Rok w kryptoanalizie VMPC, schemat uwierzytelnionego szyfrowania VMPC-MAC oraz algorytm VMPC-HASH

Bartosz Żółtak
www.VMPCfunction.com
bzoltak@vmpcfunction.com

Funkcja jednokierunkowa VMPC oraz bazujący na niej szyfr strumieniowy VMPC, opracowane przez autora, zostały zaprezentowane na międzynarodowej konferencji kryptograficznej organizowanej przez IACR, Fast Software Encryption 2004 (FSE'04).

Na tegorocznej edycji konferencji FSE'05 pojawiła się praca autorstwa Alexandra Maximova z Lund University w Szwecji, opisująca atak odróżniający strumień generowany przez szyfr VMPC od strumienia idealnie losowego (distinguisher) o złożoności 2^{54}. W referacie atak ten zostanie opisany.

Zaprezentowany zostanie także atak przeciwko zaprezentowanemu na Enigmie'04 schematowi uwierzytelnionego szyfrowania Tail-MAC, uzyskujący kolizję w kodach MAC z prawdopodobieństwem 2^{-32} oraz atak wymuszający powstanie wyraźnych zależności liniowych w stanie wewnętrznym schematu (internal state).

Zaproponowany zostanie zmodyfikowany schemat, VMPC-MAC, będący rozwinięciem projektu, odpornym na obydwa ataki przeciwko schematowi Tail-MAC.

Opisany zostanie także sposób, jak można przedefiniować schemat VMPC-MAC tak, aby działał jak funkcja skrótu (funkcja hashująca). Algorytm uzyskany w ten sposób określony został mianem VMPC-HASH.

Przedstawione zostaną także wyniki i zaleceniami autora odnośnie długości i jakości stosowanych haseł (kluczy)
– zarówno dla celów kryptograficznych (na przykładzie aplikacji do szyfrowania danych VMPC Data Security)
– jak i generalnie dla wszystkich zastosowań wymagających podania hasła (np. logowanie).

1. Wstęp

Technologia szyfrowania VMPC została zaprezentowana na międzynarodowej konferencji kryptograficznej Fast Software Encryption 2004, Delhi, Indie, 5-7 lutego 2004 oraz (wraz ze schematem uwierzytelnionego szyfrowania Tail-MAC) na VIII Krajowej Konferencji Zastosowań Kryptografii Enigma 2004.

Nazwa VMPC jest anglojęzycznym skrótem od słów Variably Modified Permutation Composition – zmiennie modyfikowane złożenie permutacji i odnosi się do funkcji jednokierunkowej VMPC, na której bazuje szyfr strumieniowy VMPC. Funkcja VMPC jest bardzo prostym przekształceniem permutacji (do wyliczenia jednego elementu permutacji wynikowej wystarczą zaledwie 3 podstawowe instrukcje procesora), natomiast odwrócenie funkcji, dla 256-elementowych permutacji, wymaga średniego nakładu mocy obliczeniowej na poziomie 2^{260} operacji.

Jako praktyczne zastosowanie funkcji VMPC w kryptografii zaproponowany został szyfr strumieniowy VMPC. Bazuje on na wewnętrznej 256-elementowej permutacji. Permutacja ta jest inicjowana algorytmem inicjowania klucza VMPC-KSA (ang. Key Scheduling Algorithm) i ma w praktyce strukturę nieodróżnialną od struktury permutacji losowej. Permutacja ta jest podczas szyfrowania dynamicznie przekształcana, co pozwala generować strumienie wartości o dowolnej w praktyce długości bez obawy o wejście w krótki cykl oraz bez praktycznej obawy, że generowany strumień zostanie odróżniony od strumienia danych losowych.

Konstrukcyjnie szyfr VMPC jest bliski popularnemu algorytmowi RC4 (także bazuje na wewnętrznej, dynamicznie przekształcanej 256-elementowej permutacji). Wg przeprowadzonych testów statystycznych na generowanym strumieniu danych oraz analiz bezpieczeństwa procedury inicjowania klucza (KSA) szyfr VMPC nie wykazuje wad statystycznych charakterystycznych dla RC4, a jego procedura inicjowania klucza nie pozwala na zastosowanie ataków kluczami pokrewnymi (related key) oraz ataków przeciwko schematowi używania wektora inicjującego (Initialization Vector) – taże charakterystycznych dla RC4. Można zatem uznać, że szyfr VMPC stanowić może dobrze uzasadniony substytut w tych zastosowaniach kryptograficznych, gdzie stosuje się lub rozważa się zastosowanie szyfru RC4, a także tam, gdzie priorytetem jest wydajność w implementacjach programowych oraz prostota implementacji.

Schemat Tail-MAC został zaproponowany jako rozwinięcie szyfru VMPC w algorytm pozwalający obliczać kody uwierzytelniające wiadomość (Message Authentication Codes). Schemat ten wykorzystuje dodatkową 32-bajtową tablicę, dynamicznie aktualizowaną na podstawie szyfrowanych danych oraz wewnętrznych zmiennych szyfru i schematu. Tablica ta jest następnie przekształcana w 160-bitowy ciąg bajtów, stanowiący wynikową wartość kodu uwierzytelniającego wiadomość.

W referacie przedstawione zostaną wyniki kryptoanalizy szyfru strumieniowego VMPC oraz schematu Tail-MAC wraz z propozycją algorytmu VMPC-MAC, który jest ewolucyjnym rozwinięciem Tail-MACa uodparniającym go na obydwa ataki znalezione przeciwko schematowi Tail-MAC. Zaprezentowane zostaną także analizy statystyczne schematu VMPC-MAC, z których wynika, że schemat zapewnia prawidłowy efekt dyfuzji z dodatkowym marginesem bezpieczeństwa.

Zaprezentowany zostanie także sposób, jak schemat VMPC-MAC można przedefiniować jedynie na poziomie określenia danych wejściowych, aby uzyskać funkcję skrótu (hashującą), nazwaną mianem VMPC-HASH.
Przeprowadzone analizy bezpieczeństwa schematu VMPC-MAC mogą być przeniesione bezpośrednio na ocenę głównych cech bezpieczeństwa algorytmu VMPC-HASH.

Przedstawione zostaną także wyniki pomiarów i wnioski autora odnośnie stosowania krótkich haseł. Na przykładzie aplikacji do szyfrowania danych VMPC Data Security zaprezentowane zostaną wyniki realnych ataków przeciwko hasłom o długości od 3 do 7 znaków – wykonanych metodą brutalnej siły (brute force). Wyniki są w pewnym stopniu zaskakujące, nie dlatego, że są odkrywcze z naukowego punktu widzenia, ale dlatego, że mimo swojej warsztatowej prostoty pozwalają rzeczywiście łamać krótkie hasła, bardzo powszechnie stosowane przez użytkowników komputerów do różnych celów, jak szyfrowanie danych, tworzenie szyfrowanych partycji czy logowanie.

2. Zarys wydarzeń w kryptoanalizie VMPC

Na tegorocznej edycji konferencji Fast Software Encryption (FSE'05, 21-23 lutego 2005, Paryż) Alexander Maximov, doktorant prof. Thomasa Johanssona w Lund University w Szwecji, zaprezentował atak typu distinguisher przeciwko szyfrowi strumieniowemu VMPC. Atak ten pozwala odróżnić strumień danych generowany przez szyfr VMPC od strumienia danych idealnie losowych po obserwacji 2^{54} bajtów wygenerowanych przez szyfr. W referacie atak ten zostanie przybliżony wraz z wnioskami autora odnośnie jego znaczenia dla praktycznego bezpieczeństwa szyfru VMPC.

Następną część referatu stanowią ataki przeciw schematowi Tail-MAC, które pozwoliły zaproponować ulepszoną wersję tego schematu, nazwaną VMPC-MAC i niwelującą obydwa znalezione ataki. Pierwszy z ataków pozwala na uzyskanie kolizji w wewnętrznej tablicy schematu, a tym samym w generowanych kodach, z prawdopodobieństwem 2^{-32}. Drugi atak pozwala na znaczne ograniczenie efektu dyfuzji i powstanie wyraźnych zależności liniowych pomiędzy wewnętrznymi zmiennymi schematu poprzez zastosowanie wiadomości o odpowiednich wzajemnych relacjach.

Ulepszony schemat VMPC-MAC odznacza się nie tylko odpornością na opisane dwa ataki, ale także znacznie poprawionym efektem dyfuzji, posiadającym znaczną rezerwę bezpieczeństwa (testy dyfuzji przeprowadzone zostały bez uwzględnienia ostatniej fazy „mieszania" tablicy przy pomocy algorytmu VMPC-KSA. Faza ta posiada już przetestowane właściwości efektu dyfuzji (omówione w [1] oraz [2])).

3. 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 (patrz rozdział 4)
s : 8-bitowa zmienna inicjowana przez VMPC KSA
$+$: dodawanie modulo 256

```
1. n = 0

2. Powtarzaj kroki 3-7 dla kolejnych bajtów Wiadomości:
   3. s = P[ s + P[n] ]
   4. X = P[P[P[s]]+1]
   5. Szyfrogram = Wiadomość XOR X
   6. k    = P[n]
      P[n] = P[s]
      P[s] = k
   7. n = n + 1
```

Tabela 1. Szyfr strumieniowy VMPC

4. 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 3 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. k    = P[n]
      P[n] = P[s]
      P[s] = k

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. k    = P[n]
       P[n] = P[s]
       P[s] = k
```

Tabela 2. Algorytm inicjowania klucza wewnętrznego (VMPC KSA)

5. Ataki typu distinguisher

Ataki typu distinguisher należą do klasy ataków ze znanym tekstem jawnym (do ich przeprowadzenia potrzebna jest znajomość zarówno szyfrogramu, jak i tekstu jawnego). Celem ataku jest wykazanie odchylenia pewnych własności statystycznych (zwykle częstotliwości występowania pewnego zjawiska) w generowanym przez szyfr strumieniu danych w porównaniu z losowym rozkładem prawdopodobieństwa.

Jeśli okaże się, że jakieś zjawisko występuje w generowanym strumieniu z prawdopodobieństwem różnym od losowego, wówczas w wyniku ataku typu distinguisher uzyskujemy wniosek postaci „ten strumień danych nie pochodzi ze źródła losowego, lecz prawdopodobnie z danego szyfru".

Ataki typu distinguisher są najsłabszymi lub jednymi z najsłabszych w kryptoanalizie – nie prowadzą bowiem do ujawnienia żadnych danych o kluczu kryptograficznym lub o tekście jawnym, a dodatkowo z założenia wymagają znajomości tekstu jawnego. Z punktu widzenia teoretycznej kryptoanalizy są jednak interesującymi rezultatami.

Podstawowym pojęciem w atakach typu distinguisher jest odchylenie **e** (bias). Określa ono wartość bezwzględną różnicy pomiędzy prawdopodobieństwem zajścia danego zdarzenia **X** w strumieniu generowanym przez szyfr a strumieniu idealnie losowych danych:

$$\mathbf{e = | P(X)_{szyfr} - P(X)_{rand} |}$$

Parametrem bezpośrednio powiązanym z wartością odchylenia jest wielkość próbki **N**, jaką trzeba zaobserwować, aby zaobserwowane odchylenie było statystycznie istotne. Wielkość próbki potrzebna do wykazania odchylenia jest określona wzorem:

$$\mathbf{N = \frac{1}{e^2}}$$

Widzimy więc, co zgodne z intuicją, że im mniejsza wartość odchylenia (im strumień generowany przez szyfr jest bliższy strumieniowi idealnie losowych danych), tym większa wielkość próbki, potrzebna do statystycznie istotnego zdiagnozowania, czy odchylenie rzeczywiście występuje.

5.1 Distinguisher Alexandra Maximova dla szyfru VMPC

Na tegorocznej konferencji FSE'05 Alexander Maximov opisał atak typu distinguisher, który pozwala odróżnić strumień generowany przez szyfr VMPC od strumienia idealnie losowego po zaobserwowaniu 2^{54} bajtów (około 18 trylionów bajtów lub 18 milionów gigabajtów).

Niech X_t oznacza wartość t-tego kolejnego bajtu wygenerowanego przez szyfr VMPC (w kroku 4, Tabela 1).

Maximov oparł swój distinguisher na obserwacji, że zdarzenie polegające na wygenerowaniu dwóch kolejnych wartości zero (X_t =0 oraz X_{t+1} =0) przy określonych wartościach zmiennych wewnętrznych szyfru (n = 0 oraz s = 1) występuje zbyt rzadko w porównaniu z rozkładem prawdopodobieństwa oczekiwanym od idealnie losowego ciągu danych. Maximov oszacował, że dla szyfru VMPC odpowiadające tej obserwacji prawdopodobieństwo zajścia zdarzenia X_t =0 oraz X_{t+1} =0 pod warunkiem, że n = 0 oraz s = 1 wynosi:

$$\mathbf{P}(X_t = 0 \text{ oraz } X_{t+1} = 0 \mid n = 0 \text{ oraz } s = 1) \approx 1 / 256^3$$

podczas gdy dla ciągu danych idealnie losowych prawdopodobieństwo to powinno wynosić $\mathbf{1 / 256^2}$.

Sytuacja n=0 oraz s=1 zachodzi jednak rzadko, średnio co 65536 wygenerowanych bajtów.

Korzystając z tego wyniku Maximov oblicza prawdopodobieństwo zajścia X_t =0 oraz X_{t+1} =0 jedynie pod warunkiem n = 0 (bez warunku s = 1). Do tego celu zakłada on, że dla s ≠ 1:
$P(X_t = 0 \text{ oraz } X_{t+1} = 0 \mid n = 0 \text{ oraz } s \neq 1) = 1 / 256^2$ a więc tyle, ile dla strumienia idealnie losowego.

Uzyskujemy więc przybliżenie, że

$$\mathbf{P}(X_t = 0 \text{ oraz } X_{t+1} = 0 \mid n = 0) \approx \frac{1}{256} * \frac{1}{256^3} + \frac{255}{256} * \frac{1}{256^2} \approx \frac{1}{256^2} * 0{,}996109$$

Podczas gdy dla strumienia idealnie losowego $P(X_t = 0 \text{ oraz } X_{t+1} = 0 \mid n = 0) = 1/256^2$.

Stosując metodę symulacji Maximov oszacował to prawdopodobieństwo z większą dokładnością i uzyskał:

$$\mathbf{P}(X_t = 0 \text{ oraz } X_{t+1} = 0 \mid n = 0) \approx \frac{15938227062862998000}{256*4096374767995023500000} \approx 2^{-16,0057127}$$

Na podstawie tych danych Maximov oblicza odchylenie **e** (bias) od idealnego prawdopodobieństwa równego 2^{-16} na poziomie około 2^{-23}:

$$\mathbf{e_{VMPC} \approx 2^{-23}}$$

Tym samym wielkość próbki potrzebnej do zaobserwowania odchylenia wynosi $N_0=1/e_{VMPC}^2$ obserwacji. Ponieważ w szyfrze VMPC sytuacja n=0 występuje co 256 kroków, ostatecznie Maximov uzyskuje, że długością strumienia w bajtach, niezbędną do zaobserwowania zjawiska, jest $N = N_0 * 256$, a więc:

$$\mathbf{N=2^{54}}$$

Dysponując zatem szyfrogramem o długości 2^{54} bajtów oraz odpowiadającym mu tekstem jawnym tej samej długości (około 18 trylionów bajtów) możemy, korzystając z obserwacji Alexandra Maximova stwierdzić, że szyfrowanie miało miejsce najprawdopodobniej z wykorzystaniem szyfru strumieniowego VMPC.

Atak ten nie pozwala jednak ujawnić żadnych informacji o kluczu kryptograficznym lub o tekście jawnym (atak z założenia wymaga, aby tekst jawny był znany).

6. Schemat Tail-MAC

Schemat uwierzytelnionego szyfrowania Tail-MAC został zaproponowany podczas Rump Session konferencji FSE'04 oraz w [2] na konferencji Enigma'04. Algorytm ten pozwala na obliczanie kodów uwierzytelniających wiadomość (MAC, Message Authentication Code) równolegle z szyfrowaniem przy pomocy szyfru strumieniowego VMPC. Algorytm schematu Tail-MAC przedstawia się następująco:

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
n, m, g : zmienne robocze całkowitoliczbowe
$+$: 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 2)

19. Umieść 20 nowych wartości wygenerowanych przez szyfr VMPC (Tabela 1) 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*

Schemat ten charakteryzuje się prostotą konstrukcji, jednak okazuje się, że jest podatny na dwa ataki, z których jeden pozwala na uzyskanie realnych kolizji w wygenerowanych kodach, a drugi prowadzi do powstania silnych zależności liniowych pomiędzy wewnętrznymi zmiennymi algorytmu.

6.1. Kolizje w schemacie Tail-MAC z prawdopodobieństwem 2^{-32}

Uzyskanie dwóch różnych wiadomości generujących tę samą wartość kodu uwierzytelniające wiadomość jest dla schematu Tail-MAC możliwe z prawdopodobieństwem 2^{-32} poprzez dopisanie bajtu o wartości zero do oryginalnej wiadomości.

Rozważmy dwie wiadomości długości d: W_1 oraz W_2, gdzie $W_2 = W_1$ + „0”. Zauważmy, że ostatnia iteracja szyfrowania wiadomości W_2 wykona dokładnie te same operacje na zmiennych $x_1..x_4$ oraz na tablicy T, co pierwsza iteracja przetwarzania końcowego (zainicjowanego w kroku 17) dla wiadomości W_1 (we wszystkich 32 krokach fazy przetwarzania końcowego wartość `Szyfrogram[m]` jest ustawiona na zero). Następnie pierwsza iteracja przetwarzana końcowego dla wiadomości W_2 wykona dokładnie te same operacje, co druga faza przetwarzania końcowego dla wiadomości W_1. Proces ten będzie się powtarzał aż do zakończenia 32 iteracji przetwarzania końcowego dla wiadomości W_2. Podczas tych 32 iteracji zmienne schematu Tail-MAC będą miały te same wartości zarówno dla wiadomości W_1, jak i W_2. Dopiero ostatnia iteracja przetwarzania końcowego dla wiadomości W_2 może zmienić tablicę T wygenerowaną dla W_2 w porównaniu z tą wygenerowaną dla W_1 (W_2 jest o jeden bajt dłuższa). Jednocześnie ta ostatnia iteracja staje się w analizowanym przypadku jedynym czynnikiem różnicującym kody uwierzytelniające wiadomość dla W_1 oraz W_2. Ponieważ w każdej iteracji cztery bajty tablicy T są aktualizowane (operacja XOR) wartościami zmiennych $x_1...x_4$, wystarczy, aby zaszła sytuacja $x_1 = x_2 = x_3 = x_4 = 0$, aby iteracja ta nie zmieniła wartości tablicy T ($T[a]$ XOR 0 = $T[a]$). Ponieważ zmienne $x_1..x_4$ są 8-bitowe oraz zakładając, że przyjmują one wartości wg rozkładu jednostajnego (prawdopodobieństwo wystąpienia każdej z 256 możliwych wartości jest takie samo), otrzymujemy automatycznie, że

$$\mathbf{P(x_1 = x_2 = x_3 = x_4 = 0) = (1/256)^4 = 2^{-32}}$$

Otrzymanie tych samych wartości tablicy T po wykonaniu kroku 17 dla wiadomości W_1 oraz W_2 jest równoznaczne z tym, że dla obydwu wiadomości wygenerowana zostanie w kolejnych krokach 18-20 ta sama wartość kodu uwierzytelniającego wiadomość MAC. Tym samym stosując opisaną procedurę możemy uzyskać kolizję w schemacie Tail-MAC z prawdopodobieństwem 2^{-32}.

Zasadniczym czynnikiem, który pozwala na przeprowadzenie ataku jest kompatybilność fazy szyfrowania (kroki 3-16) oraz fazy przetwarzania końcowego zainicjowanej w kroku 17. Kompatybilność oznacza tutaj, że dla tych samych wartości danych wejściowych (zmiennych wewnętrznych algorytmu oraz wiadomości) obie fazy wygenerują te same dane wyjściowe (te same wartości zmiennych $x_1..x_4$ i zmodyfikują tablicę T w ten sam sposób).

Operacją, która pozwala zneutralizować atak i zniwelować kompatybilność tych faz, jest wprowadzenie zmiennej ***R*** w schemacie VMPC-MAC. Pozwala ona uzyskać nie tylko niekompatybilność fazy szyfrowania z fazą przetwarzania końcowego, ale dodatkowo wzajemną niekompatybilność wszystkich 32 iteracji fazy przetwarzania końcowego. W rezultacie te same dane, ale przetwarzane przez algorytm w różnym czasie, będą w schemacie VMPC-MAC generowały inne dane w każdej iteracji przetwarzania końcowego (inne wartości $x_1..x_4$ i automatycznie inne modyfikacje tablicy T). Pozwoli to zneutralizować opisany atak na algorytm Tail-MAC.

6.2. Wymuszenie silnych zależności liniowych między zmiennymi $x_1..x_4$

Atak ten bazuje na liniowości przekształceń zmiennych $x_1..x_4$ w krokach 5-7. Rozważmy dwie wiadomości, dla których zachodzi $x_{1\_W2}^{(t)} = (x_{1\_W1}^{(t)} + 128)$ modulo 256 oraz $x_{1\_W2}^{(t+8)} = (x_{1\_W1}^{(t+8)} + 128)$ modulo 256 oraz $W_2[x]=W_1[x]$ dla $x \neq t$ oraz $x \neq (t+8)$, gdzie $W_n[t]$ oznacza t-ty bajt n-tej wiadomości, a $x_{1\_Wn}^{(t)}$ oznacza wartość zmiennej x_1 dla wiadomości Wn w t-tej iteracji. Zauważmy, że wartość zmiennej x_1 schematu Tail-MAC po wykonaniu kroku t+8 będzie miała tę samą wartość dla wiadomości W_1 jak i W_2. Stanie się tak, ponieważ (128+128) modulo 256=0 i w wyniku zachodzi $x_{1\_W2} = (x_{1\_W1} + 128 + 128)$ modulo 256. Co więcej wartości zmiennej x_{2_W2} oraz x_{2_W1} po wykonaniu kroku t+9 także będą równe, z tego samego powodu. Zależność liniowa jest zatem przenoszona ze zmiennej x_1 na kolejne zmienne schematu – x_2,x_3 oraz x_4.

Poprzez tę właściwość powstają liczne wyraźne zależności liniowe pomiędzy zmiennymi ($x_{n\_W2} = x_{n\_W1}$ lub $x_{n\_W2} = (x_{n\_W1} + 128)$ modulo 256) z które znacznie ograniczają efekt dyfuzji zmian w wiadomości na zmiany wewnętrznych zmiennych schematu ($x_1..x_4$) i pośrednio tablicy T.

Rozwiązaniem tego problemu może być zastąpienie operacji liniowych (dodawanie modulo 256) wykowywanych na zmiennych $x_1..x_4$ w krokach 5-7 operacjami zrandomizowanymi – np. korzystającymi z faktu, że struktura permutacji P jest nieodróżnialna od struktury permutacji losowej. Tym samym, jeśli po wykonaniu operacji dodawania przepuścimy wynik dodatkowo przez permutację P, wówczas liniowe zależności między zmiennymi $x_1..x_4$ zostaną całkowicie zakłócone.

Omówiony w rozdziale 7 schemat VMPC-MAC w celu zniwelowania opisanego zjawiska korzysta właśnie z dodatkowego odwołania się do permutacji P przy aktualizowaniu zmiennych $x_1..x_4$.

7. Schemat VMPC-MAC

Zmienne: Jak dla schematu Tail-MAC (rozdział 6) oraz:

R : 8-bitowa zmienna

0. Zainicjuj permutację *P* algorytmem VMPC-KSA (rozdział 4).
1. (x_1,x_2,x_3,x_4,*n*,*m*,*g*)=0; *T*[*x*]=0 dla *x*=0,1,...,31

2. Powtórz kroki (2.1 - 2.14) *Długość_Wiadomości* razy:
   2.1. *s* = *P*[*s* + *P*[*n*]]
   2.2. *Szyfrogram*[*m*] = *Wiadomość*[*m*] xor *P*[*P*[*P*[*s*]]+1]

   2.3. x_4 = ***P***[x_4 + x_3]
   2.4. x_3 = ***P***[x_3 + x_2]
   2.5. x_2 = ***P***[x_2 + x_1]
   2.6. x_1 = ***P***[x_1 + *s* + *Szyfrogram*[*m*]]

   2.7. *T*[*g*] = *T*[*g*] xor x_1
   2.8. *T*[*g*+1] = *T*[*g*+1] xor x_2
   2.9. *T*[*g*+2] = *T*[*g*+2] xor x_3
   2.10. *T*[*g*+3] = *T*[*g*+3] xor x_4

   2.11. *x* = *P*[*n*]; *P*[*n*] = *P*[*s*]; *P*[*s*] = *x*
   2.12. *g* = (*g* + 4) modulo 32
   2.13. *n* = *n* + 1
   2.14. Zwiększ wartość *m* o 1

3. ***R* = 1**
4. Powtórz kroki (4.1 - 4.14) 24 razy:
   4.1. *s* = *P*[*s* + *P*[*n*]]

   4.2. x_4 = ***P***[x_4 + x_3 + ***R***]
   4.3. x_3 = ***P***[x_3 + x_2 + ***R***]
   4.4. x_2 = ***P***[x_2 + x_1 + ***R***]
   4.5. x_1 = ***P***[x_1 + *s* + ***R***]

   4.6. *T*[*g*] = *T*[*g*] xor x_1
   4.7. *T*[*g*+1] = *T*[*g*+1] xor x_2
   4.8. *T*[*g*+2] = *T*[*g*+2] xor x_3
   4.9. *T*[*g*+3] = *T*[*g*+3] xor x_4

   4.10. *x* = *P*[*n*]; *P*[*n*] = *P*[*s*]; *P*[*s*] = *x*
   4.11. *g* = (*g* + 4) modulo 32
   4.12. *n* = *n* + 1
   4.13. ***R* = *R* + 1**
   4.14. Zwiększ wartość *m* o 1

5. Umieść *T* w *K*; ustaw *c* = 32. Wykonaj krok 3 VMPC-KSA (Tabela 2)

6. Umieść 20 nowych wartości wygenerowanych przez szyfr VMPC (Tabela 1)
   w tablicy *M* (Dla *n* od 0 do 19: wykonaj kroki 6.1 - 6.3:
   6.1. *s* = *P*[*s* + *P*[*n*]]
   6.2. *M*[*n*] = *P*[*P*[*P*[*s*]]+1]
   6.3. *x* = *P*[*n*]; *P*[*n*] = *P*[*s*]; *P*[*s*] = *x*)

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

8. Analizy statystyczne schematu VMPC-MAC

Schemat był testowany na występowanie efektu dyfuzji zmian danych wejściowych na zmiany danych wyjściowych. Dane wejściowe stanowią: szyfrowana `Wiadomość` (oznaczona dalej jako W) oraz klucz K i wektor inicjujący V, użyte w kroku 0 do zainicjowania permutacji P. Faktycznymi danymi wyjściowymi jest 20 bajtów wygenerowanych przez szyfr VMPC w kroku 6, jednakże aby uzyskać bardziej rygorystyczną miarę efektu dyfuzji, za dane wyjściowe dla celów testów statystycznych będziemy uważać tablicę T uzyskaną tylko w krokach od 1 do 4. Dalsze kroki (5 i 6) zapewniają dodatkowe zintensyfikowanie efektu dyfuzji schematu dzięki efektowi dyfuzji algorytmu VMPC-KSA, połączonym z szyfrem VMPC, szerzej przeanalizowanemu w [1] oraz [2]. Takie podejście zapewnia dodatkowy margines bezpieczeństwa dla efektu dyfuzji uzyskanego w testach.

Prawidłowy efekt dyfuzji (nieodróżnialna od losowej relacja między danymi wyjściowymi uzyskanymi z różnych danych wejściowych) występuje wtedy, gdy dla T_0, będącej 32-bajtową tablicą uzyskaną z generatora liczb prawdziwie losowych, $T_1=MAC(W_1 , K_1 , V_1)$ oraz $T_2=MAC(W_2 , K_2 , V_2)$, gdzie spełniony jest przynajmniej jeden z warunków: $W_1 \neq W_2$; $K_1 \neq K_2$; $V_1 \neq V_2$ (różnice mogą być ograniczone do pojedynczego bitu), prawdopodobieństwo rozstrzygnięcia, która z wartości (T_0 , T_1 czy T_2) była wygenerowana ze źródła prawdziwie losowego wynosi 1/3.

Dyfuzja zmian klucza K oraz wektora inicjującego V na zmiany w tablicy T może być przyjęta za nieodróżnialną od losowej jako bezpośrednia konsekwencja efektu dyfuzji algorytmu VMPC-KSA (opisanego dokładniej w [1] oraz w [2]).

Zasadniczym pytaniem jest, czy efekt dyfuzji występuje, gdy $K_1 = K_2$; $V_1 = V_2$ oraz $W_1 \neq W_2$. Aby to określić, przeprowadzono dwa rodzaje testów statystycznych: dla W_1 i W_2 różniących się tylko ostatnim bajtem oraz dla W_1 i W_2, gdzie W_2 jest uzyskane poprzez skopiowanie wartości W_1 do W_2 oraz przez dopisanie jednego bajtu do W_2.

8.1. W_1 i W_2 różniące się ostatnim bajtem

W teście tym W_1 oraz W_2 miały długość 8 bajtów, a ostatni bajt W_2 (ten, który jest odczytywany w najpóźniejszym czasie) przyjmował wszystkie 255 możliwych wartości (oprócz wartości ostatniego bajtu W_1). Dla każdej wartości W_2 dwie tablice $T_1=MAC(W_1 , K_0 , V_0)$ oraz $T_2=MAC(W_2 , K_0 , V_0)$ były porównywane. Wartości W_1 , K_0 i V_0 były zmieniane po każdej porcji 255 porównań. Test wykonano dla $2^{37.2}$ par W_1 i W_2. Częstotliwości występowania sytuacji, gdzie $T_1[x] = T_2[x]$ dla $x \in \{0..31\}$, nie wykazały statystycznie istotnego odchylenia od wartości oczekiwanej 1/256.

8.2. W_2 z dopisanym ostatnim bajtem

W teście tym W_1 miała długość 8 bajtów, natomiast W_2 miała długość 9 bajtów. Dla $x \in \{1..8\}$: $W_1[x] = W_2[x]$, a ostatni bajt W_2 ($W_2[9]$) przyjmował wszystkie 256 możliwych wartości. Dla każdej wartości W_2 dwie tablice $T_1=MAC(W_1 , K_0 , V_0)$ oraz $T_2=MAC(W_2 , K_0 , V_0)$ były porównywane. Wartości W_1 , K_0 , V_0 były zmieniane po każdej porcji 256 porównań. Test wykonano dla $2^{37.2}$ par W_1 i W_2. Częstotliwości występowania sytuacji, gdzie $T_1[x] = T_2[x]$ dla $x \in \{0..31\}$, nie wykazały statystycznie istotnego odchylenia od wartości oczekiwanej 1/256.

9. Propozycja algorytmu VMPC-HASH

W kryptografii szeroko stosowaną rodziną algorytmów są funkcje skrótu, nazywane także funkcjami hashującymi (hash functions). Ich podstawowym zastosowaniem jest tworzenie podpisów cyfrowych. Dowolnej długości strumień danych po przepuszczeniu przez funkcję skrótu generuje stałej długości „odcisk palca" podanego strumienia.

Głównymi cechami bezpieczeństwa wymaganymi od funkcji skrótu są:

- Jednokierunkowość. Mając dane HASH(X), znalezienie X powinno być obliczeniowo niewykonalne.
- Bezkolizyjność. Nie powinno być możliwe znalezienie dwóch wiadomości X_1 i X_2, takich że $X_1 \neq X_2$, ale HASH(X_1) = HASH(X_2).

Uważana za bezpieczną i powszechnie stosowana funkcja skrótu SHA-1 została w tym roku ogłoszona jako złamana przez zespół kryptologów z Shandong University w Chinach. Znaleźli oni sposób na uzyskanie kolizji w SHA-1 po wykonaniu 2^{69} operacji. Oczywiście jest to wciąż bardzo wysoka złożoność, ale może budzić pewne obawy o bezpieczeństwo SHA-1 w przyszłości.

Opisany w rozdziale 7 schemat VMPC-MAC służący do generowania zależnych od klucza kodów uwierzytelniających wiadomość (MAC) jest tak skonstruowany, ze można zastosować go, w minimalnie zmodyfikowanej formie, jako właśnie funkcja skrótu.

9.1. Algorytm VMPC-HASH

Jedyną modyfikacją schematu VMPC-MAC, jaka jest potrzebna, aby algorytm ten mógł pełnić rolę funkcji skrótu, jest ustalenie sztywnej wartości klucza (i wektora inicjującego), który zostanie wykorzystany do zainicjowania permutacji P oraz zmiennej s. Proponowane są następujące wartości:

K_h	0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120
V_h	128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248

Tabela 3. Wartości stałe klucza K_h i wektora inicjującego V_h dla schematu VMPC-HASH

Pierwszym krokiem algorytmu VMPC-HASH jest zainicjowanie permutacji P oraz zmiennej s kluczem K_h oraz wektorem inicjującym V_h – przy pomocy algorytmu VMPC-KSA, opisanego w rozdziale 4. Jest to równoważne z wykonaniem kroku 0 algorytmu VMPC-MAC (rozdział 7), dla podanych w Tabeli 3 wartości klucza K_h i wektora V_h.

Hashowana wiadomość będzie się znajdowała w tablicy `Wiadomość` algorytmu VMPC-MAC, natomiast tablica `Szyfrogram[m]` (w kroku 2.2 schematu VMPC-MAC) dla ułatwienia implementacji może zostać zastąpiona pojedynczą 8-bitową zmienną, ponieważ wartość szyfrogramu nie musi być nigdzie zapamiętana (celem jest uzyskanie jedynie wartości funkcji skrótu).

Pozostałe kroki algorytmu VMPC-HASH będą identyczne z krokami schematu VMPC-MAC. W ostatnim 7 kroku z oczywistych względów wyeliminowane zostanie dopisanie wartości MAC do szyfrogramu.

Algorytm VMPC-HASH:

Zmienne: Jak dla schematu VMPC-MAC (rozdział 7) oraz:

H : 20-bajtowa tablica

0. Zainicjuj permutację P algorytmem VMPC-KSA (rozdział 4)
   dla: klucza $K=K_h$ oraz wektora inicjującego $V = V_h$ *(patrz Tabela 3)*

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

2. Powtórz kroki (2.1 - 2.13) *Długość_Wiadomości* razy:
   2.1. $s = P[\ s + P[n]\ ]$

   2.2. $x_4 = P[\ x_4 + x_3\ ]$
   2.3. $x_3 = P[\ x_3 + x_2\ ]$
   2.4. $x_2 = P[\ x_2 + x_1\ ]$
   2.5. $x_1 = P[\ x_1 + s + (Wiadomość[m]$ xor $P[P[P[s]]+1])]$

   2.6. $T[g] = T[g]$ xor x_1
   2.7. $T[g+1] = T[g+1]$ xor x_2
   2.8. $T[g+2] = T[g+2]$ xor x_3
   2.9. $T[g+3] = T[g+3]$ xor x_4

   2.10. $x = P[n]$; $P[n] = P[s]$; $P[s] = x$
   2.11. $g = (g + 4)$ modulo 32
   2.12. $n = n + 1$
   2.13. Zwiększ wartość m o 1

3. $R = 1$
4. Powtórz kroki (4.1 - 4.14) 24 razy:
   4.1. $s = P[\ s + P[n]\ ]$

   4.2. $x_4 = P[\ x_4 + x_3 + R\ ]$
   4.3. $x_3 = P[\ x_3 + x_2 + R\ ]$
   4.4. $x_2 = P[\ x_2 + x_1 + R\ ]$
   4.5. $x_1 = P[\ x_1 + s + R\ ]$

   4.6. $T[g] = T[g]$ xor x_1
   4.7. $T[g+1] = T[g+1]$ xor x_2
   4.8. $T[g+2] = T[g+2]$ xor x_3
   4.9. $T[g+3] = T[g+3]$ xor x_4

   4.10. $x = P[n]$; $P[n] = P[s]$; $P[s] = x$
   4.11. $g = (g + 4)$ modulo 32
   4.12. $n = n + 1$
   4.13. $R = R + 1$
   4.14. Zwiększ wartość m o 1

5. Umieść T w K; ustaw $c = 32$. Wykonaj krok 3 VMPC-KSA (Tabela 2)

6. Umieść 20 nowych wartości wygenerowanych przez szyfr VMPC (Tabela 1)
   w tablicy H (Dla n od 0 do 19: wykonaj kroki 6.1 - 6.3:
   6.1. $s = P[\ s + P[n]\ ]$
   6.2. $H[n] = P[P[P[s]]+1]$
   6.3. $x = P[n]$; $P[n] = P[s]$; $P[s] = x$)

Wygenerowana w kroku 6 dwudziestobajtowa tablica H stanowi wartość funkcji skrótu dla wiadomości *Wiadomość*.

9.2. Zarys analizy bezpieczeństwa algorytmu VMPC-HASH

Bezpieczeństwo algorytmu VMPC-HASH w zasadniczym stopniu korzysta z analiz bezpieczeństwa schematu VMPC-MAC. Przeprowadzone testy na efekt dyfuzji, omówione w rozdziale 8, odnoszą się automatycznie do algorytmu VMPC-HASH. Ponieważ wyniki wskazują na występowanie prawidłowego efektu dyfuzji dla schematu VMPC-MAC, można podejrzewać, że znalezienie kolizji dla algorytmu VMPC-HASH powinno być w praktyce niewykonalne. Podobnie jak dla algorytmu VMPC-MAC, także dla VMPC-HASH najlepszym znanym atakiem uzyskującym kolizję – innym niż atak typu brute force z wykorzystaniem tzw. paradoksu urodzinowego (birthday paradox) – jest atak o złożoności 2^{144}, opisany w [4].

Wobec analiz algorytmu VMPC-KSA, opisanych w [1] oraz w [2], sam krok 5 algorytmu VMPC-HASH powinien zapewniać praktyczną jednokierunkowość przekształcania tablicy `Wiadomość` w tablicę *H*. Dodatkowe przemieszanie uzyskanej w kroku 5 permutacji *P* szyfrem VMPC w kroku 6, a także faza przetwarzania końcowego (krok 4), mieszająca tablicę *T*, wydają się zapewniać dodatkowe na tyle złożone przekształcenia danych wejściowych w wyjściowe, że próba ich odwrócenia wydaje się w praktyce niemożliwa do wykonania.

Tym samym można uznać, że zastosowanie stałego klucza w schemacie VMPC-MAC pozwala w łatwy sposób uzyskać algorytm funkcji skrótu VMPC-HASH, który uzasadnienie swoich podstawowych cech bezpieczeństwa czerpie z już przeprowadzonych analiz dla algorytmów VMPC-MAC oraz VMPC-KSA.

Cechą, która może pozytywnie wyróżniać algorytm VMPC-HASH na tle klasycznego podejścia blokowego do konstrukcji funkcji skrótu jest fakt, że VMPC-HASH możemy używać do każdej długości podpisywanych wiadomości, bez konieczności stosowania dodatkowych algorytmów wyrównujących długość danych wejściowych do wielokrotności wielkości bloku, VMPC-HASH działa bowiem na zasadzie strumieniowej.

10. Zagrożenia wynikające ze stosowania krótkich i/lub regularnych haseł

W tym rozdziale zwróćmy uwagę na obserwację, która często w praktyce jest niedoceniana. Użytkownicy bardzo często stosują do zabezpieczenia danych (jak np. szyfrowanie czy logowanie) krótkich, łatwych do zapamiętania haseł lub haseł o bardzo regularnej strukturze. Celem są zwykle względy praktyczne, zwłaszcza konieczność zapamiętania hasła.

Standardową dziś długością symetrycznego klucza kryptograficznego jest 128 bitów. Zwróćmy uwagę, że gdybyśmy chcieli uzyskać taki poziom bezpieczeństwa od wpisanego z klawiatury hasła, musiałoby ono mieć – przy założeniu, że składa się z dużych i małych liter oraz cyfr – **aż 22 znaki!** [$(26+26+10)^{22} \approx 2^{128}$].

Co więcej, hasło takie nie powinno zawierać ułatwiających zapamiętanie regularności, gdyż atakujący mógłby rozpatrzyć hasła regularne na samym początku i – dla przykładu – mógłby mieć uzasadnione podejrzenia, aby hasło „abcabcabcabcabcabcabc” sprawdzić przed hasłem np. „bexmdndrdavthfvevmxnb”.

W celu namacalnego zilustrowania problemu posłużyliśmy się symulacjami wykonanymi przy pomocy programu do szyfrowania plików/folderów i poczty elektronicznej **VMPC Data Security** (dostępnemu na stronie internetowej www.VMPCfunction.com) oraz przy pomocy dostępnego za darmo na tej samej stronie programu do łamania krótkich haseł. Program do łamania haseł stosuje uniwersalną metodę ataku brutalnej siły (brute force), skuteczną dla każdego szyfru. Spójrzmy, ile czasu na domowym komputerze 3,4 GHz zajęłoby złamanie tą metodą haseł o długości od 3 do 7 znaków, złożonych tylko z małych (lub tylko z dużych) liter:

Długość hasła w znakach	Średni czas złamania
3	0,1 sekundy
4	3,1 sekundy
5	1 minuta 18 sekund
6	34 minuty
7	14 godzin 45 minut

Tabela 4. Czasy łamania haseł metodą brute force
***** *Na przykładzie szyfrogramów stworzonych za pomocą aplikacji VMPC Data Security.*

Z symulacji wynika, że hasło 7-znakowe złożone tylko z małych liter (lub tylko dużych), które intuicyjnie można by uznać za stosunkowo długie, można złamać metodą brutalnej siły średnio w ciągu kilkunastu godzin pracy jednego domowego komputera.

Wyjściem, które wydaje się rozwiązać ten problem z kryptograficznego punktu widzenia, jest zastosowanie haseł o odpowiedniej długości i, co więcej, wygenerowanych nie bezpośrednio przez człowieka, który z natury szuka regularności, ale z generatora liczb losowych.

Przykładowym realnym rozwiązaniem jest zastosowany w aplikacji VMPC Data Security moduł generowania kluczy, który pozwala transformować chwilowe parametry kursora myszki (położenie oraz odstępy czasu między zmianami pozycji kursora mierzone do jednej tysięcznej części sekundy) na ciąg danych w praktyce nieodróżnialny od ciągu losowego. Przyjrzyjmy się przykładowemu 128-bitowemu hasłu wygenerowanemu w ten sposób. Hasło złożone z małych, dużych liter oraz cyfr. Długość hasła to 23 znaki zamiast 22, ponieważ dla uniknięcia nieporozumień aplikacja nie stosuje liter *I i; L l; O o* w generowanych hasłach:

Przykładowe bezpieczne hasło 128-bitowe: **d8PmZMWy140zVabRBqTHQNX**

Zapamiętanie takiego hasła z pewnością nie jest łatwe. Rozwiązaniem może tu być zapis hasła na nośniku wymiennym, najlepiej w wielu kopiach na jednym nośniku (funkcja taka jest np. dostępna w przywoływanej aplikacji VMPC Data Security). Wówczas jednak istnieje ryzyko np. kradzieży lub zgubienia nośnika z hasłem, co jest jednoznaczne z utratą zaszyfrowanych danych i/lub dostania się ich w niepowołane ręce.

Wydaje się więc, że hasła ogólnie rzecz biorąc nie są idealnym rozwiązaniem zapewniającym szeroko rozumiane bezpieczeństwo. Hasła krótkie są bowiem łatwe do złamania, natomiast te odpowiednio długie i nieregularne – z istoty rzeczy – trudne do zapamiętania. Można zatem uznać, że długość hasła powinno się świadomie dobierać do stopnia poufności zabezpieczanych nim danych. Pewne jest, że zastosowanie hasła kilkuznakowego rodzi realne ryzyko jego złamania bez angażowania środków większych niż domowy komputer, co potwierdziły przeprowadzone symulacje.

Bibliografia

[1] Bartosz Żółtak „VMPC One-Way Function and Stream Cipher”, FSE 2004, Springer-Verlag LNCS 3017

[2] Bartosz Żółtak “Funkcja jednokierunkowa VMPC I system uwierzytelnionego szyfrowania VMPC-Tail-MAC”, KKZK Enigma 2004

[3] Alexander Maximov „Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers”, FSE 2005, http://www.it.lth.se/movax/Publications/2005/vmpc.pdf

[4] Bartosz Zoltak, „VMPC-MAC: A Stream Cipher Based Authenticated Encryption Scheme”, IACR ePrint Archive http://eprint.iacr.org/2004/301; www.VMPCfunction.com

[5] Bartosz Zoltak, „Security of Symmetric Encryption Schemes with One-Way IND-CNA Key Setup”, IACR ePrint Archive http://eprint.iacr.org/2004/120; www.VMPCfunction.com

[6] www.VMPCfunction.com