Marcin KOŁODZIEJCZYK*

TABLICE TĘCZOWE JAKO SKUTECZNA OPTYMALIZACJA ALGORYTMU *BRUTE-FORCE*

STRESZCZENIE

Publikacja jest próbą analizy bezpieczeństwa funkcji skrótów takich jak MD5 czy SHA-1. Omówione zostały podstawowe właściwości i wymagania stawiane przed takimi funkcjami. Artykuł przechodzi kolejno przez poszczególne metody łamania skrótów, zaczynając od prymitywnych metod typu brute-force, poprzez tablice Hellmana oraz ich optymalizację, opartą na punktach wyróżnionych Rivesta, a skończywszy na tablicach tęczowych. Pokazana została ewolucja metod kryptoanalitycznych, które korzystają z ogromnej mocy obliczeniowej komputerów, nie wykorzystując przy tym słabości konkretnych funkcji skrótu. W artykule poruszone zostały praktyczne zagadnienia implementacyjne oraz wskazano na pewne ograniczenia wyżej wymienionych metod kryptoanalizy. Zaprezentowano również rzeczywiste pomiary skuteczności omówionych wyżej metod. Publikacja opisuje również kolizje jako zjawisko niepożądane i w znacznym stopniu utrudniające kryptoanalizę. W publikacji rozdzielone zostały systemy i algorytmy odporne na opisane ataki, takie jak podpis cyfrowy oraz części systemu podatne na atak z wykorzystaniem tablic tęczowych, takie jak krótkie hasła, które w chwili obecnej są jeszcze wykorzystywane. Na koniec podano praktyczną metodę zabezpieczania się przed możliwymi atakami, tzw. „solenie" haseł. Metody tej nie da się niestety dynamicznie włączyć w istniejących systemach. Wymaga ona konkretnej implementacji przez producenta oprogramowania.

***Słowa kluczowe**: tablica tęczowa, funkcja skrótu, algorytm brute-force, funkcja redukcji, kolizje, solenie haseł, metoda Hellmana, punkty wyróżnione Rivesta, podpis cyfrowy, łamanie haseł, fałszywe alarmy*

RAINBOW TABLES AS BRUTE-FORCE ALGORITHM OPTIMIZATION

This article is the analysis of the message digest functions' security, such as MD5 or SHA-1. All basic, common properties and requirements for message digest functions are described in this article. This publication presents the whole path of cryptanalytic method's evolution, beginning from simple methods like brute-force, going through the Hellman's tables and optimization based on distinguished points, described by Rivest and finishing on the rainbow tables. It is shown the evolution path of methods which uses huge computer's power, not some bugs or vulnerabilities in specific functions. This article contains some practice advices, describes potential implementation problems and shows some disadvantages of described cryptanalytic methods. Some, simple real examples are presented and the results of real measurements are contained in this article. One of the parts describes collisions in message digest functions and shows how hard can be cryptanalytic process with many collisions. Author distinguished safe parts of the system or algorithm (like digital signature) and vulnerable parts (like passwords) which are still used. The practical and very safe method (salt passwords) was described at the end of the article. Salt passwords require specific implementation by software developers. Unfortunately, this method cannot be enabled in easy way in existing system.

***Keywords**: rainbow table, Message Digest Function (hash function), brute-force algorithm, reduction function, collision, salt passwords, Time-Memory Trade-Off, distinguished points, digital signature, password cracking, false alarms*

1. WSTĘP

Celem niniejszej publikacji jest próba analizy bezpieczeństwa funkcji skrótów takich jak MD5 czy SHA-1 oraz opis jednej ze skuteczniejszych metod kryptoanalizy tych funkcji jaką jest stosowanie tablic tęczowych. Praca opisuje podstawowe zagrożenia wynikające z tablic tęczowych, a także przestawia skuteczną metodę uniemożliwiającą stosowanie technik związanych z tymi tablicami. Zawarta tutaj została również analiza wydajności tablic tęczowych, opis podstawowych warunków jakie powinna spełniać funkcja redukcji, która czasami jest bagatelizowana w teoretycznych rozważaniach, a także próba ich optymalizacji.

2. CHARAKTERYSTYKA KRYPTOGRAFICZNYCH FUNKCJI SKRÓTU

Na początek rozważań wypada przypomnieć czym są funkcje skrótu (*cryptographic hash functions*[1]). Funkcje skrótu to jednokierunkowe funkcje mieszające, które dla zadanego tekstu wejściowego P tworzą n-bitowy ciąg znaków oznaczany jako $MD(P)$. Nie ma tutaj żadnego ograniczenia na długość tekstu wejściowego. Najpopularniejszymi funkcjami skrótu, używanymi dzisiaj w kryptografii są MD5 (*Message Digest 5*) oraz SHA-1 (*Secure Hash Algorithm*). Tworzą one na wyjściu ciąg odpowiednio 128 i 160 bitów nazywanych skrótem wiadomości. Od funkcji skrótu wymaga

* Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki, Akademia Górniczo-Hutnicza w Krakowie

[1] Pojęcie *hash functions* – często jest tłumaczone jako funkcje „haszujące", jednak pojęcie to jest również używane w stosunku do bardzo prostych funkcji stosowanych w algorytmice do wypełniania pewnych struktur danych np. tablic haszujących (*hash tables*).

się pewnych niezbędnych cech, bez których funkcje te nie spełniałby oczekiwań kryptologów. Pierwszą cechą charakterystyczną i najczęściej wykorzystywaną jest fakt, że bardzo małe zaburzenia danych wejściowych powodują duże zmiany na wyjściu funkcji (tzw. lawinowy efekt rozpraszający). Cecha ta ma bardzo praktyczne zastosowanie na przykład przy ściąganiu dużych plików z Internetu, takich jak obrazy dystrybucji Linuxa czy innego oprogramowania. Zazwyczaj na serwerze, z którego ściągamy pliki, znajdują się obliczone skróty tych plików. Po ściągnięciu pliku, użytkownik może samemu wygenerować skrót i porównać go ze skrótem umieszczonym na serwerze, porównując tylko kilkanaście bajtów. Jeśli skróty są różne, znaczy to, że podczas ściągania plik został uszkodzony[2]. Drugą cechą funkcji skrótu jest szybkość jej działania. Ze względu na powszechne zastosowanie w kryptografii, np. do podpisów cyfrowych czy przechowywania haseł, cecha ta jest dość istotna. Trzecią, przeciwstawną i nie mniej ważną cechą jest fakt, że nie można w łatwy sposób stworzyć funkcji odwrotnej do zadanej funkcji skrótu, która z podanego skrótu wiadomości tworzyłaby wiadomość wejściową. Obliczenie P na podstawie $MD(P)$ musi być problemem obliczeniowo trudnym. W przeciwnym wypadku funkcji skrótu nie można by stosować ani w podpisie cyfrowym, ani do przechowywania haseł, gdyż intruz z łatwością mógłby wygenerować takie hasło. Funkcje skrótów muszą być funkcjami jednokierunkowymi.

Ponieważ, jak już wspomniano wcześniej, nie istnieje żadne ograniczenie na długość tekstu wejściowego P, natomiast $MD(P)$, czyli sam skrót jest zawsze stałej długości. Dlatego musi istnieć wiele tekstów wejściowych P i Q, dla których $MD(P) = MD(Q)$. Taką sytuację nazywa się kolizją. Z kolizjami związana jest czwarta – pominięta wcześniej – cecha funkcji skrótu, która jest bardzo ważna w przypadku podpisu cyfrowego. Dla zadanego P – problem znalezienia Q takiego, że $MD(P) = MD(Q)$ jest problemem obliczeniowo trudnym.

Wymaganie to połączone z wymaganiem, aby funkcje skrótów były jednokierunkowe, ma jeszcze jedną istotną zaletę. Niekiedy stosuje się synchronizację haseł dla różnych systemów, np. tak, aby każdy pracownik miał jedno silne hasło[3], które umożliwia jego autentykację w kilku różnych systemach. Zbyt restrykcyjne wymagania dotyczące haseł powodują, że pracownicy zapisują hasła na kartkach, w notesach itp. Zbyt wielka ilość różnych haseł również staje się przyczyną tego niebezpiecznego zjawiska. W przypadku gdy systemy wymagające autentykacji korzystają z różnych funkcji skrótu, złamanie hasła do jednego systemu nie pociąga za sobą złamania hasła do innych systemów. Dzieje się tak dlatego, że intruz nie ma pewności, że odgadł poprawne hasło a nie hasło kolidujące z poprawnym, czyli dające taki sam hash[4]. Kolizja dla tekstów P i Q dla funkcji skrótu $A()$, nie oznacza kolizji dla funkcji skrótu $B()$ różnej od $A()$.

3. PROSTE METODY ŁAMANIA

Najprostszą metodą łamania funkcji skrótu jest atak typu *brute force* (atak brutalny), jednak ze względu na złożoność czasową jest praktycznie nieużywany i bardzo trudny do przeprowadzenia. Można się jednak próbować zastanowić, czy aby na pewno należy go całkowicie zdyskwalifikować. Odpowiedź na to pytanie zależy od konkretnego zastosowania funkcji skrótu. Jeśli chcielibyśmy przeprowadzić atak na sterownik urządzenia wejścia/wyjścia, podpisany cyfrowo przez producenta, wpisując w miejsce sterownika dowolny, inny ciąg bajtów, tak aby jego skrót był zgodny ze skrótem oryginalnego sterownika, to szanse na powodzenie takiego ataku są marne. W przypadku funkcji MD5 istnieje około 2^{128} kombinacji funkcji skrótu. Prawdopodobieństwo, że dany tekst posiada taką samą funkcję jak skrót sterownika wynosi $p = 1/2^{128}$. Zakładając, że w przypadku ataku *brute force* każda próba jest zdarzeniem niezależnym oraz korzystając z prawdopodobieństwa zdarzenia przeciwnego, możemy wyliczyć prawdopodobieństwo tego, że przy n próbach co najmniej jeden raz trafimy na tekst dający szukany wynik funkcji skrótu.

$$p(n) = 1 - \left(1 - \frac{1}{2^{128}}\right)^n .$$

Tabela 1 przedstawia zależność prawdopodobieństwa sukcesu ataku *brute force* od ilości przeprowadzonych prób.

Tabela 1. Prawdopodobieństwo sukcesu złamania MD5

n	$p(n)$
2^{20}	$3{,}081 \cdot 10^{-33}$
2^{30}	$3{,}155 \cdot 10^{-30}$
2^{40}	$3{,}231 \cdot 10^{-27}$
2^{60}	$3{,}388 \cdot 10^{-21}$
2^{80}	$3{,}553 \cdot 10^{-15}$
2^{128}	0,63212

Widać wyraźnie, że prawdopodobieństwo złamania funkcji MD5 w rozsądnym czasie, dla dowolnego tekstu wejściowego w przypadku metody *brute force* jest prawie równe 0.

[2] Ze względu na sumy kontrolne w nagłówkach pakietów TCP, sytuacja taka występuje niezwykle rzadko, chociaż nie jest niemożliwa, czego doświadczył sam autor.

[3] Silne hasło oznacza hasło niemożliwe do złamania metodą *brute force* w realnym, przydatnym dla intruza czasie. Zbyt restrykcyjne wymagania dot. haseł powodują, że pracownicy zapisują hasła na kartkach, w notesach itd. Zbyt wielka ilość różnych haseł również staje się przyczyną tego niebezpiecznego zjawiska.

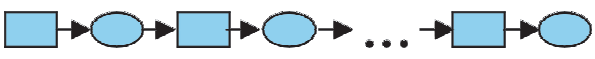
[4] Wynik z funkcji skrótu.

Można jednak sobie zadać pytanie: co się stanie w przypadku haseł? Wszystkich możliwych kombinacji haseł zazwyczaj jest o wiele mniej niż możliwych kombinacji funkcji skrótu. Badając wszystkie hasła, w końcu natrafimy na to właściwe, bądź takie, które daje taki sam skrót w wyniku kolizji. Dodatkowym ułatwieniem jest fakt, że hasła mogą zawierać tylko niektóre znaki ze zbioru ASCII, co daje znacznie mniejszą liczbę możliwych kombinacji do sprawdzenia. Niektóre, zwłaszcza starsze wersje systemów operacyjnych zmniejszają sztucznie bezpieczeństwo przez ograniczenie długości hasła np. do 9 znaków (aby się zalogować, wystarczy podać pierwsze 9 znaków) lub przez dzielenie długiego hasła na części np. 7-bajtowe. Podział taki w znaczącym stopniu zmniejsza liczbę kombinacji wymagających sprawdzenia. Załóżmy, że pojedynczy znak hasła może przyjmować jedną z 70 różnych wartości z alfabetu, który jest podzbiorem ASCII (małe i duże litery, cyfry oraz kilka znaków specjalnych). 14-znakowe hasło posiada 70^{14} kombinacji, podczas gdy przy podziale na dwa 7-znakowe, w najgorszym przypadku musimy sprawdzić $2 \cdot 70^7$ kombinacji.

Warto również zwrócić uwagę, że o ile w przypadku podpisów cyfrowych liczba możliwych kombinacji funkcji skrótu ma znaczenie, o tyle w przypadku haseł ważna jest ich długość oraz liczba znaków w alfabecie, tzw. rozmiar alfabetu[5].

3.1. Metoda Hellmana

Jak już wcześniej wspomniano, problem łamania skrótów takich jak MD5 czy SHA-1 metodą *brute force* jest niesłychanie czasochłonny. Jak więc ograniczyć ilość czasu niezbędną do złamania skrótu? Można wcześniej wygenerować tablicę skrótów, posortować ją alfabetycznie (wg otrzymanego skrótu), a następnie mając dany skrót, odczytać odpowiadający mu tekst wejściowy. Powstaje jednak kolejny trudny do rozwiązania problem: jak pomieścić taką ilość danych? W 1980 roku Martin Hellman opisał metodę kryptoanalizy zwaną *Time-Memory Trade-Off*. Jest ona kompromisem pomiędzy wspomnianymi dwoma metodami – *brute force* i składowanie skrótów w bazie. Metoda Hellmana polega na tworzeniu bardzo długich łańcuchów składających się naprzemiennie z tekstu jawnego i jego skrótu (rys. 1).

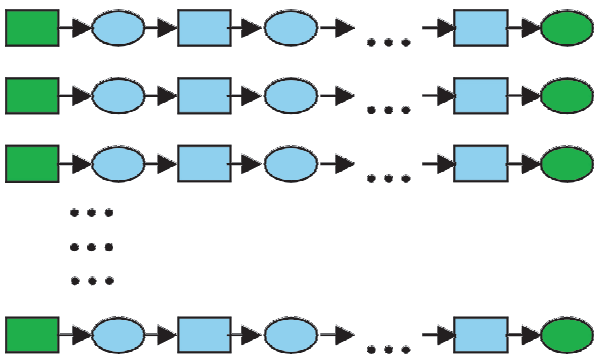


Rys. 1. Łańcuch naprzemiennych tekstów jawnych (prostokąt) i skrótów (elipsa)

Teksty jawne generowane są ze skrótów za pomocą funkcji redukcji. Funkcja redukcji może być dowolną funkcją, która przyjmując skrót na wejściu, zwraca na wyjściu tekst jawny nad dopuszczalnym alfabetem generującym teksty jawne[6]. Mając tablicę takich łańcuchów, możemy zapamiętywać tylko pierwszy i ostatni element tablicy, gdyż środek może być wygenerowany w stosunkowo krótkim czasie, zależnym od długości łańcucha. Kryptoanaliza polega na sprawdzeniu, czy szukany skrót znajduje się w tablicy końcowych elementów łańcucha. Jeśli tak, to znaczy, że przedostatni element łańcucha jest tekstem dającym szukany skrót. Wystarczy teraz wygenerować łańcuch, zaczynając od jego początku, który jest pamiętany razem z jego końcem. Jeśli natomiast szukanego skrótu nie odnajdziemy, to możemy wygenerować kolejny skrót według następującej reguły:

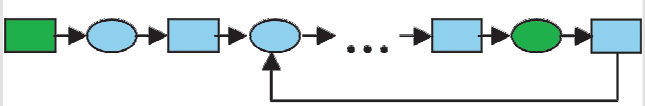
$$MD_{n+1} = MD(R(MD_n)) \tag{1}$$

gdzie MD_n oznacza n-ty skrót w łańcuchu (oznaczony jako elipsa), MD funkcję skrótu, a R funkcję redukcji. Mając kolejny skrót, możemy znów przeszukać końcowe elementy łańcuchów. Jeśli go tam znajdziemy, to istnieje duże prawdopodobieństwo, że drugi od końca tekst w łańcuchu zawierającym szukany skrót jest szukanym tekstem jawnym. Jeśli nie znaleziono skrótu, można powtarzać operacje, generując kolejne skróty i szukając ich w tablicy. Maksymalna liczba iteracji jest ograniczona długością łańcuchów (rys. 2).



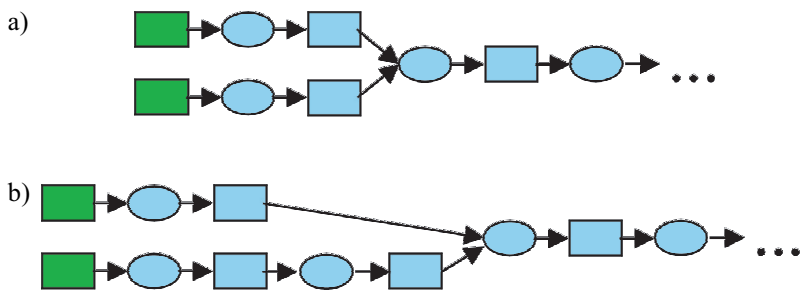
Rys. 2. Tablica Hellmana – zapamiętywane są tylko pierwsze i ostatnie elementy łańcuchów (elementy zielone)

Należy jednak zauważyć, że funkcja skrótu nie jest, a funkcja redukcji nie musi być różnowartościowa. Mogą więc wystąpić kolizje w zbiorze takich łańcuchów oraz cykle w pojedynczych łańcuchach. Takie przypadki zostały pokazane na rysunkach 3 i 4.



Rys. 3. Cykl w pojedynczym łańcuchu

[5] Użytkownik, który używa ograniczonego alfabetu (np. same małe litery), znacznie osłabia swoje hasło.
[6] Ograniczenie to ma sens w przypadku haseł, które są generowane przez alfabet będący podzbiorem właściwym zbioru ASCII.



Rys. 4. Kolizja w dwóch tekstów jawnych w dwóch różnych łańcuchach

Ewentualne cykle – mimo że nie są pożądane – nie mają tak dużego wpływu na proces kryptoanalizy, jak kolizje. Przykładowy cykl zilustrowany jest na rysunku 3. W przypadku cyklu tracimy jedynie część łańcucha (tracona część jest nadmiarowa). Kolizje natomiast są zjawiskiem bardzo niepożądanym. Kolizje występujące na tym samym miejscu w ciągu (rys. 4a) powodują zmniejszenie liczby różnych skrótów na końcu łańcucha, co znacznie wydłuża kryptoanalizę, gdyż należy badać każdy początkowy tekst jawny dający szukany skrót. W przypadku kolizji w dwóch różnych miejscach łańcucha, rzecz się ma niewiele lepiej, gdyż po kilku iteracjach kryptoanalizy dojdziemy do skrótu, który już był badany, gdyż znajdował się pod innym indeksem w innym łańcuchu. Aby zmniejszyć ilość kolizji, w metodzie Hellmana zaleca się używanie wielu tablic z różnymi funkcjami redukcji. Wtedy nawet w przypadku kolizji, funkcja redukcji wyprodukuje różne teksty jawne, które następnie z dużym prawdopodobieństwem dadzą różne skróty.

3.2. Optymalizacja Rivesta

W 1982 roku Ronald Rivest zaproponował optymalizację metody opartej na tablicach Hellmana. Metoda Rivesta opiera się na tzw. *Distinguished Points* (dosłownie: wyróżnione punkty), którymi są skróty wiadomości posiadające pewną szczególną własność. Własność ta nie jest ściśle zdefiniowana i można ją dowolnie określać. Przykładem mogą być skróty, których pierwszych 10 bitów jest równe 0. Można w ten sposób, w niewielkim stopniu zmniejszyć ilość zajmowanego miejsca przez skróty, jednak prawdziwa zaleta tkwi w przyspieszeniu procesu kryptoanalizy. Implementując tablice łańcuchów na tablicach mieszających[7], musimy poświęcić nieco więcej miejsca w pamięci na ich przechowywanie, co nie jest pożądane. Inne algorytmy wyszukiwania (np. w posortowanej tablicy skrótów) mają złożoność obliczeniową co najmniej $O(\ln n)$, tak więc mając w tablicy tylko skróty spełniające określoną, wcześniej zdefiniowaną własność (mając tylko *distinguished points*), większość operacji wyszukiwania możemy pominąć, z góry odrzucając skróty nieposiadające założonej własności. W przypadku 10^{12} skrótów pomijamy 40 operacji porównywania skrótów (gdyż $2^{40} \approx 10^{12}$) w każdej iteracji, co może zaowocować ogromnym przyspieszeniem samej kryptoanalizy.

W przypadku metody Rivesta sam łańcuch (rys. 1) nie ma stałej długości. Specyfikuje się jedynie jego maksymalną dopuszczalną długość. W przypadku gdy podczas tworzenia pojedynczego łańcucha napotkamy na skrót, który posiada określoną wcześniej właściwość, kończymy tworzenie łańcucha i zapamiętujemy go w tablicy (rys. 2). Jeśli natomiast zbudowany został łańcuch o maksymalnej długości i ostatni skrót nie jest punktem wyróżnionym, taki łańcuch nie jest dodawany do tablicy. Nie jest to duża wada, gdyż podobnie jak w przypadku metody Hellmana wykorzystuje się tutaj wiele różnych tablic z różnymi funkcjami redukcji. Istnieje więc duże prawdopodobieństwo, że odrzucony łańcuch zakończy się odpowiednim skrótem w innej tablicy, korzystającej z innej funkcji redukcji i zostanie tam dodany do tablicy łańcuchów. Metoda Rivesta rozwiązuje również częściowo problem kolizji. Jeśli dwa różne łańcuchy dają ten sam skrót, który jest wyróżnionym punktem, to zapamiętany powinien zostać tylko dłuższy łańcuch.

4. TABLICE TĘCZOWE

Philippe Oechslin w swoim artykule opisuje tablice tęczowe jako optymalizację czasową metody Hellmana. Pomysł tablic tęczowych opiera się na tym, że funkcja redukcji jest nie tylko funkcją skrótu, ale również numeru iteracji, w którym jest wywoływana. Tak więc kolejny skrót jest wyliczany jest już według następującego wzoru:

$$MD_{n+1} = MD(R(MD_n, n)) \qquad (2)$$

Właśnie ze względu na zmianę w funkcji redukcji, Oechslin nazwał swoje tablice, tablicami tęczowyni[8] (*rainbow tables*). Zmiana funkcji redukcji w znacznym stopniu uodparnia tablice tęczowe na kolizje. Jeśli kolizja w dwóch różnych łańcuchach nastąpi na dwóch różnych pozycjach,

[7] *Hash Table* – struktura danych o złożoności obliczeniowej O(1), Tablice mieszające nie powinny być nigdy maksymalnie zapełnione. W literaturze podaje się, że maksymalne zapełnienie tablicy, przy którym spełnia jeszcze swoją funkcję, wynosi 80%.

[8] Funkcja redukcji w każdym z łańcuchów zmienia się jak kolory tęczy.

automatycznie jest rozwiązywana przez funkcję redukcji w kolejnym kroku. Problem występuje jedynie w przypadku kolizji dokładnie na tych samych pozycjach. Podobnie rzecz ma się w przypadku cykli; z tablic tęczowych, dzięki podmianie funkcji redukcji, są one całkowicie wyeliminowane.

Kryptoanaliza posiada jednak nieco większą złożoność obliczeniową (zależną od długości łańcucha). O ile złożoność obliczeniowa metody Hellmana wynosiła $O(n \ln m)$, Rivesta natomiast $O(n)$, gdzie n było odpowiednio długością i maksymalną długością łańcucha a m rozmiarem tablicy, o tyle złożoność kryptoanalizy tablic tęczowych jest znacznie większa. Zwróćmy uwagę, że generując kolejny skrót, należy użyć przedostatniej funkcji redukcji. W drugiej iteracji należy użyć najpierw drugiej od końca funkcji redukcji, a następnie przedostatniej itd. Tak więc w każdym kroku iteracji ilość wywołań funkcji redukcji i skrótu zwiększa się o 1. Złożoność obliczeniowa całego algorytmu jest więc $O(n^2 \ln m)$. Metoda ta jest jednak najskuteczniejszą ze wszystkich trzech, gdyż automatycznie rozwiązuje większość kolizji i nie wymaga pamiętania ogromnej liczby funkcji redukcji. Szybkość działania kryptoanalizy zwiększa się przez zmniejszenie długości pojedynczego łańcucha.

5. OGRANICZENIA METOD KRYPTOANALIZY I PUŁAPKI

Wszystkie trzy opisane wyżej metody mają pewne ograniczenia, na które należy zwrócić uwagę. Pierwszą niezbyt oczywistą pułapką jest fakt, że jeśli po pewnej liczbie kroków znajdziemy skrót w tablicy łańcuchów, to nie oznacza to, że znaleźliśmy tekst jawny dający szukany skrót. Przypadek ten jest zilustrowany na rysunku 5.

Jasnoszary fragment łańcucha na rysunku 5 nigdy nie znajdował się w zapamiętanym łańcuchu, a mimo to po pewnej liczbie kroków doszliśmy do końca łańcucha. Idąc od początku naszego łańcucha, nigdy nie natrafimy na jasnoszary fragment zawierający szukany tekst jawny i jego skrót.

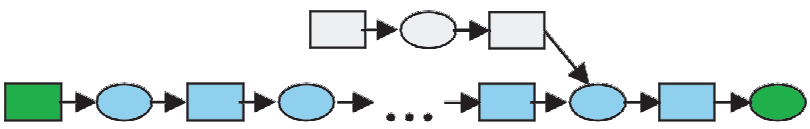
Kolejną pułapką, z jaką zetknął się osobiście autor, jest wymóg, aby funkcja redukcji generowała każde słowo nad dopuszczalnym alfabetem z jednakowym prawdopodobieństwem. Prostym przykładem złamania tego założenia jest generowanie słów różnej długości z jednakowym prawdopodobieństwem. Przykładowo, mając do dyspozycji hasła generowane przez 62-znakowy alfabet, prawdopodobieństwo wygenerowania jednoznakowego hasła musi być 62 razy mniejsze niż prawdopodobieństwo wygenerowania dwuznakowego, prawdopodobieństwo dwuznakowego 62 razy mniejsze niż trójznakowego itd. Niespełnienie tego założenia prowadzi do wielu kolizji, tym razem nie w funkcjach skrótu, a w samych funkcjach redukcji.

6. PRZYKŁADOWE POMIARY

Kolizje nie są zjawiskiem sprzyjającym wszystkim opisanym wyżej metodom. Tablice tęczowe mimo że bardzo dobrze sobie z nimi radzą to jednak posiadają największą złożoność obliczeniową. Przeprowadzono przykładowe pomiary obrazujące ilość kolizji w zależności od użytej funkcji redukcji (tab. 2 i 3).

Tabela 2 pokazuje w jaki sposób, wprowadzenie dodatkowego parametru na wejściu funkcji redukcji (w przypadku tablic tęczowych) ogranicza liczbę kolizji. Co ciekawe, całkowitoliczbowe dzielenie tego dodatkowego parametru w funkcji redukcji przez 2 (tak, aby dwa sąsiednie kroki używały tego samego parametru) potrafi w ekstremalnym przypadku zwiększyć liczbę kolizji nawet 20-krotnie. Tabela 3 obrazuje wyniki tablic tęczowych dla 5-znakowych haseł. W tym przypadku liczba możliwych haseł jest ok. 1,7 razy większa od 2^{29}. Przy 4-znakowych hasłach, liczba kombinacji jest około 1,14 razy mniejsza od 2^{24} i około 1,76 razy większa od 2^{23}. W tabeli 2 uwzględniono również wyniki dla jeszcze mniejszych tablic.

Po przeprowadzeniu pomiarów nasuwa się pytanie, czy nie warto do tablic tęczowych wprowadzić również kilku funkcji redukcji w celu zwiększenia ich skuteczności. W przypadku wielordzeniowych procesorów wprowadzenie dodatkowych funkcji nie wydłużyłoby czasu łamania skrótu, gdyż każda funkcja mogłaby korzystać z pojedynczego rdzenia procesora. Trudno jednak zakładać z góry, jaką liczbę rdzeni będzie miał użytkownik łamiący hasła. Zrównoleglanie równie dobrze, jeśli nie lepiej, wychodzi w przypadku generowania tablic. Po zrównolegleniu algorytmu (przetwarzając pojedyncze łańcuchy na pojedynczych rdzeniach) otrzymujemy przyspieszenie liniowe[9].



Rys. 5. Fałszywe alarmy

[9] Przyspieszenie liniowe oznacza, że na n rdzeniach program wykona się n razy szybciej.

Tabela 2. Porównanie funkcji redukcji dla 4-znakowych haseł z 62-znakowego alfabetu

Długość łańcucha	Ilość łańcuchów	Liczba wywołań funkcji skrótu	R(MD)		R(MD, n)	
			Ilość różnych skrótów na końcach łańcuchów	Różne skróty na końcach łańcuchów [%]	Ilość różnych skrótów na końcach łańcuchów	Różne skróty na końcach łańcuchów [%]
2^{16}	256	2^{24}	26	10,156	150	58,594
2^{12}	4 096	2^{24}	2 174	53,076	2 601	63,501
2^{8}	65 536	2^{24}	41 667	63,579	41 612	63,495
2^{16}	128	2^{23}	25	19,531	96	75,000
2^{12}	2 048	2^{23}	1 182	57,715	1 586	77,441
2^{8}	32 768	2^{23}	25 540	77,942	25 501	77,823
2^{12}	1 024	2^{22}	656	64,063	900	87,891
2^{11}	2 048	2^{22}	1 716	83,789	1 808	88,281
2^{11}	1 024	2^{21}	923	90,137	963	94,043
2^{10}	1 024	2^{20}	990	96,679	994	97,070

Tabela 3. Funkcja redukcji zależna od kroku (stosowana w tablicach tęczowych) dla haseł 5-znakowych

Długość łańcucha	Ilość łańcuchów	Liczba wywołań funkcji skrótu	Ilość różnych skrótów na końcach łańcuchów	Różne skróty na końcach łańcuchów w procentach
2^{20}	512	2^{29}	511	99,805
2^{16}	8 192	2^{29}	8 154	99,536
2^{10}	524 288	2^{29}	521 855	99,536

7. ZABEZPIECZENIA FUNKCJI SKRÓTU

Na koniec wypada powiedzieć kilka słów o metodach zabezpieczeń przed tego typu tablicami. Jedną z metod obrony przed atakami wykorzystującymi tablice tęczowe jest dodawanie „soli" (*salt*) do haseł. Polega to na zwiększeniu zbioru możliwych haseł poprzez doklejenie do hasła określonego ciągu znaków składającego się z dowolnych znaków (niekoniecznie należących do alfabetu tworzącego hasła). Przykładowo do hasła można dokleić czas utworzenia konta użytkownika z dokładnością do sekund lub nawet milisekund. Następnie w bazie haseł można przetrzymywać skróty haseł powiększonych o swoją „sól". Proces „solenia" haseł powoduje nieskuteczność wszystkich wymienionych wyżej metod, gdyż aby takie hasło złamać, funkcja redukcji musiałaby tworzyć już „posolone" hasła, co zwiększyłoby liczbę możliwości o kilka rzędów wielkości i znacznie wydłużyłoby proces tworzenia tablic. Ponadto taka sól może zależeć od systemu w ten sposób, że dwa różne systemy mogą dodawać różny rodzaj soli. Ogranicza to zastosowanie konkretnej tablicy tylko do pojedynczego systemu, a być może nawet tylko do pojedynczej wersji danego systemu.

8. PODSUMOWANIE

Mimo wielu prób łamania funkcji skrótu, nadal nie istnieją uniwersalne metody pozwalające złamać skrót w zadowalająco krótkim czasie. Wymienione wyżej tablice wymagają specjalnych założeń o dopuszczalnym języku generującym hasła. Ponadto metody tablicowe wymagają wcześniejszego wygenerowania danych, które może trwać nawet miesiącami. Metody te jednak są dość niebezpieczne, gdyż mając wcześniej wygenerowaną tablicę, łamanie skrótu hasła trwa około kilku minut. „Solenie" haseł jest bardzo skuteczną ochroną przed tego typu tablicami, gdyż znacząco zwiększa ono zbiór możliwych tekstów jawnych. Wydaje się, że w chwili obecnej istnieje realne zagrożenie ataku przy użyciu tablic tęczowych na hasła starszych systemów, w których przechowywane są jedynie skróty haseł. Wypada jeszcze zwrócić uwagę na czas, w jakim opisane metody powstawały. Od opublikowania metody Hellmana w 1980 roku do odkrycia tablic tęczowych minęło 13 lat (pierwsza publikacja – 2003 r.). Analizując tempo rozwoju metod kryptoanalitycznych, można przypuszczać, że używanie skrótów w podpisie elektronicznym w najbliższej przyszłości będzie uznawane za metodę bezpieczną obliczeniowo.

Literatura

[1] Yilmaz B.: *Time-Memory Trade-off using distinguished points.* 2007.
[2] Oechslin P.: *Making a Faster Cryptanalytic Time-Memory Trade-Off.* 2003.
[3] Maziarz P.: *Wykorzystywanie tęczowych tablic do łamania haseł.* hakin9.pl, 2008.

[4] Hellman M.: *A cryptanalytic time-memory trade off.* 1980.
[5] Ogiela M.R.; *Bezpieczeństwo systemów komputerowych.* Wydawnictwa AGH, Kraków, 2002.
[6] Denning D.E.: *Cryptography and Data Security.* Addison-Wesley, 1982.

Wpłynęło: 29.01.2009

Marcin KOŁODZIEJCZYK

Jest absolwentem kierunku informatyka na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie. Studia dzienne, magisterskie ukończył w 2006 roku. Wcześniej uczęszczał do V Liceum Ogólnokształcącego im. Augusta Witkowskiego w Krakowie. Od 2008 roku jest słuchaczem studium doktoranckiego AGH w Krakowie, na wydziale EAIiE, na kierunku informatyka. W pracy naukowej zajmuje się kryptografią oraz bezpieczeństwem protokołów sieciowych.

W 2004 roku odbył studenckie praktyki wakacyjne w firmie IBM BTO Business Consulting Services Sp. z o.o. z siedzibą w Krakowie. Od 2005 roku pracuje w firmie Motorola Polska Electronics Sp. z o.o. W chwili obecnej zawodowo zajmuje się poprawą bezpieczeństwa wytwarzanych systemów informatycznych. Od 2009 roku posiada certyfikat „CompTIA Security+ Certified Professional 2008 edition". Jest autorem kilku publikacji z dziedziny bezpieczeństwa systemów komputerowych, a także recenzentem publikacji dla: „Computers and Mathematics with Applications" oraz „Security and Communication Networks".

e-mail: marcink@agh.edu.pl