zachodzi EAX =AL)

Instrukcja	Opis
`MOV AL, [Pm] + EAX`	Umieść (EAX=AL)-ty element *Pm* w AL
`MOV AL, [Pm] + EAX`	Umieść (EAX=AL)-ty element *Pm* w AL
`MOV AL, [Pm] + EAX + 1`	Umieść ((EAX=AL)+1)-ty element *Pm* w AL.

Tabela 1. Implementacja assemblerowa funkcji VMPC$_1$

6. Przykładowe wartości funkcji VMPC

Tabela 2 zawiera wartości funkcji VMPC 1.,2.,3. i 4. stopnia dla przykładowej 10-elementowej permutacji *P*

indeks	0	1	2	3	4	5	6	7	8	9
P	2	0	4	3	6	9	7	8	5	1
Q_1=VMPC$_1$(P)	9	3	8	6	5	4	1	7	2	0
Q_2=VMPC$_2$(P)	0	9	2	5	8	7	3	1	6	4
Q_3=VMPC$_3$(P)	3	4	9	5	0	2	7	6	1	8
Q_4=VMPC$_4$(P)	8	5	3	1	6	7	0	2	9	4

Tabela 2. Przykładowe wartości funkcji VMPC

7. Przykładowe złożoności odwrócenia funkcji VMPC

Zawarte w tabeli 3 średnie złożoności odwrócenia funkcji VMPC dla wybranych rozmiarów permutacji (n) oraz dla stopni funkcji $k \in \{1,2,3,4\}$ zostały obliczone jako symulowana średnia liczba kroków, jakie musi wykonać zarysowany w rozdziale 4 oraz opisany dokładnie w [16] i [19] algorytm odwracania funkcji. Liczby elementów P, jakie musza być zgadnięte, podano w tabeli 3 w nawiasach.

Q=VMPC$_1$(P) : $Q[x] = P[P[P[x]]+1]$
Q=VMPC$_2$(P) : $Q[x] = P[P[P[P[x]]+1]+2]$
Q=VMPC$_3$(P) : $Q[x] = P[P[P[P[P[x]]+1]+2]+3]$
Q=VMPC$_4$(P) : $Q[x] = P[P[P[P[P[P[x]]+1]+2]+3]+4]$

Funkcja / n	**VMPC$_1$**	**VMPC$_2$**	**VMPC$_3$**	**VMPC$_4$**
6	$\mathbf{2^{4,1}}$ (2,3)	$\mathbf{2^{5,5}}$ (3,1)	$\mathbf{2^{6,1}}$ (3,3)	$\mathbf{2^{6,9}}$ (3,8)
8	$\mathbf{2^{5,5}}$ (2,7)	$\mathbf{2^{7,5}}$ (3,4)	$\mathbf{2^{8,8}}$ (4,0)	$\mathbf{2^{9,8}}$ (4,4)
10	$\mathbf{2^{7,1}}$ (3,0)	$\mathbf{2^{9,7}}$ (4,0)	$\mathbf{2^{11,5}}$ (4,7)	$\mathbf{2^{13,0}}$ (5,2)
16	$\mathbf{2^{11,5}}$ (3,8)	$\mathbf{2^{16,6}}$ (5,4)	$\mathbf{2^{20,4}}$ (6,6)	$\mathbf{2^{23,3}}$ (7,5)
32	$\mathbf{2^{24}}$ (6,0)	$\mathbf{2^{37}}$ (9,1)	$\mathbf{2^{47}}$ (11,5)	$\mathbf{2^{54}}$ (13,4)
64	$\mathbf{2^{53}}$ (10,2)	$\mathbf{2^{84}}$ (16,2)	$\mathbf{2^{108}}$ (21,0)	$\mathbf{2^{127}}$ (24,9)
128	$\mathbf{2^{117}}$ (18,5)	$\mathbf{2^{190}}$ (30,0)	$\mathbf{2^{245}}$ (40,0)	$\mathbf{2^{292}}$ (47,0)
256	$\mathbf{2^{260}}$ (34,0)	$\mathbf{2^{420}}$ (57,0)	$\mathbf{2^{550}}$ (77,0)	$\mathbf{2^{660}}$ (92,0)

Tabela 3. Przykładowe złożoności odwrócenia funkcji VMPC

Przykład: Dla funkcji VMPC 1. stopnia zastosowanej na 256-elementowej permutacji średnio około 34 elementy P muszą zostać zgadnięte, a przeszukanie połowy stanów tych elementów P przez algorytm odwracania funkcji VMPC wymaga wykonania średnio około 2^{260} operacji.

8. Założenia dla szyfru strumieniowego VMPC

Szyfr strumieniowy VMPC jest propozycją zastosowania funkcji jednokierunkowej VMPC w kryptografii symetrycznej. Szyfr posiada zdefiniowaną procedurę inicjowania klucza wewnętrznego (internal state), tzw. Key Scheduling Algorithm (KSA), generującego klucz wewnętrzny (256-elementową permutację P) z klucza kryptograficznego o rozmiarze od 128 do 512 bitów.

Przed szyfrem strumieniowym VMPC postawiono poniższe założenia dotyczące cech bezpieczeństwa. Zgodnie z analizami omówionymi w rozdziałach 13-14, szyfr wszystkie z założeń spełnia:

- Szyfr nie może wymagać odrzucenia początkowych wartości generowanego strumienia bezpośrednio po KSA
- Prawdopodobieństwo, że wartości strumienia szyfru wejdą w krótki cykl, ma być zaniedbywalnie niskie
- Wartości strumienia generowanego przez szyfr mają być wolne od wad statystycznych
- Złożoność obliczeniowa odtworzenia klucza wewnętrznego z wartości strumienia ma być wyższa od złożoności przeszukania wszystkich możliwych kluczy kryptograficznych o rozmiarze 512 bitów
- KSA szyfru ma uniemożliwiać ataki kluczami pokrewnymi (related key attacks) oraz ataki przeciwko schematowi użycia wektora inicjującego (initialization vector, IV).
- KSA ma dostarczać nieodróżnialnej od losowej dyfuzji zmian jednego bitu oraz bajtu klucza o rozmiarze do 512 bitów na generowaną permutację P oraz na generowany przez szyfr strumień wartości

9. Opis szyfru strumieniowego VMPC

Szyfr strumieniowy VMPC generuje strumień 8-bitowych wartości, które użyte zostają do szyfrowania wiadomości według poniższego schematu:

Zmienne:

P : 256-bajtowa tablica zawierająca permutację inicjowaną przez VMPC KSA
s : 8-bitowa zmienna inicjowana przez VMPC KSA
+ : dodawanie modulo 256

```
1. n = 0

2. Powtarzaj kroki 3-6 dla kolejnych bajtów Wiadomości:
   3. s = P[ s + P[n] ]
   4. Szyfrogram = Wiadomość XOR P[P[P[s]]+1]
   5. x    = P[n]
      P[n] = P[s]
      P[s] = x
   6. n = n + 1
```

Tabela 4. Szyfr strumieniowy VMPC

10. Opis algorytmu inicjowania klucza wewnętrznego (VMPC KSA)

VMPC KSA przekształca klucz kryptograficzny i wektor inicjujący w 256-elementową permutację *P* (klucz wewnętrzny) oraz inicjuje zmienną *s*.

Zmienne jak w rozdziale 9 oraz:

c : ustalona długość klucza kryptograficznego w bajtach, $16 \le c \le 64$
K : *c*-elementowa tablica zawierająca klucz kryptograficzny
z : ustalona długość wektora inicjującego w bajtach, $16 \le z \le 64$
V : *z*-elementowa tablica zawierająca wektor inicjujący

```
1. s = 0
2. Dla n od 0 do 255: P[n]=n
3. Dla m od 0 do 767: wykonaj kroki 4-6:
   4. n = m modulo 256
   5. s = P[ s + P[n] + K[m modulo c] ]
   6. x    = P[n]
      P[n] = P[s]
      P[s] = x

7. Dla m od 0 do 767: wykonaj kroki 8-10:
    8. n = m modulo 256
    9. s = P[ s + P[n] + V[m modulo z] ]
   10. x    = P[n]
       P[n] = P[s]
       P[s] = x
```

Tabela 5. Algorytm inicjowania klucza wewnętrznego (VMPC KSA)

11. Wartości testowe szyfru VMPC

Tabela 6 zawiera 16 testowych wartości wygenerowanych przez szyfr VMPC dla podanego 128-bitowego klucza kryptograficznego (K) oraz podanego 128-bitowego wektora inicjującego (V). Symbol [hex] oznacza zapis w systemie szesnastkowym; symbol [dec] oznacza zapis w systemie dziesiętnym.

K; c = 16 [hex]	96, 61, 41, 0A, B7, 97, D8, A9, EB, 76, 7C, 21, 17, 2D, F6, C7							
V; z = 16 [hex]	4B, 5C, 2F, 00, 3E, 67, F3, 95, 57, A8, D2, 6F, 3D, A2, B1, 55							
Numer wartości strumienia [dec]	0	1	2	3	252	253	254	255
Wartość strumienia [hex]	A8	24	79	F5	B8	FC	66	A4
Numer wartości strumienia [dec]	1020	1021	1022	1023	102396	102397	102398	102399
Wartość strumienia [hex]	E0	56	40	A5	81	CA	49	9A

Tabela 6. Wartości testowe szyfru VMPC

12. Szybkość działania szyfru VMPC

Tabele 7 i 8 zawierają informacje o szybkości umiarkowanie zoptymalizowanej implementacji assemblerowej szyfru strumieniowego VMPC oraz VMPC KSA, zmierzoną na procesorze Intel Pentium 4; 2,66 GHz

Mbajtów / s	Mbitów / s	Cykli / bajt
210	1680	12.7

Tabela 7. Szybkość programowa szyfru VMPC

Kluczy / s	Milisekund / klucz
310 000	0.0032

Tabela 8. Szybkość programowa VMPC KSA

Wyniki te plasują szyfr VMPC wśród najszybszych algorytmów kryptograficznych znanych publicznie na świecie.

13. Analiza bezpieczeństwa szyfru VMPC

13.1. Teoretyczne prawdopodobieństwo równych sąsiednich wartości

Prawdopodobieństwo wygenerowania dwóch sąsiednich równych sobie wartości jest istotnym parametrem dla szyfru bazującego na wewnętrznej permutacji (P). Sama permutacja jest w oczywisty sposób odróżnialna od prawdziwie losowego strumienia wartości, ponieważ jej wartości nigdy się nie powtarzają. Konstrukcja szyfru opartego na wewnętrznej permutacji powinna zakłócać regularną strukturę permutacji tak, aby generowane wartości powtarzały się z prawdopodobieństwem nieodróżnialnym od losowego.

W tym rozdziale wykażemy teoretycznie, dlaczego prawdopodobieństwo wygenerowania przez szyfr VMPC równych sąsiednich wartości jest takie samo, jakiego oczekiwalibyśmy od generatora losowego, to jest, że Prawdop($out[x]$=$out[x+1]$) = 2^{-N}, gdzie $out[x]$ oznacza x-tą kolejna wartość strumienia generowanego przez szyfr, a N oznacza rozmiar słów, na jakich szyfr operuje w bitach (zwykle N=8).

Aby obliczyć Prawdop($out[x]$=$out[x+1]$), dwa scenariusze muszą zostać uwzględnione:
(1) – nie ma operacji zamiany elementów permutacji P w kroku 5 (tabela 4) oraz **(2)** – jest operacja zamiany.

W scenariuszu **(1)** mamy: Prawdop(brak zamiany) = Prawdop($s[x] = n[x]$) = 2^{-N}

W wyniku zajścia **(1)** permutacja P będzie miała taką sama postać w krokach x oraz x+1.
Pociąga to za sobą rozróżnienie na dwa podscenariusze: **(1a):** $s[x] = s[x+1]$ oraz **(1b):** $s[x] \neq s[x+1]$, co bezpośrednio determinuje, czy – odpowiednio – $out[x] = out[x+1]$ czy $out[x] \neq out[x+1]$.

W podscenariuszu **(1a)** mamy: Prawdop($s[x] = s[x+1]$) = 2^{-N} oraz Prawdop($out[x] = out[x+1]$) = 1
W podscenariuszu **(1b)** mamy: Prawdop($s[x] \neq s[x+1]$) = $1 - 2^{-N}$ oraz Prawdop($out[x] = out[x+1]$) = 0

W scenariuszu **(2)** mamy: Prawdop(zamiana) = Prawdop($s[x] \neq n[x]$) = $1 - 2^{-N}$
oraz Prawdop($out[x] = out[x+1]$) = 2^{-N},
bez względu na relację między $s[x]$ a $s[x+1]$, ponieważ P ma inną postać w krokach x oraz $x+1$. Prawdopodobieństwo to zostało także potwierdzone eksperymentalnie.

Łącząc prawdopodobieństwa w scenariuszach **(1a)**, **(1b)** oraz **(2)**, otrzymujemy:

$$\text{Prawdop}(out[x] = out[x+1]) = 2^{-N} \times 2^{-N} \times 1 \quad + \quad 2^{-N} \times (1 - 2^{-N}) \times 0 \quad + \quad (1 - 2^{-N}) \times 2^{-N} = \mathbf{2^{-N}}$$

co było do wykazania.

13.2. Złożoność odtworzenia klucza wewnętrznego

Metoda analogiczna do algorytmu wnioskowania Forward Tracking, zaproponowanego przez autorów [3] dla szyfru RC4, została zastosowana do złamania szyfru VMPC. Oszacowano, że przy zastosowaniu tej metody średnia złożoność obliczeniowa odtworzenia klucza wewnętrznego (P) ze strumienia wartości generowanych przez szyfr VMPC wynosi około 2^{900} operacji.

13.3. Rozkłady pojedynczych wartości, digrafów i trigrafów

Częstotliwość wystąpień każdej z możliwych 2^8 wartości wygenerowanych przez szyfr ($out[x]$) została zmierzona w próbce $2^{41.85}$ wygenerowanych wartości. Żadna ze zmierzonych częstotliwości nie wykazała statystycznie istotnego odchylenia od wartości oczekiwanej od źródła losowego (1 / 256).

Częstotliwość wystąpień każdej z możliwych 2^{16} konfiguracji wartości wygenerowanej przez szyfr oraz wartości zmiennej n ($out[x]$, n) została zmierzona w próbce $2^{39.4}$ wygenerowanych wartości. Żadna ze zmierzonych częstotliwości nie wykazała statystycznie istotnego odchylenia od wartości oczekiwanej od źródła losowego (1 / 65536).

Częstotliwość wystąpień każdej z możliwych 2^{16} par kolejnych wygenerowanej przez szyfr wartości (digraph probabilities) ($out[x]$, $out[x+1]$) została zmierzona w próbce $2^{40.1}$ wygenerowanych wartości. Żadna ze zmierzonych częstotliwości nie wykazała statystycznie istotnego odchylenia od wartości oczekiwanej od źródła losowego (1 / 65536).

Częstotliwość wystąpień każdej z możliwych 2^{24} trójek dwóch kolejnych wygenerowanych przez szyfr wartości oraz wartości zmiennej n ($out[x]$, $out[x+1]$, n) została zmierzona w próbce $2^{41.85}$ wygenerowanych wartości. Żadna ze zmierzonych częstotliwości nie wykazała statystycznie istotnego odchylenia od wartości oczekiwanej od źródła losowego (1 / 16777216).

Częstotliwość wystąpień każdej z możliwych 2^{24} trójek kolejnych wygenerowanej przez szyfr wartości (trigraph probabilities) ($out[x]$, $out[x+1]$, $out[x+2]$) została zmierzona w próbce $2^{41.6}$ wygenerowanych wartości. Żadna ze zmierzonych częstotliwości nie wykazała statystycznie istotnego odchylenia od wartości oczekiwanej od źródła losowego (1 / 16777216).

Autorzy [5] zauważają, że rozkłady ($out[x]$, $out[x+1]$) oraz [$out[x]$, $out[x+1]$, n) zmierzone dla szyfru RC4 pozwalają odróżnić generowany przez RC4 strumień wartości od ciągu losowego wykorzystując zaledwie $2^{30.6}$ wartości.

13.4. Rozkłady początkowych wartości (bezpośrednio po KSA)

Częstotliwości wystąpienia każdej z 256 możliwych postaci każdej z początkowych 256 wartości wygenerowanych przez szyfr VMPC bezpośrednio po wykonaniu procedury KSA (bez użycia wektora inicjującego) zostały zmierzone w próbce $2^{40.3}$ wygenerowanych wartości dla $2^{32.3}$ różnych kluczy. Żadna ze zmierzonych częstotliwości nie wykazała statystycznie istotnego odchylenia od wartości oczekiwanej od źródła losowego (1 / 256).

W [6] autorzy zauważają, że druga wygenerowana przez RC4 wartość ($out[2]$) bezpośrednio po KSA przyjmuje wartość 0 z prawdopodobieństwem 1/128 zamiast 1/256, co stanowi poważną wadę statystyczną RC4.

13.5. Krótkie cykle

Zaczerpnięte z [16] długości cykli zaobserwowane w szyfrze VMPC ze zmniejszoną z 256 do M wielkością słów, dla $M \in \{4,5,...,10\}$, nie wykazują istotnej różnicy od długości cykli, jakich oczekiwalibyśmy od losowej $M! \times M^2$ elementowej permutacji ($M! \times M^2$ określa liczbę możliwych postaci permutacji P i zmiennych s oraz n). Wyekstrapolowane z tego oraz z [1] zostało, iż prawdopodobieństwo, że wartości generowane przez szyfr VMPC wejdą w cykl długości nie większej niż X wynosi $X / (256! \times 256^2)$, gdzie przykładowym oszacowaniem jest, iż prawdopodobieństwo, że wartości generowane przez szyfr wejdą w cykl nie dłuższy niż 2^{850} wynosi około 2^{-850}, jest więc zaniedbywalnie niskie.

Zdefiniowane w [10] tzw. stany Finney'a, które mogą teoretycznie wystąpić dla szyfru RC4 i poprzez użycie w operacji zamiany-elementów-permutacji zawsze elementu $P[n]$=1 powodować wejście w cykl o długości 65280, nie są możliwe dla szyfru VMPC, ponieważ operacja $s = P[s + P[n]]$ w kroku 3 szyfru VMPC (tabela 4) zakłóca ewentualną liniową strukturę $P[n]$ oraz s, tym samym uniemożliwiając z definicji wystąpienie stanów Finney'a. Bardziej szczegółowa analiza tego problemu znajduje się w [16].

13.6. Sumy binarne

Test sum binarnych (binary derivatives of bit output sequences) został zaproponowany przez autora [8], który wykorzystując ten test wykazał słabość statystyczną w strumieniu wartości generowanym przez szyfr RC4, która pozwala odróżnić strumień RC4 od strumienia losowego z wykorzystaniem 2^{40} wartości (autorzy [5] uważają, że jest to nieco więcej – $2^{44.7}$). Autor [8] zauważa, że ($out_k[x]+out_k[x+2]=1$), gdzie k oznacza numer bitu x-tej wartości wygenerowanej przez szyfr, dla k=0 (k=0 oznacza najmniej znaczący bit) występuje w RC4 z częstotliwością statystycznie istotnie wyższą od oczekiwanej od źródła losowego 0.5.

Strumień wartości generowany przez szyfr VMPC został poddany analogicznej rodzinie testów: z 21 częstotliwości wystąpienia $out_k[x]=out_k[x+A]$ dla wielkości słowa w bitach N=7 oraz dla $k \in \{0,1,...,6\}$ i $A \in \{1,2,3\}$ żadna ze zmierzonych częstotliwości nie wykazała statystycznie istotnego odchylenia od wartości oczekiwanej od źródła losowego (0.5).

13.7. Standardowe testy statystyczne

Strumienie wartości generowane przez szyfr VMPC zostały przetestowane dwoma popularnymi zestawami narzędzi statystycznych – DIEHARD [11] oraz NIST [12]. Żadne odchylania od wartości oczekiwanych od źródeł losowych nie zostały znalezione przez żaden z 15 testów zawartych w pakiecie DIEHARD, ani przez żaden z 16 testów NIST.

14. Analiza bezpieczeństwa algorytmu inicjowania klucza wewnętrznego (VMPC KSA)

Algorytm VMPC KSA był testowany na występowanie efektu nieodróżnialnej od losowej dyfuzji zmian klucza kryptograficznego o rozmiarach 128, 256 oraz 512 bitów na zmiany generowanej przez KSA permutacji P oraz na zmiany generowanego przez szyfr VMPC strumienia wartości. W wyniku działania efektu dyfuzji dwie permutacje (oraz strumienie wartości) wygenerowane z dwóch kluczy różniących się jednym bitem lub bajtem, są od siebie pseudolosowo różne (relacje między nimi są nieodróżnialne od losowych).

Efekt dyfuzji mierzony był bez użycia wektora inicjującego (wykorzystując tylko kroki 1-6 KSA; tabela 5). Zastosowanie wektora inicjującego i wykonanie pozostałych kroków 7-10 KSA w oczywisty sposób dodatkowo przemieszałoby generowaną permutację, co zintensyfikowałoby efekt dyfuzji. W praktycznych zastosowaniach kryptograficznych użycie wektora inicjującego jest powszechne, można zatem uznać, że efekt dyfuzji zapewniany przez VMPC KSA posiada znaczny margines bezpieczeństwa. Rozdziały 14.1 – 14.3 opisują przeprowadzone testy na dyfuzję bez użycia wektora inicjującego.

Dokładne specyfikacje testów opisanych w rozdziałach 14.1 – 14.3, wraz z wartościami oczekiwanych prawdopodobieństw, znajdują się na stronie internetowej poświęconej VMPC, www.vmpcfunction.com [19].

14.1. Dane liczby równych elementów permutacji

Częstotliwości wystąpienia sytuacji, gdzie w dwóch permutacjach wygenerowanych z kluczy różniących się jednym bajtem, występuje dana liczba (0,1,2,3,4,5) równych elementów na odpowiadających pozycjach oraz średnia liczba równych elementów na odpowiadających pozycjach – nie wykazały statystycznie istotnych odchyleń od wartości oczekiwanych od źródła losowego w testach $2^{33.2}$ par kluczy o rozmiarze 128, 256 oraz 512 bitów.

14.2. Dane liczby równych wartości strumienia generowanego przez szyfr VMPC

Częstotliwości wystąpienia sytuacji, gdzie w dwóch 256-bajtowych strumieniach wygenerowanych przez szyfr VMPC bezpośrednio po KSA dla kluczy różniących się jednym bajtem, występuje dana liczba (0,1,2,3,4,5) równych wartości na odpowiadających pozycjach strumienia oraz średnia liczba równych wartości na odpowiadających pozycjach strumienia – nie wykazały statystycznie istotnych odchyleń od wartości oczekiwanych od źródła losowego testach $2^{33.2}$ par kluczy o rozmiarze 128, 256 oraz 512 bitów.

14.3. Częstotliwości równych odpowiadających sobie elementów permutacji

Częstotliwości występowania sytuacji, gdzie odpowiadające sobie elementy dwóch permutacji wygenerowanych z kluczy różniących się jednym bajtem są równe (dla każdej z 256 możliwych pozycji) – nie wykazały statystycznie istotnych odchyleń od wartości oczekiwanych od źródła losowego testach $2^{33.2}$ par kluczy o rozmiarze 128, 256 oraz 512 bitów.

15. Dowód bezpieczeństwa kryptosystemu VMPC

Kryptosystem VMPC określa sposób przesłania przez Stronę *A* Stronie *B* j wiadomości zaszyfrowanych tym samym kluczem K z wykorzystaniem komponentów VMPC. Za [18], po przyjęciu podstawowych założeń, możliwe jest udowodnienie, że złamanie kryptosystemu VMPC jest możliwe tylko metodą „brute force" – przeszukania wszystkich możliwych wartości klucza *K*.

15.1. Kryptosystem VMPC

Oznaczenia:

$Wiadomość_i[m]$: m-ty bajt i-tej wiadomości
$Szyfrogram_i[m]$: m-ty bajt zaszyfrowanej i-tej wiadomości
P_i : 256-elementowa tablica zawierająca permutację
s_i : 8-bitowa zmienna
K : c-bajtowa tablica zawierająca klucz; $16 \leq c \leq 64$
V_i : z-bajtowa tablica zawierająca wektor inicjujący; $16 \leq z \leq 64$
\+ : dodawanie modulo 256

1. Strony *A* i *B* uzgadniają klucz K, który jest znany tylko *A* i *B*
2. Strona *A* przekazuje j wiadomości Stronie *B* wykonując kroki 3-8:
3. Dla i od 1 do j: wykonaj kroki 4-8:
4. Wygeneruj wektor inicjujący V_i tak, aby $V_i \neq V_d$ dla dowolnych $d \neq i$
5. Użyj K i V_i w VMPC KSA wykonując kroki 5.1 – 5.4.2:
   $[P_i=KSA(KSA(KSA(P^{(0)}, K),V_i),K)]$:

   5.1. $s_i = 0$; Dla n od 0 do 255: $P^{(0)}[n]=n$; $P_i=P^{(0)}$
   5.2. Dla m od 0 do 767: wykonaj kroki 5.2.1 - 5.2.2:
        5.2.1. $s_i = P_i[s_i + P_i[m \bmod 256] + K[m \bmod c]]$
        5.2.2. $x = P_i[m \bmod 256]$; $P_i[m \bmod 256] = P_i[s_i]$; $P_i[s_i] = x$
   5.3. Dla m od 0 do 767: wykonaj kroki 5.3.1 - 5.3.2:
        5.3.1. $s_i = P_i[s_i + P_i[m \bmod 256] + V_i[m \bmod z]]$
        5.3.2. $x = P_i[m \bmod 256]$; $P_i[m \bmod 256] = P_i[s_i]$; $P_i[s_i] = x$
   5.4. Dla m od 0 do 767: wykonaj kroki 5.4.1 - 5.4.2:
        5.4.1. $s_i = P_i[s_i + P_i[m \bmod 256] + K[m \bmod c]]$
        5.4.2. $x = P_i[m \bmod 256]$; $P_i[m \bmod 256] = P_i[s_i]$; $P_i[s_i] = x$

6. Zaszyfruj i-tą wiadomość szyfrem VMPC wykonując kroki 6.1 – 6.1.3:
   $[Szyfrogram_i = Wiadomość_i$ xor $SC(P_i)]$:

   6.1. Dla m od 0 do ($Długość\ Wiadomości_i$)-1: wykonaj kroki 6.1.1 - 6.1.3:
        6.1.1. $s_i = P_i[s_i + P_i[m \bmod 256]]$
        6.1.2. $Szyfrogram_i[m] = Wiadomość_i[m]$ xor $P_i[P_i[P_i[s_i]]+1]$
        6.1.3. $x = P_i[m \bmod 256]$; $P_i[m \bmod 256] = P_i[s_i]$; $P_i[s_i] = x$

7. Dopisz V_i na początku $Szyfrogramu_i$
8. Prześlij $Szyfrogram_i$ Stronie *B*

15.2. Dowód bezpieczeństwa kryptosystemu VMPC

Przyjmijmy, że „określić *X*" oznacza odkryć jakiekolwiek informacje o *X* (niekoniecznie postać całego *X*).

Definicja 1. Złamanie kryptosystemu: określenie $Wiadomo\acute{s}ci_i$ na podstawie $Szyfrogramu_i$ oraz $Szyfrogramu_d$ i $Wiadomo\acute{s}ci_d$ dla dowolnych wartości $d \neq i$

Definicja 2. Atak na kryptosystem: algorytm odtwarzający postać P_d na podstawie $SC(P_d)$, gdzie $SC(P_d) = Szyfrogram_d$ xor $Wiadomo\acute{s}\acute{c}_d$

Założenie 1. Określenie $Wiadomo\acute{s}ci_i$, gdzie $Wiadomo\acute{s}\acute{c}_i = Szyfrogram_i$ xor $SC(P_i)$, wymaga jakichkolwiek wiedzy o $SC(P_i)$.

Założenie 2. Określenie $SC(P_i)$ wymaga jakiejkolwiek wiedzy o P_i

Założenie 3. Określenie P_i, gdzie $P_i = KSA(KSA(KSA(P^{(0)}, K), V_i), K)$, wymaga jakiejkolwiek wiedzy o K, nawet jeśli P_d jest znane dla dowolnych wartości $d \neq i$

Założenie 4. Jedynym źródłem informacji, z których K może być określone, jest $P_d = KSA(KSA(KSA(P^{(0)}, K), V_d), K)$ dla $d \neq i$

Założenie 5. Określenie K z P_d z prawdopodobieństwem wyższym niż $2^{-8\times c}$ wymaga przeszukiwania wszystkich możliwych $2^{8\times c}$ wartości K, aż jedna z wartości będzie zgodna z $P_d = KSA(KSA(KSA(P^{(0)}, K), V_d), K)$

Twierdzenie 1. Złamanie kryptosystemu wymaga przeszukania wszystkich możliwych $2^{8\times c}$ kluczy K, przyjmując, że założenia 1 – 5 są prawdziwe.

Dowód. Zgodnie z Założeniem 1 złamanie kryptosystemu wymaga określenia $SC(P_i)$; zgodnie z Założeniem 2 określenie $SC(P_i)$ wymaga określenia P_i; zgodnie z Założeniem 3 określenie P_i wymaga określenia K; zgodnie z Założeniem 4 określenie K wymaga określenia P_d dla d $\neq i$, gdzie P_d jest przyjęte jako znane w wyniku skutecznego ataku. Zgodnie z Założeniem 5 określenie K na podstawie P_d wymaga średnio $2^{8\times c-1}$ operacji, co jest równoznaczne z przeszukiwaniem wszystkich możliwych wartości klucza K.

15.3. Uzasadnienie założeń

Założenie 1. Zgodnie z analizami w rozdziale 13 odróżnienie $SC(P_i)$ od ciągu losowego jest niemożliwe w praktyce. W konsekwencji $Szyfrogram_i = Wiadomo\acute{s}\acute{c}_i$ xor $SC(P_i)$ jest także nieodróżnialny od ciągu losowego, co uniemożliwia wywnioskowania jakichkolwiek informacji o $Wiadomo\acute{s}ci_i$ bezpośrednio z $Szyfrogramu_i$.

Spośród pozostałych w opisanym kryptosystemie źródeł informacji tylko $SC(P_i)$ wydaje się być wystarczająco powiązane z $Wiadomo\acute{s}ci_i$, aby umożliwić określenie $Wiadomo\acute{s}ci_i$.

Założenie 2. Zgodnie z analizami w rozdziale 13 odróżnienie $SC(P_i)$ od ciągu losowego jest niemożliwe praktycznie. W konsekwencji $Szyfrogram_i = Wiadomo\acute{s}\acute{c}_i$ xor $SC(P_i)$ jest także nieodróżnialny od ciągu losowego, co uniemożliwia wywnioskowanie jakichkolwiek informacji o $SC(P_i)$ bezpośrednio z $Szyfrogramu_i$.

Zgodnie z efektem dyfuzji opisanym w Rozdziale 14 relacje między P_i a P_d oraz relacje między $SC(P_i)$ a $SC(P_d)$ są nieodróżnialne od losowych, co uniemożliwia wywnioskowanie jakichkolwiek informacji o $SC(P_i)$ z $SC(P_d)$ dla $d \neq i$.

Spośród pozostałych w opisanym kryptosystemie źródeł informacji tylko P_i wydaje się być wystarczająco powiązane z $SC(P_i)$, aby umożliwić określenie $SC(P_i)$.

Założenie 3. Zgodnie z efektem dyfuzji opisanym w rozdziale 14 relacje między $KSA(KSA(P^{(0)},K),V_i)$ a $KSA(KSA(P^{(0)},K),V_d)$ oraz, w wyniku kroków 5.4 – 5.4.2, relacje między P_i a P_d są nieodróżnialne od losowych dla $V_i \neq V_d$, co uniemożliwia wywnioskowanie jakichkolwiek informacji o P_i z P_d.

Spośród pozostałych w opisanym kryptosystemie źródeł informacji tylko K wydaje się być wystarczająco powiązane z P_i, aby umożliwić określenie P_i.

Założenie 4. W opisanym kryptosystemie K nie jest użyte w żadnym innym miejscu oprócz wyliczenia $P_d = KSA(KSA(KSA(P^{(0)},K),V_d),K)$ w krokach 5.1 – 5.4.2 dla $d \in \{1,2,...,j\}$

Założenie 5. $X = KSA(KSA(P^{(0)},K),V_d)$, gdzie $P^{(0)}$ i V_d są znane, jest nieznane, dlatego każda z $2^{8\times c}$ możliwych wartości K jest równie prawdopodobnie zgodna z $P_d = KSA(X,K) = KSA(KSA(KSA(P^{(0)},K),V_d),K)$, gdzie P_d jest uznane jako dane w wyniku skutecznego ataku.

15.4. Znaczenie dowodu

Kluczowym elementem, który umożliwił zbudowanie dowodu dla opisanego kryptosystemu, jest trzecia faza KSA (kroki 5.4 – 5.4.2). Bez tej fazy skuteczny atak, a więc algorytm odtwarzający P_d z $SC(P_d)$, pozwoliłby w relatywnie prosty sposób odtworzyć K na podstawie P_d, co w już bezpośredni sposób pozwala deszyfrować nowe wiadomości szyfrowane kluczem K.

Dodatkowe przekształcenie P_d w krokach 5.4 – 5.4.2 z nieznanym parametrem K sprawia, że odtworzenie K na podstawie P_d staje się trudne i jak opisano w uzasadnieniu założenia 5 – jest możliwe tylko metodą przeszukania wszystkich możliwych wartości K.

Opisany kryptosystem pozostaje więc bezpieczny, nawet jeśli skuteczny atak (skuteczny algorytm dotwarzający P_d z $SC(P_d)$) zostałby znaleziony. Jednak, jak opisano w rozdziale 13.2, najszybszy znany algorytm odtwarzający P_d z $SC(P_d)$ wymaga średnio 2^{900} operacji. Opisany kryptosystem zapewnia w ten sposób dwu-warstwowy poziom bezpieczeństwa – odnalezienie P_d z $SC(P_d)$ wymaga średnio 2^{900} operacji, ale nawet jeśli praktyczny algorytm realizujący to zadanie zostałby znaleziony, kryptosystem i tak pozostaje bezpieczny na mocy opisanego w rozdziale 15.2 dowodu – dzięki trójfazowej konstrukcji algorytmu KSA.

Mimo że przedstawiony dowód nie jest bezpośredni, ale bazuje na pewnych założeniach, to opisany kryptosystem oferuje istotnie wyższy poziom bezpieczeństwa niż inne kryptosystemy, dla których znalezienie skutecznego algorytmu odtwarzającego klucz wewnętrzny na podstawie generowanego strumienia (P_d z $SC(P_d)$) oznacza w praktyce ich złamanie. W przypadku kryptosystemu VMPC wykazano, że zaistnienie takiej sytuacji nie stanowi zagrożenia dla szyfrowanych danych.

16. Uwierzytelnione szyfrowanie z wykorzystaniem schematu VMPC-Tail-MAC

Uwierzytelnione szyfrowanie (authenticated encryption) jest istotnym wymogiem stawianym przed praktycznymi zastosowaniami algorytmów szyfrujących. Realizowane jest poprzez dodanie do szyfrogramu kodu uwierzytelniającego wiadomość (Message Authentication Code, MAC), który stanowi zależną od klucza sumę kontrolną szyfrowanej wiadomości. MAC pozwala zweryfikować, czy uzyskana po deszyfracji wiadomość jest rzeczywiście tą samą wiadomością, która została zaszyfrowana oraz zweryfikować, czy do deszyfrowania użyty został ten sam klucz, co do szyfrowania.

Zasadniczym wymogiem bezpieczeństwa dla algorytmów obliczania MACów jest odporność na sfałszowanie wiadomości. Algorytm obliczania MACów powinien być skonstruowany tak, aby modyfikacja szyfrogramu bez jednoczesnej zmiany MACa była niemożliwa w praktyce.

Opisana w [17] koncepcja Tail-MAC realizuje to zadanie w efektywny sposób dzięki wykorzystaniu części danych obliczonych już przez sam algorytm szyfrowania, z jakim jest zintegrowana. Tail-MAC jest koncepcją ogólną i może być zastosowany dla dowolnych szyfrów strumieniowych, jednak ze względu na ograniczenia objętościowe zostanie tutaj omówiony na przykładzie konkretnego systemu uwierzytelnionego szyfrowania, opartego na szyfrze VMPC i określanego jako VMPC-Tail-MAC.

16.1. Schemat VMPC-Tail-MAC

Zmienne:

P : 256-elementowa tablica zawierająca permutację
s : 8-bitowa zmienna
T : 32-bajtowa tablica
x_1, x_2, x_3, x_4 : 1-bajtowe zmienne
M : 20-bajtowa tablica, w której umieszczony zostanie 160-bitowy MAC
$+$: dodawanie modulo 256

1. $(x_1, x_2, x_3, x_4, n, m, g)=0$; $T[x]=0$ dla $x=0,1,...,31$

2. Powtórz kroki 3-16 (*Długość Wiadomości*) razy:
   3. $s = P[\ s + P[n]\ ]$
   4. $Szyfrogram[m] = Wiadomość[m]$ xor $P[P[P[s]]+1]$

   5. $x_4 = x_4 + x_3$
   6. $x_3 = x_3 + x_2$
   7. $x_2 = x_2 + x_1$
   8. $x_1 = P[\ x_1 + s + Szyfrogram[m]\ ]$

   9. $T[g] = T[g]$ xor x_1
   10. $T[g+1] = T[g+1]$ xor x_2
   11. $T[g+2] = T[g+2]$ xor x_3
   12. $T[g+3] = T[g+3]$ xor x_4

   13. $x = P[n]$; $P[n] = P[s]$; $P[s] = x$
   14. $g = (g + 4)$ modulo 32
   15. $n = n + 1$
   16. Zwiększ wartość m o 1

17. Powtórz kroki (3, 5–15)(bez 4 i bez 16) 32 razy dla $Szyfrogramu[m]=0$

18. Umieść T w K; ustaw $c = 32$. Wykonaj krok 3 VMPC KSA (tabela 5)

19. Umieść 20 nowych wartości wygenerowanych przez szyfr VMPC (tabela 4) w tablicy M (Dla n od 0 do 19: wykonaj kroki 19.1 - 19.3:
    19.1. $s = P[\ s + P[n]\ ]$
    19.2. $M[n] = P[P[P[s]]+1]$
    19.3. $x = P[n]$; $P[n] = P[s]$; $P[s] = x$)

20. Dopisz MAC, umieszczony w 20-bajtowej tablicy M, na końcu $Szyfrogramu$

16.2. Szybkość działania schematu VMPC-Tail-MAC

Tabele 9 zawiera informacje o szybkości umiarkowanie zoptymalizowanej implementacji assemblerowej schematu VMPC-Tail-MAC, zmierzoną na procesorze Intel Pentium 4; 2,66 GHz

Mbajtów / s	Mbitów / s	Cykli / bajt
127	1016	20.9

Tabela 9. Szybkość programowa VMPC-Tail-MACa

Szybkość na poziome około 21 cykli na bajt plasuje VMPC-Tail-MAC wśród najszybszych znanych publicznie na świecie schematów uwierzytelnionego szyfrowania.

16.3. Analiza bezpieczeństwa schematu VMPC-Tail-MAC

Schemat Tail-MAC został zaprojektowany tak, aby utrzymywać odpowiednio długą pamięć informacji uzyskanych z szyfrowanej wiadomości oraz z użytego klucza i wektora inicjującego (reprezentowanych przez postać permutacji P) w zmiennych x_1, x_2, x_3, x_4 oraz zostawiać ślady tych zmiennych w tablicy T w sposób na tyle intensywny, aby w praktyce uniemożliwić taką zmianę szyfrogramu, która mogłaby pozostawić tablicę T niezmienioną. Zgodnie z dotychczasowymi analizami złożoność najefektywniejszego algorytmu uzyskania niezmienionej tablicy T dla zmienionego *Szyfrogramu* wynosi 2^{144}.

Konstrukcja kroku 8 ($x_1 = P[x_1 + s + Szyfrogram[m]]$), poprzez odwołanie się do nieznanej i pseudolosowej permutacji P oraz zmiennej s, zakłóca ewentualne regularności, jakie atakujący mógłby chcieć przenieść z wybranego/manipulowanego przez siebie szyfrogramu na zmienną x_1 i w konsekwencji na zmienne x_2, x_3, x_4 oraz tablicę T.

Istotną właściwością opisanego schematu jest łańcuchowa nieuniknioność zmian – jeśli zmianie ulegnie którykolwiek bajt szyfrogramu – zmiana ta spowoduje nieuniknioną zmianę zmiennej x_1 w danym kroku. Nieuniknionej zmianie ulegnie jednak także zmienna x_2 w kroku następnym, zmienna x_3 w kroku kolejnym i x_4 w jeszcze dalszym. Jednocześnie przy każdym kroku zmienne x_1, x_2, x_3, x_4 modyfikują tablicę T, co sprawia, że odwrócenie tych zmian poprzez manipulowanie szyfrogramem staje się bardzo złożone. Szczegółowa analiza zjawiska wskazuje, że problem ten ma złożoność obliczeniową na poziomie 2^{144} i że poziom ten może być łatwo zwiększony poprzez zwiększenie liczby zmiennych x_n (np. z 4 do 5) i zwiększenie rozmiaru tablicy T (np. z 4×8=32 do 5×8=40 bajtów).

Zastosowanie fazy kończącej (kroki 17,18,19) ma na celu przede wszystkim zapewnienie, że tablica T, zanim zostanie dopisana do *Szyfrogramu*, będzie przemieszana w pseudolosowy sposób, co eliminuje ryzyko ewentualnego wycieku informacji o P przez ujawnienie M w *Szyfrogramie* oraz ryzyko przeniesienia ewentualnych regularności tablicy T na wynikową tablicę M.. W ten sposób korelacja między szyfrowaną wiadomością czy uzyskaną z klucza i wektora inicjującego permutacją P a powstałym MACem (tablicą M) jest nieodróżnialna od losowej i przez to trudna do kontrolowania przez atakującego.

Najbardziej efektywny algorytm fałszujący wiadomość szyfrowaną schematem VMPC-Tail-MAC został opisany dokładnie w [17]. Algorytm ten, dla czterech zmiennych x_n (x_1, x_2, x_3, x_4) ma złożoność na poziomie 2^{144}, podczas gdy dodanie jednej zmiennej x_n (i korzystanie w analogiczny sposób ze zmiennych x_1, x_2, x_3, x_4, x_5) oraz zwiększenie rozmiaru tablicy T z 32 do 40 bajtów powoduje zwiększenie tej złożoności do 2^{200}. Jednak już wartość 2^{144} można uznać za całkowicie wystarczający poziom bezpieczeństwa, który znacznie przewyższa aktualnie standardowy w zastosowaniach kryptograficznych poziom 2^{128}.

17. Podsumowanie

Zaprezentowana została bardzo prosta i łatwa do implementacji programowej funkcja jednokierunkowa VMPC. Otwartym problemem jest, czy być może prostota funkcji pozwoli w przyszłości zbudować formalny dowód jej praktycznej jednokierunkowości. Przedstawiony został bardzo prosty w konstrukcji szyfr strumieniowy bazujący na funkcji VMPC wraz z procedurą inicjowania klucza wewnętrznego, VMPC KSA. Przeprowadzone analizy wskazują, że szyfr VMPC wraz z jego KSA stanowią bezpieczny algorytm szyfrowania danych, charakteryzujący się szeregiem przewag w bezpieczeństwie nad popularnych szyfrem RC4 oraz bardzo wysoką wydajnością w implementacji programowej.

Zaproponowany został kryptosystem oparty na szyfrze VMPC i jego KSA, dla którego, po przyjęciu podstawowych założeń, możliwe jest udowodnienie bezpieczeństwa i wykazanie, iż niemożliwe jest złamanie kryptosystemu metodą inną niż metoda przeszukania wszystkich możliwych kluczy.

Przedstawiony został bardzo prosty w konstrukcji schemat uwierzytelnionego szyfrowania VMPC-Tail-MAC, którego analiza wskazuje, iż schemat ten jest zarówno bezpieczny, jak i bardzo wydajny w implementacjach programowych.

18. Bibliografia

[1] Donald E. Knuth, „The Art of Computer Programming", vol. 1. *Fundamental Algorithms*, Third Edition, Addison Wesley Longman, 1997.

[2] Donald E. Knuth, „The Art of Computer Programming", vol. 2. *Seminumerical Algorithms*, Third Edition, Addison Wesley Longman, 1998.

[3] Serge Mister, Stafford E. Tavares, „Cryptanalysis of RC4-like Ciphers", Proceedings of SAC 1998, LNCS, vol. 1556, Springer-Verlag, 1999.

[4] Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, Sven Verdoolaege, „Analysis Methods for (Alleged) RC4", Proceedings of ASIACRYPT 1998, LNCS, vol. 1514, Springer-Verlag, 1998.

[5] Scott R. Fluhrer, David A. McGrew, „Statistical Analysis of the Alleged RC4 Keystream Generator", Proceedings of FSE 2000, LNCS, vol. 1978, Springer-Verlag, 2001.

[6] Itsik Mantin, Adi Shamir, „A Practical Attack on Broadcast RC4", Proceedings of FSE 2001, LNCS, vol. 2355, Springer-Verlag, 2002.

[7] Scott Fluhrer, Itsik Mantin, Adi Shamir, „Weaknesses in the Key Scheduling Algorithm of RC4", Proceedings of SAC 2001, LNCS, vol. 2259, Springer-Verlag 2001.

[8] Jovan Dj. Golic, „Linear Statistical Weakness of Alleged RC4 Keystream Generator", Proceedings of EUROCRYPT 1997, LNCS, vol. 1233, Springer-Verlag 1997.

[9] Alexander L. Grosul, Dan S. Wallach, „A Related-Key Cryptanalysis of RC4", Technical Report TR-00-358, Department of Computer Science, Rice University, 2000.

[10] Hal Finney, „An RC4 Cycle That Can't Happen", post in sci.crypt, 1994.

[11] DIEHARD battery of statistical tests with documentation, http://stat.fsu.edu/~geo/diehard.html

[12] NIST statistical tests suite with documentation, http://csrc.nist.gov/rng

[13] Federal Information Processing Standards Publication 198: „The Keyed-Hash Message Authentication Code (HMAC)", 2002 http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf

[14] Mihir Bellare, Chanathip Namprempre, Authenticated Encryption, „Relations among Notions and Analysis of the Generic Composition Paradigm", ASIACRYPT 2000, LNCS vol. 1976 Springer-Verlag, 2000.

[15] T. Bellare, J. Guerin, and P. Rogaway, „XOR MACs: New methods for message authentication using finite pseudorandom functions", Proceedings of CRYPTO 1995, LNCS vol. 963, Springer-Verlag, 1994.

[16] Bartosz Zoltak „VMPC One-Way Function and Stream Cipher", FSE 2004 Conference, proceedings to appear in LNCS, Springer-Verlag

[17] Bartosz Zoltak, „Tail-MAC Scheme for Stream Ciphers and Example Application with VMPC", FSE 2004 Rump Session and IACR ePrint Archive http://eprint.iacr.org/2004/048

[18] Bartosz Zoltak, „A Proof of Security for VMPC Cryptosystem", http://www.vmpcfunction.com/vmpc_proof.pdf

[19] http://www.vmpcfunction.com