Szyfrowanie i uwierzytelnianie z użyciem VMPC

Bartosz Żółtak

Bartosz Żółtak jest autorem funkcji jednokierunkowej oraz szyfru strumieniowego VMPC, zaprezentowanych m.in. na międzynarodowej konferencji kryptograficznej FSE 2004 w Indiach. Na algorytmach VMPC bazuje także stworzona przez autora aplikacja do szyfrowania danych VMPC Data Security. Więcej informacji o autorze i VMPC można znaleźć pod adresem *www.VMPCfunction.com*

We wrześniowym numerze Software 2.0 z roku 2004 pisaliśmy o funkcji jednokierunkowej i szyfrze strumieniowym VMPC. Są to algorytmy kryptograficzne opracowane przez Bartosza Żółtaka i zaprezentowane m.in. na konferencji *Fast Software Encryption* 2004 (FSE'04, *www.isical.ac.in/~fse2004*), w Delhi w Indiach w dniach 5-7 lutego 2004. Główną cechą VMPC jako techniki szyfrowania danych jest prostota konstrukcji oraz łatwość implementacji.

Przypomnijmy – funkcja VMPC to przekształcenie permutacji (tablicy liczb całkowitych, w której elementy się nie powtarzają i występują wszystkie kolejne liczby z danego zakresu – np. od 0 do 255) utworzone wg wzoru $Q[x]=P[P[P[x]]+1]$, gdzie + jest dodawaniem modulo wielkość permutacji – u nas zwykle 256. Okazuje się, że takie złożenie permutacji P jest bardzo trudne do odwrócenia – znalezienie permutacji P na podstawie Q wymaga średnio około 2^{260} operacji. Jednocześnie zaimplementowanie funkcji VMPC w assemblerze wymaga jedynie trzech jednocyklowych rozkazów MOV na bajt.

Na bazie tych własności powstał prosty w konstrukcji i wydajny szyfr strumieniowy VMPC, opisany dokładniej w Software 2.0 nr 9/2004. Jego opis można także znaleźć na stronie poświęconej VMPC - *www.VMPCfunction.com/cipher.htm*.

Aby zaszyfrować ciąg bajtów, potrzebujemy klucza, wektora inicjującego, algorytmu inicjowania klucza (tzw. KSA, Key Scheduling Algorithm) oraz algorytmu szyfrowania. Klucz i wektor to zwykle 16-bajtowe tablice. Klucz musi być tajny, a wektor unikalny dla każdej wiadomości szyfrowanej tym samym kluczem (wektor tajny być nie musi). Algorytm KSA przekształca klucz oraz wektor w 256-elementową permutację P. Permutacja ta jest następnie wykorzystana przez szyfr VMPC do wygenerowania strumienia bajtów, który poddawany jest operacji XOR z kolejnymi bajtami szyfrowanej wiadomości. Tak powstaje szyfrogram. Deszyfrowanie odbywa się wg dokładnie tego samego algorytmu, ponieważ A xor B xor B = A, z tą tylko różnicą, że na wejściu używamy szyfrogramu i jego XORujemy ze strumieniem VMPC, odtwarzając w ten sposób postać oryginalnej wiadomości.

Przy założeniu, że klucz udaje nam się utrzymać w tajemnicy oraz że szyfru nie da się złamać – można by sądzić, że dane po zaszyfrowaniu są już całkowicie bezpieczne. Wyobraźmy sobie jednak sytuację, gdzie np. otrzymujemy zaszyfrowaną wiadomość z numerem konta, na jakie mamy wykonać przelew. Po zdeszyfrowaniu widzimy ciąg cyfr, który wygląda sensownie (a więc najprawdopodobniej użyliśmy właściwego klucza i właściwego wektora). Skąd możemy jednak mieć pewność, że np. jeden bajt numeru konta nie został przekłamany...? Np. w wyniku błędu transmisji lub przez celowe działanie wroga? Nawet jeśli wróg nie jest w stanie kontrolować zmiany w szyfrogramie – to i tak dokonując takiej zmiany – może mieć pewność, że utrudni nam życie. Skąd możemy mieć zatem pewność, że wiadomość po zdeszyfrowaniu jest rzeczywiście tą samą wiadomością, którą zaszyfrował dla nas nadawca?

Z pomocą przychodzą tak zwane kody uwierzytelniające wiadomość (MAC, Message Authentication Code). Szyfru VMPC także możemy użyć w trybie rozszerzonym o algorytm obliczania kodów MAC.

MAC to nic innego jak rodzaj sumy kontrolnej szyfrowanych danych. Jednakże istotną cechą odróżniającą MAC od tradycyjnych sum kontrolnych, jak CRC czy funkcje skrótu (np. SHA-1), jest fakt, iż MAC jest sumą kontrolną zależną od klucza i wektora inicjującego. Ta sama wiadomość zaszyfrowania różnymi kluczami lub ta sama wiadomość zaszyfrowana tym samym kluczem, ale z różnymi wektorami inicjującymi – powinna zawsze dać inny wynik sumy kontrolnej MAC.

Główną cechą bezpieczeństwa kodów MAC jest ich bezkolizyjność. Znalezienie dwóch różnych wiadomości, ale generujących tę samą wartość kodu MAC, powinno być obliczalnie niewykonalne. Cechą statystyczną algorytmu MAC, ściśle powiązaną z problemem występowania kolizji, jest efekt dyfuzji (patrz ramka). Gdy algorytm MAC zapewnia prawidłowy efekt dyfuzji – wówczas najprawdopodobniej znalezienie kolizji jest dla niego praktycznie niewykonalne.

Efekt dyfuzji oznacza, że dla dwóch wiadomości różniących się w dowolnie małym stopniu (np. tylko ostatnim bajtem) wygenerowane kody MAC są całkowicie inne. „Całkowicie” oznacza tutaj, że relacje między dwoma tak wygenerowanymi kodami MAC są nieodróżnialne od relacji, jakich oczekiwalibyśmy od dwóch losowych ciągów bajtów.

Procedura korzystania z MACów polega na tym, że równolegle z szyfrowaniem obliczana jest zależna od klucza i wektora inicjującego suma kontrolna szyfrowanej wiadomości. Następnie suma ta dopisywana jest na końcu wiadomości. Odbiorca wiadomości deszyfruje ją, także obliczając równolegle jej sumę kontrolną. Po zakończeniu deszyfrowania porównuje obliczoną przez siebie wartość z wartością dołączoną do szyfrogramu. Jeśli wartości są równe – oznacza to, że najprawdopodobniej wiadomość nie została po drodze zmieniona i możemy mieć praktyczną pewność, że czytamy dokładnie tę samą wiadomość, jaką nadawca chciał nam przekazać. Jeśli natomiast wartości są różne – może to oznaczać, że wiadomość została zmieniona podczas transmisji lub że używamy niewłaściwego klucza.

Kody MAC są bardzo ważnym elementem systemów kryptograficznych i w zasadzie wszędzie, gdzie tylko stosujemy szyfrowanie – powinniśmy używać także kodów MAC, ponieważ tylko wtedy mamy pewność, że wiadomość po zdeszyfrowaniu jest tą wiadomością, jaką chciał nam przekazać nadawca.

Przyjrzyjmy się teraz, jak możemy szyfrować algorytmem VMPC z rozszerzeniem o kody uwierzytelniające wiadomość – schemat VMPC-MAC (listing 1).

Listing 1. Schemat VMPC-MAC

L : Długość szyfrowanej wiadomości w bajtach
P : 256-elementowa tablica zawierająca permutację
s,*R* : 8-bitowe zmienne
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
+ oznacza dodawanie modulo 256

```
0. Zainicjuj permutację P algorytmem VMPC-KSA3 (Listing 2)
1. (x1,x2,x3,x4,n,m,g)=0; T[x]=0 dla x=0,1,...,31

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

    2.3.  x4 = P[ x4 + x3 ]
    2.4.  x3 = P[ x3 + x2 ]
    2.5.  x2 = P[ x2 + x1 ]
    2.6.  x1 = P[ x1 + s + Szyfrogram[m] ]

    2.7. T[g]   = T[g]   xor x1
    2.8. T[g+1] = T[g+1] xor x2
    2.9. T[g+2] = T[g+2] xor x3
   2.10. T[g+3] = T[g+3] xor x4

   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.  x4 = P[ x4 + x3 + R ]
    4.3.  x3 = P[ x3 + x2 + R ]
    4.4.  x2 = P[ x2 + x1 + R ]
    4.5.  x1 = P[ x1 + s + R ]

    4.6. T[g]   = T[g]   xor x1
    4.7. T[g+1] = T[g+1] xor x2
    4.8. T[g+2] = T[g+2] xor x3
    4.9. T[g+3] = T[g+3] xor x4

   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 5 VMPC-KSA3 (Listing 2)

6. Umieść 20 nowych wartości wygenerowanych przez szyfr VMPC
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
```

Kroki 2.1, 2.2 i 2.11 to główne operacje szyfru VMPC. Natomiast kroki od 2.3 do 2.10 to zasadnicze operacje rozszerzenia VMPC o algorytm obliczania kodów MAC. Zmienne `x1..x4` oraz tablica `T` stanowią ślad szyfrowanej wiadomości. Dzięki temu, że przy ich aktualizowaniu używana jest także permutacja `P`, są one zależne także od klucza oraz wektora inicjującego (te bowiem determinują postać permutacji `P` w kroku 0 algorytmu). Krok 4 pełni rolę finalizującą – pozwalającą przenieść ewentualne zmiany ostatnich bajtów wiadomości na całą tablicę `T`.

Wg przeprowadzonych testów statystycznych, algorytm zapewnia prawidłowy efekt dyfuzji, przez co nie powinniśmy się martwić o ryzyko wystąpienia w nim kolizji. Z testów wynika, że nawet przy różnicy na poziomie jednego bitu w którejkolwiek z trzech danych wejściowych (wiadomość, klucz, wektor inicjujący), już powstałe po kroku 4 tablice `T` są od siebie „całkowicie" różne (relacje nieodróżnialne od losowych), a zastosowanie dodatkowych kroków 5 i 6 dodatkowo pogłębia efekt dyfuzji, przez co posiada on margines bezpieczeństwa ponad wynikami testowymi.

Algorytm VMPC-MAC jest prosty w implementacji, działa szybko i pozwala rozwiązać problem kodów uwierzytelniających wiadomość tam, gdzie korzystamy z szyfrowania VMPC. Więcej o schemacie VMPC-MAC można przeczytać na stronie *www.VMPCfunction.com/vmpcmac.htm* (są tam dostępne także testowe wartości wejścia i wyjścia algorytmu) oraz w pracy o VMPC zaprezentowanej w czerwcu na tegorocznej edycji konferencji Enigma w Warszawie.

Szyfrowanie z ubezpieczeniem

Jedną z najostrzejszych kategorii złamania szyfru jest możliwość znalezienia klucza na podstawie analizy szyfrogramów i odpowiadających im tekstów jawnych. Szyfr, dla którego da się wskazać efektywny algorytm odnalezienia na podstawie tych danych klucza (wszystko jedno czy klucza użytkownika, przed użyciem go w algorytmie inicjowania klucza KSA, czy też klucza wewnętrznego szyfru po użyciu KSA) jest uważany za kategorycznie złamany. Najczęściej stosowaną miarą, zwłaszcza dla szyfrów strumieniowych, o jakich tu mówimy, jest złożoność odnalezienia właśnie klucza wewnętrznego. W przypadku VMPC byłaby to permutacja `P` po wykonaniu algorytmu VMPC-KSA.

Wg dotychczasowych analiz złożoność takiego zadania szacowana jest na około 2^{900} operacji. Jest to więc niewyobrażalnie dużo, ale mimo to można pokusić się o pewien zabieg, który pozwoli na uzyskanie swoistego ubezpieczenia od złamania szyfru VMPC. Dzięki temu zabiegowi uzyskamy schemat szyfrowania, który pozostaje bezpieczny nawet jeśli efektywny algorytm łamiący szyfr VMPC zostałby znaleziony.

W tym celu niezbędna jest niewielka modyfikacja oryginalnego 2-fazowego algorytmu inicjowania klucza VMPC-KSA w algorytm trójfazowy (VMPC-KSA3). Jego specyfikacja znajduje się na listingu 2.

Listing 2. Algorytm VMPC-KSA3

Zmienne:

- *c* : ustalona długość klucza w bajtach, 16 ≤ ***c*** ≤ 64
- *K* : ***c***-elementowa tablica zawierająca klucz
- *z* : ustalona długość wektora inicjującego w bajtach, 16 ≤ ***z*** ≤ 64
- *V* : ***z***-elementowa tablica zawierająca wektor inicjujący
- *s* : 8-bitowa zmienna
- \+ oznacza dodawanie modulo 256

```
1. s = 0
2. Dla n od 0 do 255: P[n]=n

3. Dla m od 0 do 767: wykonaj kroki 3.1 - 3.3:
   3.1. n = m modulo 256
   3.2. s = P[ s + P[n] + K[m modulo c] ]
   3.3. x = P[n]; P[n] = P[s];  P[s] = x

4. Dla m od 0 do 767: wykonaj kroki 4.1 - 4.3:
   4.1. n = m modulo 256
   4.2. s = P[ s + P[n] + V[m modulo z] ]
   4.3. x = P[n];  P[n] = P[s];  P[s] = x

5. Dla m od 0 do 767: wykonaj kroki 5.1 - 5.3:
   5.1. n = m modulo 256
   5.2. s = P[ s + P[n] + K[m modulo c] ]
   5.3. x = P[n];  P[n] = P[s];  P[s] = x
```

Kroki od 1 do 4 to nic innego jak oryginalny algorytm VMPC-KSA. Rozszerzony został tutaj jedynie o dodanie kroku 5, który jest powtórzeniem kroku 3. Modyfikacja więc z pozoru wydawać by się mogła mało istotna. Ma jednak bardzo interesujące konsekwencje.

Oryginalny algorytm VMPC-KSA (listing 2, ale bez kroku 5) jest funkcją odwracalną. Innymi słowy – mając wartość permutacji *P* po wykonaniu tego algorytmu oraz znając wartość wektora inicjującego *V*, jesteśmy w stanie stosunkowo łatwo odtworzyć od tyłu poszczególne iteracje kroku 4 i uzyskać permutację *P* po kroku 3. Nazwijmy ją *P1* – jest to permutacja *P* po wykonaniu kroków 1,2 i 3 algorytmu z listingu 2. Zauważmy, że jeśli znamy *P1*, wówczas jesteśmy w stanie deszyfrować dowolne nowe wiadomości zaszyfrowane tym samym kluczem *K*. Każda nowa wiadomość przesyłana jest bowiem z jawnym wektorem inicjującym. Wektor ten (tablica *V*) wystarczy podstawić do kroku 4 algorytmu i po jego wykonaniu uzyskujemy wartość permutacji *P* gotową do zdeszyfrowania wiadomości (pamiętamy, że wciąż nie uwzględniamy kroku 5).

Widzimy więc, że jeśli atakujący znajdzie wartość *P* dla jednej pary szyfrogram - wiadomość, to jest w stanie zdeszyfrować wszystkie nowe wiadomości zaszyfrowane tym samym kluczem. Fundamentalnym czynnikiem, który mu na to pozwala jest fakt, że VMPC-KSA (listing 2 bez kroku 5) jest funkcją odwracalną. Oczywiście wciąż złożoność znalezienie *P* jest szacowana na 2^{900} operacji, więc VMPC-KSA pozostaje bezpiecznym algorytmem, ale mimo to sprawdźmy, jak można się dodatkowo ubezpieczyć od hipotetycznego scenariusza, że ktoś te 2^{900} operacji jakimś sposobem jednak pokona.

Popatrzmy, co się dzieje, jeśli włączymy krok 5 do algorytmu VMPC-KSA, a więc wykorzystamy jego trójfazową wersję VMPC-KSA3, jak przedstawiono na pełnym listingu 2.

Załóżmy, że atakujący wykonał 2^{900} operacji i znalazł wartość klucza wewnętrznego algorytmu – permutację *P*. W wersji bez kroku 5 był on wtedy w stanie odwrócić krok 4, ponieważ znał wartość wektora *V*. Ostatniego kroku algorytmu VMPC-KSA3 nie jest on jednak w stanie już odwrócić z bardzo prostego powodu: wartość klucza *K* jest nieznana dla atakującego. A właśnie *K* steruje 768 operacjami zamiany elementów permutacji *P* w kroku 5. Atakujący bez znajomości *K* nie jest w stanie niczego powiedzieć o permutacji *P2* (powstałej po wykonaniu kroków 1-4, przed krokiem 5).

Zauważmy także, że *P2* będzie dla każdej wiadomości, nawet szyfrowanej tym samym kluczem, miała „całkowicie" inną wartość. Każda wiadomość jest szyfrowana bowiem z wykorzystaniem innej wartości wektora inicjującego *V*, a 768 iteracji algorytmu VMPC-KSA charakteryzuje się prawidłowym efektem dyfuzji. Innymi słowy permutacje wygenerowane z wektorów (czy kluczy) różniących się choćby jednym bitem nie wykazują żadnych wzajemnych relacji innych niż relacje między dwiema losowymi permutacjami.

Permutacje *P2* (po kroku 4) będą więc dla dowolnych dwóch wiadomości losowo od siebie różne. Atakujący znając *P* po wykonaniu całego algorytmu VMPC-KSA3 jest więc dodatkowo „pogrążony", gdyż nie tylko nie potrafi odtworzyć *P2* odwracając krok 5, ale nie jest także w stanie nic powiedzieć o *P2* np. korzystając z analizy statystycznej – ponieważ *P2* dla różnych wiadomości zawsze są od siebie losowo różne.

Dzięki wykorzystaniu algorytmu VMPC-KSA3 atakujący – mimo że pokonał barierę 2^{900} operacji i znalazł wartość *P* – nie jest w stanie zagrozić tajności kolejnych wiadomości szyfrowanych tym samym kluczem. Każda jest bowiem szyfrowana losowo różną postacią permutacji *P*. Atakujący nawet po znalezieniu wartości *P* dla jednej wiadomości tkwi w martwym punkcie, ponieważ nic z wartości *P* nie jest w stanie wywnioskować i kolejne wiadomości pozostają bezpieczne.

Kryptograficzne samobójstwo, a więc o jakości kluczy

Omówione powyżej techniki zapewniają nam całkiem sporo korzyści – tajność dzięki szyfrowaniu, uwierzytelnianie dzięki schematowi VMPC-MAC, a nawet dodatkowy poziom bezpieczeństwa, będący rodzajem ubezpieczenia szyfru na wypadek jego złamania, dzięki algorytmowi VMPC-KSA3. Cóż jednak z tego, jeśli pogrzebanie wszystkich tych korzyści jest zaskakująco proste i co więcej – często popełniane przez użytkowników. Cóż bowiem z tego, że użyjemy wyrafinowanych algorytmów szyfrujących, jeśli zastosujemy zbyt krótkie klucze / hasła?! Wówczas bez względu na jakość algorytmu złamanie naszych szyfrogramów będzie możliwe najbardziej podstawową metodą kryptoanalityczną – tzw. atakiem brutalnej siły (brute force). Polega on tylko na przetestowaniu wszystkich możliwych wartości klucza, aż trafi się na prawidłowy...

O problemie tym mówiłem także w czerwcu na konferencji Enigma 2005 w Warszawie. Zadajmy sobie pytanie – ile znaków ma najdłuższe stosowane przez nas hasło – do logowania, do dostępu do szyfrowanej partycji czy szyfrowanego archiwum plików?

Przyjrzyjmy się tabeli 1, w której zestawiono zmierzone stoperem czasy łamania haseł. Do eksperymentu wykorzystano dostępny za darmo moduł łamania haseł aplikacji VMPC Data Security. Aplikacja ta służy do tworzenia szyfrowanych archiwów zawierających pliki i foldery oraz do szyfrowania tekstów i przesyłania ich pocztą elektroniczną. Jest więc typowym narzędziem kryptograficznym, wykorzystującym technologię VMPC. Mimo że nie są znane sposoby na złamanie któregokolwiek z algorytmów VMPC – moduł łamania haseł dla szyfrogramów utworzonych przy pomocy tej aplikacji działa i łamie hasła na oczach użytkownika. Wykorzystuje bowiem tylko i wyłącznie fakt, że przeszukanie wszystkich możliwych kluczy o niewielkiej długości jest zadaniem na tyle prostym obliczeniowo, że można je wykonać na domowym komputerze i będzie ono skuteczne bez względu na jakość zastosowanych algorytmów. Tabela 1 przedstawia średnie czasy łamania haseł (kluczy) złożonych z małych liter o długościach od 3 do 7 znaków na komputerze 3,4 GHz.

Tabela 1. Średnie czasy łamania haseł metodą brute force

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

Widzimy zatem, że nawet hasło 7-literowe można złamać metodą brutalnej siły w ciągu jednego dnia na jednym domowym komputerze. Niewątpliwie dobrym pomysłem jest rozszerzenie zestawu znaków i stosowanie nie tylko małych liter, ale małych, dużych, a także cyfr. Gdy rozszerzymy zestaw znaków jedynie o duże litery (stosujemy więc i małe i duże) wówczas czasy złamania haseł wzrosną 2^n-krotnie, gdzie n jest długością hasła. Złamanie hasła 4-znakowego zajmie więc już nie około 3 sekundy, ale 3 sekundy razy 2^4 , co daje 48 sekund. Nie jest to jednak rozwiązanie problemu, a jedynie zmniejszenie jego skali.

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”.

Wyjściem jest tu zastosowanie haseł wygenerowanych nie bezpośrednio przez człowieka, który z natury szuka regularności, ale przez generator 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: d8PmZMWy140zVabRBqTHQNX.

Hasło takiej jakości jest kryptograficznie bezpieczne, ale z drugiej strony jego zapamiętanie z pewnością nie jest łatwe. Rozwiązaniem może tu być zapis hasła na nośniku wymiennym, najlepiej w wielu kopiach (funkcja taka jest dostępna np. 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.

Niewątpliwie korzystając z wszelkiego rodzaju algorytmów szyfrujących powinniśmy pamiętać, aby nie zabić ich siły stosując łatwe do złamania hasła. Długość hasła dobierajmy zatem świadomie w zależności od skali poufności szyfrowanych nim danych.

Aplikacja VMPC Data Security

Omawiane algorytmy z rodziny VMPC są od stycznia tego roku dostępne także w postaci gotowej aplikacji do szyfrowania plików/folderów oraz poczty elektronicznej. Aplikacja pozwala także na generowanie losowych haseł z przypadkowych ruchów myszką oraz na nieodwracalne wymazywanie plików z dysku. Podstawowa wersja aplikacji kosztuje 29 zł, ale wersja testowa, moduł do deszyfrowania emaili oraz wykorzystane w artykule narzędzie do łamania haseł są dostępne bez opłat. Więcej informacji znajduje się na stronie *www.VMPCfunction.com/dsp.htm*

Jak widzimy na przykładzie omówionych algorytmów z rodziny VMPC – tajemnicza sztuka szyfrowania wcale nie musi być skomplikowana. Dzięki funkcji jednokierunkowej VMPC, szyfrowi strumieniowemu VMPC, schematowi VMPC-MAC oraz algorytmowi VMPC-KSA3 – zaimplementowanie procedur szyfrujących, a także uwierzytelniających – jest możliwe w kilkunastu linijkach kodu. Jednakże pamiętajmy, że nawet najlepsze algorytmu kryptograficzne nie zapewnią nam bezpieczeństwa, jeśli użyjemy do nich zbyt krótkich kluczy! Uważać na pewno trzeba także na implementację – nawet najdrobniejsza zmiana w algorytmach lub np. dwukrotne użycie tego samego wektora inicjującego – może prowadzić to poważnego obniżenia poziomu bezpieczeństwa.