Kryptografia i bezpieczeństwo danych - szyfrowanie z kluczem symetrycznym - uzupełnienie

Sławomir Samolej
ssamolej.kia.prz.edu.pl
ssamolej@prz.edu.pl

W naszym wykładzie dochodzimy do końca dyskusji na temat szyfrowania z kluczem symetrycznym. Pozostało nam kilka zagadnień, które uzupełnią potrzebne informacje, aby zamknąć ten temat.

Otrzymywanie wielu kluczy z jednego

Typowy scenariusz. Mamy pojedynczy klucz źródłowy (SK – ang. Source Key) uzyskany z:

- Sprzętowy generator liczb losowych
- Uzyskany z protokołu wymiany kluczy (będzie omówione później)

Potrzebujemy wielu kluczy do zabezpieczenia sesji:

- Osobne klucze do szyfrowania ruchu w każdą stronę kanału transmisyjnego; wiele kluczy do szyfrowania CBC z nonce.

Cel: wygenerowanie wielu kluczy z jednego klucza źródłowego

2

Jednym z zagadnień, które jeszcze nie było omówione, jest sposób otrzymywania wielu kluczy na podstawie pewnego klucza źródłowego. Klucz źródłowy może być wygenerowany lokalnie z zastosowaniem sprzętowych generatorów liczb losowych, lub otrzymany z zewnątrz z zastosowaniem protokołu wymiany kluczy. Metody wymiany kluczy zostaną omówione później. Potrzeba generacji takich kluczy jest naturalna. Np. w protokole TLS potrzebujemy pary kluczy, każdego do szyfrowania transmisji w jedną stronę. A tak w zasadzie, to czterech kluczy, ponieważ potrzebujemy osobnych kluczy do szyfrowania i obliczania MAC. Podobnie, w schemacie szyfrowania CBC z wartością nonce, do przeprowadzenia prawidłowego szyfrowania potrzebujemy dwóch kluczy. W kryptografii opracowano więc osobny mechanizm nazywany „funkcją otrzymywania kluczy" (ang. Key Derivation Function – KDF).

Klucz źródłowy jest losowy z jednolitym rozkładem prawdopodobieństwa

F: jest PRF z przestrzenią kluczy K i wartościami $\{0,1\}^n$

Załóżmy, że klucz źródłowy SK jest otrzymany jako ciąg wylosowanych bitów, a rozkład prawdopodobieństwa jest jednolity

- Definiujemy funkcję otrzymywania kluczy (KDF) jako:

KDF(SK, CTX, L) :=
F(SK, (**CTX || 0**)) || F(SK, (**CTX || 1**)) || ⋯ || F(SK, (**CTX || L**))

CTX: unikatowy łańcuch, który definiuje aplikację

3

Zakładamy, że mamy bezpieczną funkcję F generującą liczby pseudolosowe na bazie kluczy z pewnej przestrzeni K. Zakładamy też, że nasz klucz źródłowy (SK) ma jednolity rozkład prawdopodobieństwa w przestrzeni kluczy K. Możemy wtedy posłużyć się kluczem i funkcją do otrzymania wielu kluczy wprowadzając na wejście generatora klucz, parametr CTX (ang. Context) oraz długość. Funkcję F będziemy uruchamiać kolejno dla wartości licznika 0, 1, …, L. W ten sposób możemy wygenerować ciąg bitów o takiej długości, jaka nam odpowiada i „podzielić go" na zestaw potrzebnych nam kluczy. Metoda jest dosyć oczywista. Stosuje sią w niej bezpieczną funkcję generującą liczby pseudolosowe jako generator liczb pseudolosowych. Jaką wartość ma mieć CTX? Najprościej, unikatową dla każdej aplikacji. Ma on spowodować, że nawet jeśli kilka aplikacji, np. SSH, IPsec, czy HTTPS otrzymają ten sam losowy klucz, to ponieważ mają unikatowe CTX, wygenerują różne klucze dla przeprowadzenia swoich sesji szyfrowania.

Co się stanie, jeśli klucz nie ma jednolitego rozkładu prawdopodobieństwa?

Przypomnienie: Funkcje PRF są generatorami liczb pseudolosowych, jeśli klucz podany na wejście ma jednolity rozkład prawdopodobieństwa w przestrzeni kluczy K

- SK nie ma jednolitego rozkładu ⇒ wyjście PRF może nie wyglądać losowo

Zwykle jednak klucze nie mają jednolitego rozkładu prawdopodobieństwa:

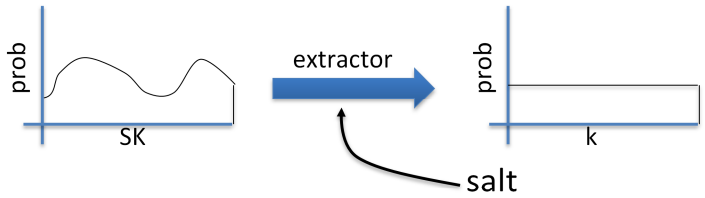
- Protokół wymiany kluczy: klucz ma rozkład jednolity w pewnym podzbiorze przestrzeni K
- Sprzętowe generatory liczb losowych: mogą wytwarzać stronnicze wyniki

4

Jeśli nasz klucz nie ma jednolitego rozkładu prawdopodobieństwa w przestrzeni kluczy K, to zastosowane go na wejście funkcji generującej liczby pseudolosowe nie gwarantuje otrzymania ciągu pseudolosowego. Pojawia się wtedy problem.
Tak na prawdę w wielu przypadkach klucz źródłowy, którym dysponujemy nie ma właściwości jednolitego rozkładu prawdopodobieństwa w przestrzeni K. W protokole wymiany kluczy otrzymujemy klucze losowe i różniące się, ale posiadający jednolity rozkład dla pewnego podzbioru przestrzeni kluczy K. Nie możemy też tak naprawdę polegać na sprzętowych generatorach liczb pseudolosowych, ponieważ podają wyniki nie zawsze dające się zakwalifikować jako w pełni losowe. W rezultacie, jedynie co możemy założyć, że dysponujemy klucz źródłowy wygenerowany jako bardzo mało powtarzalny ciąg bitów, być może przyjmujący pewne wartości w sposób bardziej stronniczy. Naszym zadaniem jest eliminacja tej stronniczości.

Paradygmat Extract-then-Expand

Krok 1: wyodrębnij pseudolosowy klucz k z bezpiecznego klucza SK



salt: ustalony jawny łańcuch wybrany losowo

Krok 2: rozszerz k używając go jako klucz dla PRF jak wcześniej

5

Rozszerzanie kluczy, które nie mają jednolitego rozkładu w przestrzeni K odbywa się z zastosowaniem podejścia Wyodrębnij-potem-Rozszerz (ang. Extract-the-Expand). W pierwszym kroku następuje „wyodrębnienie" pseudolosowego klucza z klucza źródłowego. Ekstraktor przekształca nierównomierną funkcję prawdopodobieństwa wystąpienia poszczególnych kluczy w inną funkcję, tak naprawdę nieodróżnialną od jednolitego rozkładu. Zwykle do ekstraktora wprowadza się dodatkowy parametr nazywany po angielsku salt (sól?). Powoduje on wprowadzenie dodatkowego mieszania w kluczy podczas wykonywania ekstraktora, co w konsekwencji przekształca je w ciągi nierozróżnialne od losowych. Ten ciąg jest jawny, ustalany na pewną początkową wartość raz na zawsze. Jedyny warunkiem jest, że jego wybór odbywa się w sposób losowy. Ostatecznie ekstraktor wraz z salt potrafi przekształcić SK w klucz nieodróżnialny od losowego z jednolitym rozkładem prawdopodobieństwa.
Jeśli wyodrębniliśmy taki klucz o odpowiednich parametrach w dziedzinie prawdopodobieństwa, to możemy posłużyć się metodą rozszerzania (KDF), którą wprowadziliśmy dla kluczy losowych z równomiernym rozkładem prawdopodobieństwa.

HKDF: KDF z HMAC

Implementuje paradygmat extract-then-expand:

- wyodrębnij: użyj **k ⟵ HMAC(salt, SK)**

- Potem rozszerz klucz używając HMAC jako PRF z kluczem **k**

6

Ustandaryzowana metoda rozszerzania klucza nosi nazwę HKDF. Jest to metoda otrzymania rozszerzonego klucza stosująca algorytm HMAC. Algorytm HMAC jest stosowany zarówno do wyodrębniania jak i rozszerzania klucza. W czasie wyodrębniania klucza salt (wartość znana publicznie) jest traktowany jako klucz algorytmu HMAC, a SK jako dane, które mają być przekształcone z zastosowaniem HMAC. Dysponując pseudolosowym kluczem metoda traktuje z kolei algorytm HMAC jako funkcję generującą ciąg pseudolosowy z kluczem k jak na slajdzie nr 3.
Podsumowując, jeśli otrzyma się klucz źródłowy, to nie należy go bezpośrednio stosować do szyfrowania, czy obliczania MAC. Należy go przekształcić w klucz/klucze sesji z zastosowaniem KDF i dopiero wtedy je zastosować w przyjętej konstrukcji kryptograficznej. Zwykle rozszerzenie kluczy jest ustandaryzowane z zastosowaniem metody HKDF.

Otrzymywanie kluczy w oparciu o hasło

(ang. Password-Bassed KDF (PBKDF))

są za mało losowe ...

Otrzymywanie kluczy z haseł:

- Nie jest stosowane HKDF: hasła mają niewystarczającą entropię
- Otrzymane klucze byłyby podatne na atak słownikowy

(więcej o tym później)

Obrona PBKDF: zastosowanie **salt** i **wolnej funkcji hash**

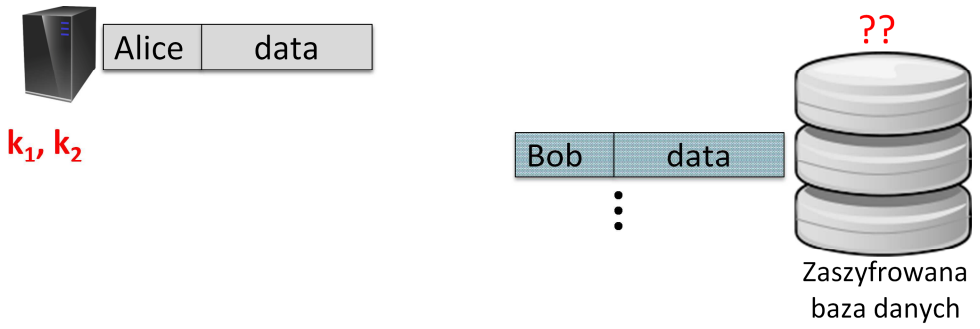
Standardowe podejście: **PKCS#5** (PBKDF1)

$H^{(c)}$(pwd ll salt): powtarzaj funkcję hash c razy

7

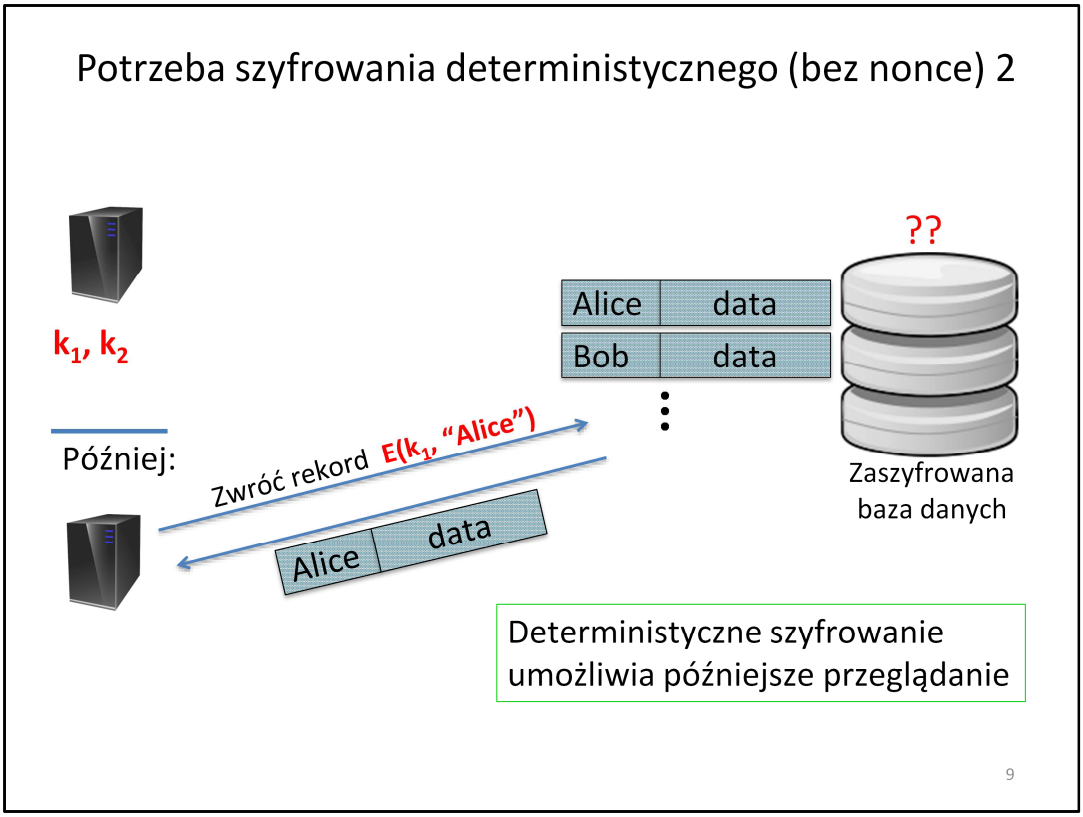
Można sobie wyobrazić, że źródłem kluczy do szyfrowania mogą być hasła zaproponowane przez użytkownika. Hasła same w sobie niestety mają zbyt mało entropi (to znaczy są za mało losowe, żeby można było na ich podstawie wytworzyć bezpieczny klucz), żeby na ich podstawie generować klucze (byłyby one np. zbyt podobne). Dodatkowo, tak wygenerowane hasła byłyby podatne na atak słownikowy (będziemy go omawiać później).
Z drugiej strony bardzo potrzebujemy takich mechanizmów generowania kluczy. Jak to należy robić? Po pierwsze, warto wiedzieć, że do generowania takich haseł nie wolno stosować schematu HKDF. Nie jest on zaprojektowany do pracy z hasłami i nie zapewnia odpowiedniej losowości wygenerowanych kluczy. Po drugie w generowaniu takich kluczy stosuje się znowu salt oraz specjalną wolna funkcję hash. Proces jest ustandaryzowany i opisany w dokumencie PKCS#5. Wersja, która będzie pokazana nosi nazwę PBKDF1 i jest zaimplementowana w większości bibliotek kryptograficznych (nie należy jej implementować samodzielnie). Stosowanie tej metody powinno polegać na wywołaniu odpowiedniej funkcji, która przyjmie Wasze hasło i zwróci klucz odpowiedniej jakości. W praktyce na wejście funkcji hash wprowadza się Wasze hasło połączone z wartością salt. Następnie proces obliczania hash powtarza się wiele razy (c-razy), np. 10 000 razy. W rezultacie wygenerowanie klucza z hasła trwa pewien znaczący ułamek sekundy, ale nie na tyle znaczący, żeby blokował pracę komputera na dłużej. System nie jest w dalszym ciągu zabezpieczony przed atakiem słownikowym, to znaczy próbą „zgadnięcia" Waszego hasła i przeliczenia z niego (wraz ze znanym salt) funkcji hash c razy. Rezultatem jest wtedy próba zgadnięcia klucza. Zabezpieczeniem tu jest właśnie „długość"/„wielokrotność" obliczania hash. Podanie kolejnego słowa do zgadnięcia, jako hasła zajmuje znaczącą część sekundy. Próba zgadnięcia hasła spośród 200 000 000 000 możliwości oznacza bardzo długi czas obliczeń…

Potrzeba szyfrowania deterministycznego (bez nonce) 1



8

Okazuje się, że system szyfrowania deterministycznego (taki, który zawsze zamienia wiadomość dokładnie na ten sam szyfrogram i nie stosuje wartości nonce) ma wiele zastosowań.
Załóżmy, że mamy serwer bazodanowy i chcemy przechowywać na nim zaszyfrowane dane. Rekord bazy danych ma postać indeksu (Alice, Bob, ….) oraz pola z danymi. Po zaktualizowaniu rekordu jest on szyfrowany. Indeks jest szyfrowany kluczem k1, a dane kluczem k2. Zaszyfrowane rekordy są przesyłane do bazy danych.

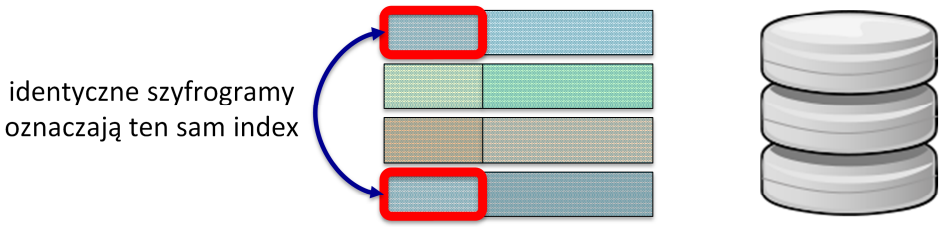


Przy takim podejściu do szyfrowania (deterministycznym), jeśli chcemy odzyskać dane, to wysyłamy do bazy danych polecenie zwróć nam rekord, ale taki, że jego pole indeksu jest równe zaszyfrowanej wartości „Alice”. Interesującą konsekwencją takiego podejścia jest fakt, że baza danych nie ma pojęcia, jakie dane przechowuje. Nawet zapytanie odbywa się poprzez porównanie zaszyfrowanych pól, a nie jawnych teksów. Godzimy się więc na szyfrowanie, które dla takich samych tekstów daje zawsze taki sam szyfrogram, ale uzyskujemy inny cel. Instytucja przechowująca dane, jeśli nie zna kluczy, to nie ma pojęcia, co jest przechowywane w bazie danych.

Problem: przecież szyfrowanie deterministyczne nie jest odporne na atak z wybranym tekstem jawnym

Problem: atakujący może stwierdzić, że dwa szyfrogramy szyfrują tę samą wiadomość ⇒ wyciek informacji

Prowadzi to do poważnych ataków, kiedy przestrzeń wiadomości M jest mała.



10

Takie rozwiązanie może być mylące, zawłaszcza dla nas, którzy wiedzą, że zastosowanie szyfrowania deterministycznego powoduje, że takie szyfrowanie nie jest odporne na atak z wybranym tekstem jawnym. Atakujący przeglądając szyfrogramy jeśli zauważy, że dwa z nich są identyczne, to oznacza, że zawierają te same wiadomości. W praktyce takie ataki mają znaczenie, jeśli przestrzeń wiadomości jest mała, np. jeśli przesyłamy kody klawiszy po każdym uderzeniu (zdalny terminal). Atakujący może szybko zbudować słownik ze wszystkich możliwych szyfrogramów (dla wiadomości równych pojedynczemu bajtowi będzie to tylko 256 możliwych szyfrogramów) i wiedząc jakie wiadomości kryją się pod każdym z szyfrogramów może w przyszłości łatwo deszyfrować kolejne depesze.
W przypadku bazy danych atakujący może dostrzec, że dwa pola w dwóch różnych rekordach mają takie same szyfrogramy. I te pola powinny zawierać indeksy w bazie danych. Tutaj znowu, chociaż nie zna on treści indeksu, to dowiaduje się, że dwa indeksy są identyczne.

Rozwiązanie: przypadek unikatowych wiadomości

Załóżmy, że szyfrator **nigdy nie** szyfruje tej samej wiadomości dwukrotnie:

para (k , m) nigdy się nie powtarza

To się wydarza, kiedy szyfrujący:

- Wybiera wiadomość w sposób losowy z dużej przestrzeni (np. klucze)
- Struktura wiadomości zapewnia unikatowość (np. unikatowy ID użytkownika)

11

Rozwiązaniem problemu jest nałożenie restrykcji na wiadomości szyfrowane za pomocą jednego klucza. Można przyjąć zasadę, że szyfrujący nigdy nie będzie szyfrował tej samej wiadomości dwukrotnie z zastosowaniem tego samego klucza, lub inaczej nigdy para (k, m) nie występuje dwa razy. W jednym procesie szyfrowania musi się zmienić klucz lub wiadomość.
Okazuje się, że takie przypadki zachodzą w realnych rozwiązaniach. Przykładowo, baza danych przechowuje wiadomości losowe z dużej przestrzeni. Jakie wiadomości? Po prostu klucze szyfrowania. Czyli jednym kluczem bazy danych szyfrujemy zbiór kluczy szyfrujących, które z bardzo dużym prawdopodobieństwem są różne. Inny przypadek, to struktury danych z gwarancją, że każda jest inna. Tutaj wracając do baz danych można sobie przypomnieć, że indeks może korespondować z unikatowym identyfikatorem użytkownika. Stąd typowa relacyjna baza danych, gdzie każdy wiersz jest unikatowy spełnia kryterium różnych wartości wiadomości.
Można zdefiniować pojęcie bezpieczeństwa na atak z wybranym tekstem jawnym dla szyfrowania deterministycznego. I warunkiem spełnienia bezpieczeństwa jest nieszyfrowanie dwukrotnie za pomocą tego samego klucza tej samej wiadomości.

Typowe błędy

- Uznanie, że **CBC ze stałym IV** jest bezpiecznym deterministycznym szyfrowaniem
- Uznanie, że **tryb licznikowy, ze stałym IV** jest bezpiecznym deterministycznym szyfrowaniem

Szyfrowanie deterministyczne – podsumowanie na tym etapie

Potrzebne do utrzymania zaszyfrowanych indeksów w bazie danych

- Daje możliwość przeglądania rekordów, bez ich odszyfrowywania

Zdefiniowane jest bezpieczeństwo na atak z wybranym tekstem jawnym dla szyfrowania deterministycznego:

- Bezpieczeństwo jest zachowane, jeśli nigdy nie zaszyfrujemy tej samej wiadomości dwukrotnie używając tego samego klucza:

para (key , msg) jest unikatowa

13

Przypominając wcześniejsze rozważania, szyfrowanie deterministyczne przydaje się do szyfrowania indeksów w bazach danych, ponieważ można ponownie wyliczyć szyfrogram indeksu i pobrać dany rekord z bazy danych, bez rozszyfrowywania go. Warunkiem bezpieczeństwa jest niedopuszczenie, żeby ta sama wiadomość była dwukrotnie szyfrowana z zastosowaniem tego samego klucza.
Dalej poznamy kilka konstrukcji, które zapewniają bezpieczeństwo na atak z wybranym tekstem jawnym dla szyfrowania deterministycznego.

Konstrukcja 1: Syntetyczne IV (SIV)

Niech (E, D) będzie konstrukcją bezpieczną na CPA **E(k, m ; r) ⟶ c**

Niech F:K × M ⟶ R będzie bezpieczną PRF

Definiujemy: $\mathbf{E_{det}((k_1,k_2), m)} =$

$r \leftarrow F(k_1, m)$

$c \leftarrow E(k_2, m; r)$

wyjście: c

Tw: $\mathbf{E_{det}}$ jest semantycznie bezpieczne na CPA z deterministycznym szyfrowaniem.

Dobrze się nadaje dla wiadomości dłuższych niż jeden blok AES (16 bajtów)

14

Pierwsza konstrukcja wygląda następująco. Załóżmy, że dysponujemy systemem szyfrowania bezpiecznym ze względu na CPA. Taka konstrukcja przyjmuje klucz k, wiadomość m, oraz „losowość" r (we wszystkich algorytmach bezpiecznych na CPA wprowadzaliśmy mechanizm losowości, który zapewniał właśnie odporność na CPA). W systemie mamy również bezpieczną PRF, która przekształca parę (k, m) w wartość losową należącą do przestrzeni R. Małe r należy do przestrzeni R.
Algorytm SVI działa następująco. Dysponujemy dwoma kluczami k1 i k2. Na początku stosujemy funkcję F i wiadomość m z kluczem k1 do wytworzenia „losowości" w postaci ciągu r. Potem używamy otrzymaną wartość r do zaszyfrowania wiadomości m z kluczem k2. Schemat „wypracowuje" losowość na podstawie klucza k1 i wiadomości m, a następnie wprowadza tę losowość do algorytmu szyfrowania wiadomości m z kluczem k2.
Podana konstrukcja jest deterministyczna, ponieważ dla danych m i k1 r jest zawsze takie same, ale równocześnie bezpieczna na atak z wybranym tekstem jawnym (pod warunkiem, że wiadomość będzie szyfrowana danym kluczem tylko raz).
Warto zwrócić uwagę, że ta konstrukcja dobrze nadaje się do szyfrowania dłuższych niż jeden blok AES wiadomości. Do szyfrowania wiadomości krótszych stosuje się inną konstrukcję.

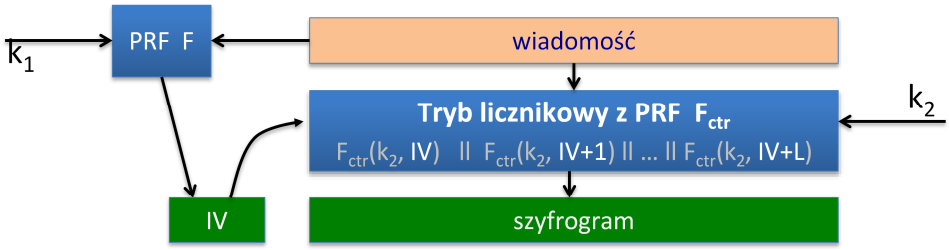
Zapewnienie integralności szyfrogramu

Cel: zapewnienie bezpieczeństwa na CPA w szyfrowaniu deterministycznym oraz integralności (uwierzytelnienia) wiadomości

⇒ **DAE: deterministic authenticated encryption**

Rozważmy SIV w specjalnym przypadku: SIV-CTR

czyli SIV gdzie szyfrowanie odbywa się w trybie licznikowym z losowym IV

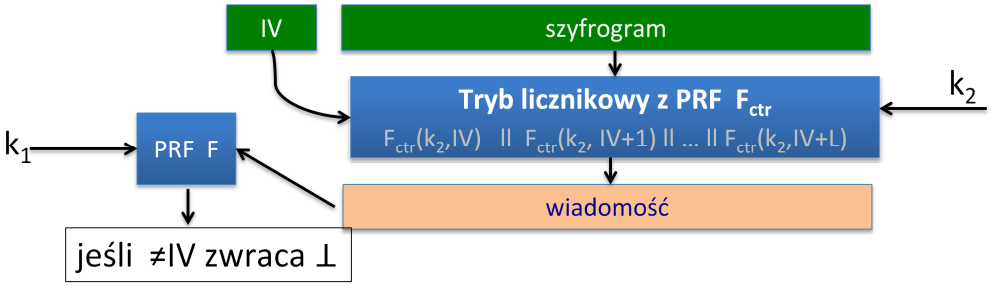


15

Ciekawą właściwością zastosowania SIV jest otrzymanie szyfrowania z uwierzytelnieniem „za darmo”. Nie musimy wtedy dodawać obliczania MAC do zapewnienia integralności. Naszym celem jest opracowanie deterministyczne szyfrowanie z uwierzytelnianiem. Wystarczy rozważyć schemat szyfrowania SIV-CTR, czyli bazujący na syntetycznym generowaniu IV tryb licznikowy. Podobnie jak w ogólnym przypadku na podstawie wiadomości i k1, z zastosowaniem funkcji F następuje utworzenie IV. Potem to IV jest zastosowane do zaszyfrowania wiadomości z zastosowaniem trybu licznikowego.

Sprawdzanie integralności wiadomości

Odszyfrowywanie:



Tw: jeśli F jest bezpieczną PRF i CTR na podstawie F_{ctr} jest bezpieczny na CPA
to konstrukcja SIV-CTR na podstawie F, F_{ctr} dostarcza DAE (ang. Deterministic Authenticated Encription) – deterministyczne szyfrowanie z uwierzytelnieniem.

16

Rozważmy, jak może być przeprowadzone odszyfrowywanie. Szyfrogram składa się z IV i zaszyfrowanych danych. Algorytm deszyfrujący zwraca kandydata na wiadomość. Dla wiadomości możemy jeszcze raz przeliczyć funkcję F. Jeśli zwróci ona IV, to wiadomość zachowała integralność, jeśli nie, to zwracamy sygnał niepowodzenia odszyfrowywania. Udowodniono, że taka konstrukcja spełnia wymagania szyfrowania deterministycznego z uwierzytelnieniem.

Konstrukcja 2: zastosowanie PRP (**P**seudo**r**andom **P**ermutation)

Niech (E, D) będzie bezpieczną PRP. $E: K \times X \longrightarrow X$

Tw: (E,D) jest semantycznie bezpieczna na CPA z szyfrowaniem deterministycznym.

W konsekwencji zastosowanie AES:
AES daje bezpieczne szyfrowanie deterministyczne dla 16-bajtowych wiadomości.

Dłuższe wiadomości??
Potrzebujemy PRPs bazujące na dłuższych przestrzeniach wiadomości…

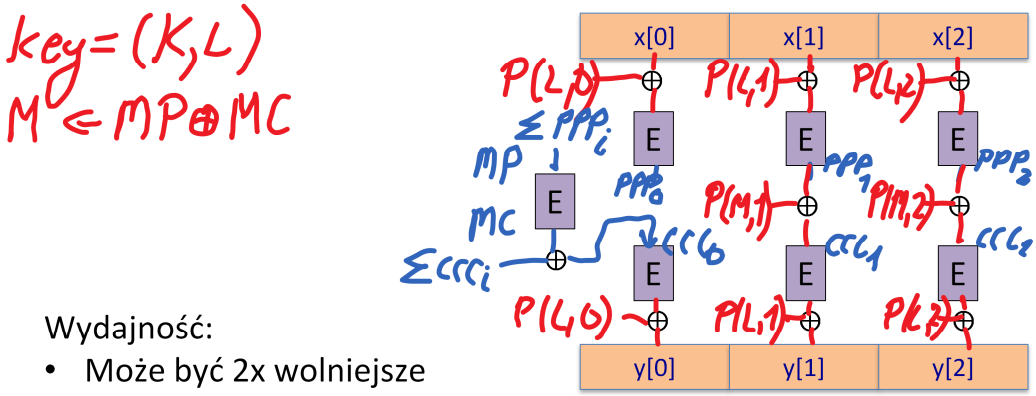
17

Poprzednie rozwiązanie jest dobre dla długich wiadomości. To, które będzie teraz omawiane nadaje się do wiadomości krótkich (np. do rozmiaru jednego bloku danych blokowego algorytmu szyfrującego). W tym rozwiązaniu stosujemy tylko PRP w bezpośredni sposób. Udowodniono twierdzenie, że zastosowanej bezpośrednio PRP do bloku danych jest bezpieczne ze względu na atak CPA w problemie szyfrowania deterministycznego.
Ponieważ AES ma właściwość PRP, możemy przeprowadzić deterministyczne szyfrowanie 16 bajtów z tym algorytmem. Na razie nie zapewniamy integralności szyfrogramu, ale powiemy o tym za chwilę. Co, jeśli wiadomości są dłuższe? Pamiętamy, że budowaliśmy konstrukcje szyfrujące, które umożliwiały bezpieczne szyfrowanie dłuższych bloków danych, jeśli dysponowaliśmy bezpiecznymi metodami szyfrowania małych bloków danych. Spróbujmy więc opracować taką metodę dla szyfrowania deterministycznego…

EME: konstruowanie długiego bloku PRP

Niech (E, D) będzie bezpieczną PRP. $E: K \times \{0,1\}^n \longrightarrow \{0,1\}^n$

EME: jest PRP na $\{0,1\}^N$ dla $N \gg n$



Wydajność:

- Może być 2x wolniejsze niż SIV

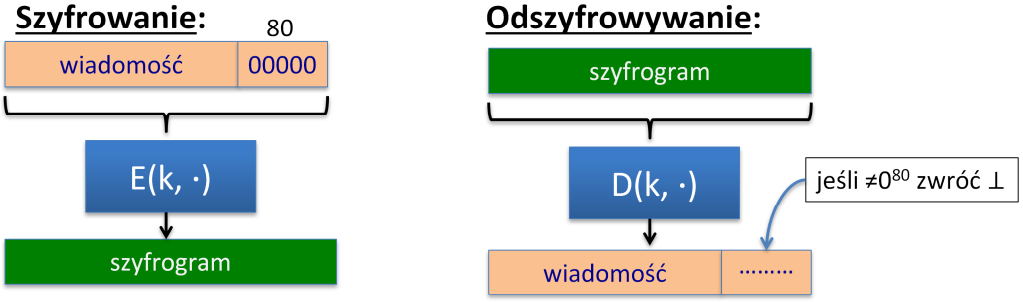
18

Zakładamy, że mamy bezpieczny mechanizm PRP dla małych bloków danych. Konstruujemy mechanizm losowej permutacji dla bloków znacznie dłuższych niż bazowy. EME używa 2 kluczy K i L. Długa wiadomość jest dzielona na bloki. Dla każdego bloku z zastosowaniem pewnej funkcji P, której argumentami są klucz i licznik, oblicza się „pad". Każdy blok ma inny pad. Następnie wykonuje się operację xor na blokach wiadomości z wartością wygenerowaną z wyjścia funkcji P. Tak przekształcone bloki danych są podawane na algorytm szyfrujący E z kluczem K (algorytm ma własność bezpiecznego PRP). Nazwijmy wyjścia z bloków szyfrujących PPP0, PPP1, PPP2 itd. Kolejną operacją jest wykonanie xor na wszystkich blokach PPPi i podane tej wartości na kolejny blok szyfrowania E. Xor wszystkich wartości PPPi nazywany jest MP, natomiast zaszyfrowane MP nazywane jest MC. Wykonując MP xor MC otrzymujemy klucz M. Wykorzystujemy go do wygenerowania następnej serii padów. Następnie wykonuje się xor pomiędzy wartościami PPPi (bez PPP0) i obliczonymi padami. Daje to serię wartości CCC1, CCC2 itd. Te wszystkich tych wartościach wykonuje się xor. Otrzymaną wartość dodaje się xoe do MC i w taki sposób powstaje CCC0. Wszystkie wartości CCCi szyfruje się z zastosowaniem algorytmu E. Kolejny raz oblicza się wartość paddingu dla każdego bloku danych z zastosowaniem funkcji P z kluczem L i licznikiem. Obliczony padding dodaje się xor z wyjściami bloków szyfrujących ostatecznie otrzymując bezpieczną pseudolosową permutację długiego ciągu danych, czyli rozszerzenie drugiej konstrukcji szyfrowania deterministycznego na dłuższe ciągi danych do zaszyfrowania. Jest udowodnione twierdzenie, że jeśli (E, D) jest bezpieczną PRP, to pokazana konstrukcja jest również bezpieczną PRP, pod warunkiem , że długość przetwarzanych danych (N) jest znacznie większa od pojedynczego bloku generującego „małe" PRP (np. pojedynczego bloku szyfrowania AES).
Ważną własnością podanego rozwiązania jest to, że można je łatwo zrównoleglić. Widać w procesie obliczeniowym dwie fazy, kiedy obliczenia mogą być prowadzone współbieżnie, oraz jedną (środkową) fazę, kiedy trzeba przeprowadzić obliczenia sekwencyjnie.
Jeśli chodzi o wydajność podanej metody, to trzeba wiedzieć, że funkcje P są proste i szybkie, tak, że często pomija się ich wpływ na czas obliczeń. Natomiast trzeba zauważyć, że dla każdego bloku danych dwukrotnie oblicza się algorytm E. W konsekwencji pokazana metoda, w porównaniu z prawidłowo oprogramowaną SIV będzie dwa razy wolniejsza.
Podsumowując, można przyjąć, że PRP są dobrymi metodami szyfrowania deterministycznego dla krótkich porcji danych, podczas gdy SIV dobrze nadaje się do szyfrowania danych o większej długości.

Deterministyczne szyfrowanie z uwierzytelnieniem na bazie PRP (1)

Cel: osiągnięcie bezpieczeństwa dla szyfrowania deterministycznego i integralności szyfrogramu

⇒ **DAE: deterministic authenticated encryption**



19

Jak dodać do szyfrowania deterministycznego opartego na PRP mechanizm zapewniający integralność? Rozwiązanie okazuje się proste. Wystarczy dodać na koniec wiadomości serie zer. Zaś po odszyfrowywaniu należy sprawdzić, czy ponownie ta seria zer z wiadomości odszyfrowanej się znalazła. Seria powinna być odpowiednio długa, na slajdzie zaproponowane jest 80. Okazuje się, że o bezpieczeństwie tego rozwiązania decyduje właśnie długość tego bloku z zerami. Twierdzenie o bezpieczeństwie mówi, że jeśli wartość $1/2^n$, gdzie n to jest liczba zer dołączonych do wiadomości jest pomijalnie mała, to tak skonstruowany system szyfrowania deterministycznego spełnia warunki bezpiecznego szyfrowania z uwierzytelniniem.

Szyfrowanie dysków – brak możliwości „wydłużenia” danych (1)

Sektory w dysku mają stały rozmiar (np. 4KB)

⇒ szyfrogram nie może być dłuższy nić wiadomość (to jest M = C)

⇒ trzeba zastosować szyfrowanie deterministyczne, bez uwierzytelnienia

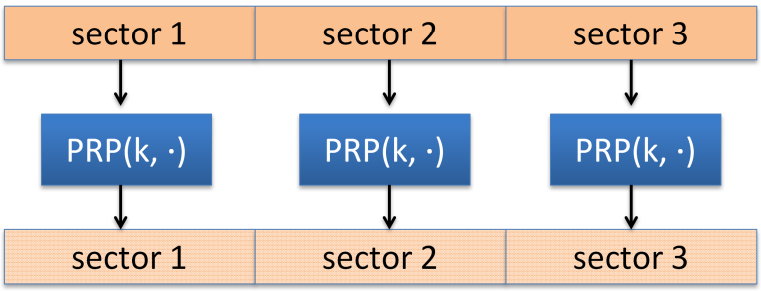
Tw. Pom.: jeśli (E, D) jest deterministycznym szyfrem bezpieczny m na atak CPA z własnością M=C to (E, D) jest PRP.

⇒ każdy sektor będzie musiał być zaszyfrowany z zastosowaniem PRP

20

Gdyby się zastanowić nad zagadnieniem szyfrowania dysków na poziomie sektorów, to okazuje się, że napotykamy problem polegający na tym, że nie dysponujemy żadnym dodatkowym miejscem w sektorze do zapamiętania szyfrogramu. Musimy znaleźć taką metodę szyfrowania, w której długość szyfrogramu może być co najwyżej długości wiadomości. Technicznie rzecz biorąc możemy ustalić, że przestrzeń szyfrogramów jest równa przestrzeni wiadomości.
W takim wypadku jesteśmy „skazani” na szyfrowanie deterministyczne, bo nie możemy zawrzeć w szyfrogramie żadnej dodatkowej informacji o losowości. Rozwiązanie, które musimy zaproponować nie może też zawierać możliwości sprawdzania integralności, bo znowu nie mamy „miejsca” na dodatkowe bity wiadomości zawierające MAC. Naszym celem jest więc „tylko” zapewnienie bezpieczeństwa na ataki z wybranym tekstem jawnym dla szyfrowania deterministycznego.
Istnieje udowodnione twierdzenie mówiące, że jeśli istnieje bezpieczny szyfr deterministyczny, który przekształca wiadomość w szyfrogram o takiej samej długości, to jest on PRP (Preudorandom Permutation – Pseudolosową permutacją). Wniosek z tego jest taki, że musimy szyfrować stosując PRP.

Szyfrowanie dysków – brak możliwości „wydłużenia” danych (2)



Problem: sektor 1 i sektor 3 mogą zawierać takie same dane

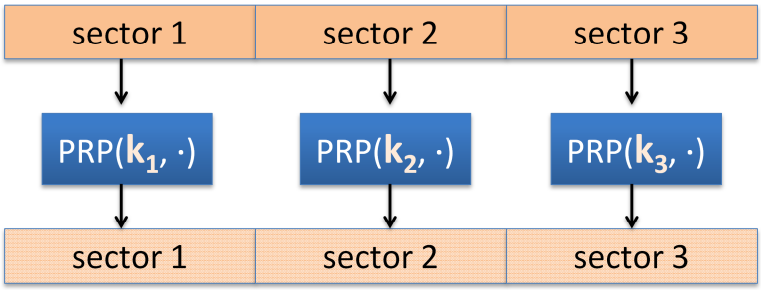
- Powoduje to wyciek danych, podobnie jak w przypadku schematu szyfrowania elektronicznej książki kodowej (ECB mode)

Czy można to zrobić lepiej?

21

Zastanówmy się więc, jak można by było zaszyfrować dysk? Podzielmy „dane do zaszyfrowania” na bloki równe sektorom dysku i każdy z nich zaszyfrujmy zastosowaniem PRP i tego samego klucza. Powracamy tu do standardowego problemu z szyfrowaniem deterministycznym. Jeśli sektor 1 i 3 mają takie same dane, to ich szyfrogramy będą identyczne, co doprowadza to wycieku informacji, którego nie chcemy. W przypadku dysków łatwo by było wtedy zidentyfikować, które z sektorów są puste, a które nie.
Powstaje pytanie, czy można to zaprojektować lepiej. Okazuje się, że tak…

Szyfrowanie dysków – brak możliwości „wydłużenia” danych (3)



Zapobiega poprzednim wyciekom informacji

- … ale atakujący może sprawdzić, czy sektor uległ zmianie i odwrócić tę zmianę

Zarządzanie kluczami: prosta konstrukcja $k_t = PRF(k, t)$, t=1,…,L

Czy można ro rozwiązać jeszcze lepiej?

22

Pierwszym pomysłem mogłoby być szyfrowanie każdego sektora innym kluczem. Wtedy, nawet jeśli zawartość sektora 1 byłaby taka sama, jak sektora nr 3, to ich szyfrogramy byłyby różne. To powoduje uniknięcia wycieku informacji z rozwiązania na poprzednim slajdzie. W tym rozwiązaniu też istnieje pewien mankament. Jeśli użytkownik zmieni choćby jeden bit w sektorze, to zmianie ulegnie szyfrogram tego sektora, niestety, kiedy użytkownik z powrotem „przywróci” stan tego bitu, to szyfrogram tego sektora powróci do początkowego stanu. Atakujący może zauważyć taką sytuację. Żeby zapobiec wykryciu takich zdarzeń trzeba zaangażować znaczne moce obliczeniowe i zwykle, jeśli przyjmie się tą metodę, to razem ze świadomością, że taki wyciek informacji może nastąpić. Może się wydawać, że trzeba w takim układzie pamiętać wiele kluczy, ale tak nie jest. Klucze dla sektorów można wygenerować „na bieżąco”. Wystarczy posiadać jeden „bazowy” klucz k, i na jego podstawie wraz z pomocnicą zmienną t, oznaczającą numer sektora z zastosowaniem PRF (funkcji generującej wartości pseudolosowe) generować klucze dla poszczególnych sektorów. Takie generowanie kluczy jest bezpieczne, ponieważ funkcja generuje liczby pseudolosowe, a podanie na jej wejście różnych wartości numerów sektorów da zawsze liczby losowe i różne od siebie.
Mamy więc bezpieczniejszą konstrukcję, która wymaga generowania tych dodatkowych kluczy i oczywiście powstaje pytanie, czy można to zaprojektować jeszcze lepiej? Okazuje się, że tak…

Blokowe szyfrowanie dostrajalne (ang. Tweakable block encryption)

Cel: skonstruować **wiele** PRPs z klucza $k \in K$.

Składnia: $E, D: K \times T \times X \longrightarrow X$

dla każdego $t \in T$ i $k \leftarrow K$:

$E(k, t, \cdot)$ jest odwracalną funkcją na zbiorze X, nierozróżnialną od losowej

Zastosowanie: używamy numer sektora jako „dostrojenie”

$\Rightarrow$ każdy sektor dostaje własne PRP

Udowodniono bezpieczeństwo blokowych szyfrów dostrajalnych.

23

Można postawić sobie nowy cel: mamy jeden klucz, ale chcemy z niego skonstruować wiele PRP. Można to uzyskać przez zaszyfrowanie klucza z zastosowaniem pseudolosowej liczby, ale istnieje bardziej efektywna metoda. Nowym elementem systemem jest „tweak” – ulepszenie i to wprowadza pojęcie dostrajalne szyfrowanie blokowe.
W nowym trybie szyfrowania wejściem algorytmu jest zbiór kluczy, zbiór ulepszeń i zbiór wiadomości, wyjściem zaś zbiór wiadomości (szyfrogramów). Jeśli weźmiemy losowy klucz i polepszacz oraz ustalimy dany klucz dla danego polepszacza, to otrzymamy funkcję odwracalną. Czyli blokujemy wartość klucza ale zmieniamy polepszacze i otrzymujemy niezależne PRP przekształcające nam zbiór X (wiadomości) z zbiór X (szyfrogramów).
W naszym przykładzie polepszaczem może być numer sektora. Wtedy każdy sektor otrzyma swoje własne przekształcenie PRP.

Przykład 1: trywialna konstrukcja

Niech (E,D) będzie bezpieczną PRP, E: $K \times X \longrightarrow X$.

- Trywialna konstrukcja z dostrajaniem: (niech K = X)

$$E_{tweak}(k, t, x) = E\big(\, E(k, t),\ x\big)$$

⇒ Żeby zaszyfrować n bloków potrzebujemy 2n wykonań E(.,.)

24

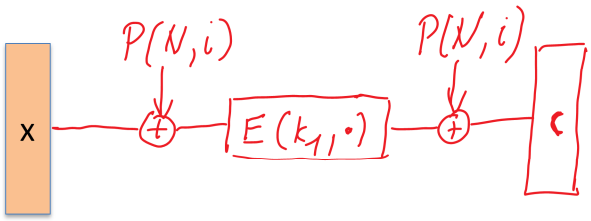
W podstawowym rozwiązaniu szyfrowania z dostrajaniem możemy założyć, że przestrzeń kluczy jest równa przestrzenie wiadomości. Czyli funkcja szyfrująca bierze wiadomość o rozmiarze X, klucz o rozmiarze X i zwraca wyjście o rozmiarze X. (np. 128bitów x 128 bitów daje 128 bitów). Można wtedy rozwiązać problem szyfrowania z ulepszeniem w taki sposób, że najpierw szyfrowane jest ulepszenie za pomocą klucza ki ten szyfrogram jest brany jako klucz szyfrowania wiadomości x. Jeśli chodzi o wydajność takiego podejścia, można zauważyć, że zaszyfrowanie pojedynczego bloku danych wymaga dwukrotnego wykonania algorytmu szyfrowania, stąd szyfrowanie n bloków wymaga 2n uruchomień algorytmu E.

Przykład 2: szyfr blokowy z dostrajaniem XTS

Niech (E,D) będzie bezpieczną PRP, E: $K \times \{0,1\}^n \longrightarrow \{0,1\}^n$.

- XTS: $E_{tweak}((k_1,k_2), (t,i), x) =$

$N \longleftarrow E(k_2, t)$

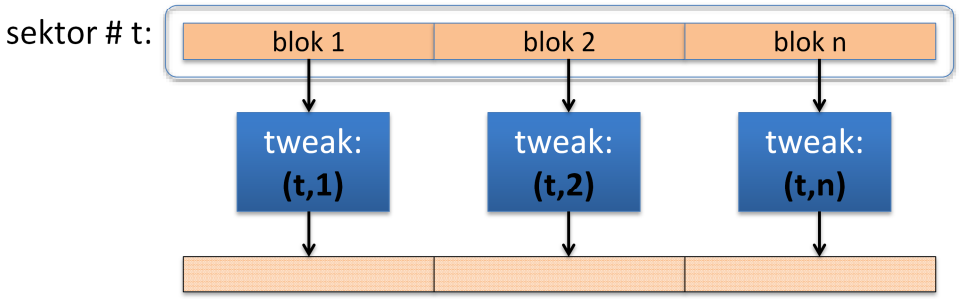


$\Rightarrow$ żeby zaszyfrować n bloków potrzebujemy n+1 wykonań E(.,.)

25

Można opracować schemat szyfrowania z dostrajaniem działający szybciej. Jednym z przykładów jest schemat XTS, wywodzący się ze schematu XEX. Załóżmy , że dysponujemy „normalnym” nie dostrajanym szyfrem E, który za pomocą klucza k szyfruje pewien (niekoniecznie mający długość klucza) ciąg bitowy. Na jego podstawie definiujemy szyfr z dostrajaniem w następujący sposób. Niech klucz składa się z pary kluczy k1 i k2, zaś ulepszenie jest opisane za pomocą dwóch wartości „t” i „i”. Ulepszacz powstaje ze złożenia tych dwóch wartości. x to blok, który chcemy zaszyfrować. Szyfrowanie rozpoczyna się od zaszyfrowania z zastosowaniem k2 części ulepszacza „t”. Wynik tego szyfrowania nazywamy N. Szyfrowanie rozpoczynamy od wykonania xor na wiadomości i wartości wyjściowej pewnej funkcji P (funkcja pad). Funkcja P bierze jako wejście wartość N oraz indeks i. Funkcja P jest bardzo szybka, tak że często pomija się czas jej wykonywania w ogólnej analizie czasowej omawianego systemu. Następnie wyjście operacji xor szyfruje się szyfrem E z kluczem k1. Z kolei otrzymane wyjście funkcji szyfrującej ponownie dodaje się xor z wywołaniem funkcji P, o takich samych parametrach jak wcześniej. W rezultacie otrzymujemy szyfrogram c. Ważną własnością pokazanego rozwiązania jest jego wydajność. W porównaniu z wcześniejszym przykładem, do zaszyfrowania n bloków danych wystarczy wykonać tylko n+1 szyfrowań z zastosowaniem algorytmu E. Wykonanie tego pierwszego szyfrowania wartości t jest kluczowe dla bezpieczeństwa tego systemu szyfrowania i nie wolno go pominąć.

Zastosowanie XTS do szyfrowania dysków



- uwaga: PRP na poziomie bloku, nie na poziomie sektora.
- Popularne w systemach szyfrowania plików:

  Mac OS X-Lion, TrueCrypt, BestCrypt, …

26

Jak wygląda zastosowanie XTS w szyfrowaniu dysków? Brany jest pod uwagę numer sektora dysku. Sam sektor jest dzielony na 16 bajtowe bloki. Wtedy blok 1 jest szyfrowany z polepszaczem (t,1), blok drugi z polepszaczem (t,2) itd. Ostatecznie każdy blok otrzymuje własny PRP, a cały sektor jest kolekcją PRP poszczególnych bloków. Nie można powiedzieć, że każdy sektor ma swój PRP, ale, że każdy blok sektora. Wchodząc w szczegóły, to tak naprawdę gwarantowana jest tu odporność na atak CPC na poziome poszczególnych bloków.
Pokazane rozwiązanie jest popularne w wielu systemach szyfrowania dysków, które zostały wymienione na slajdzie.

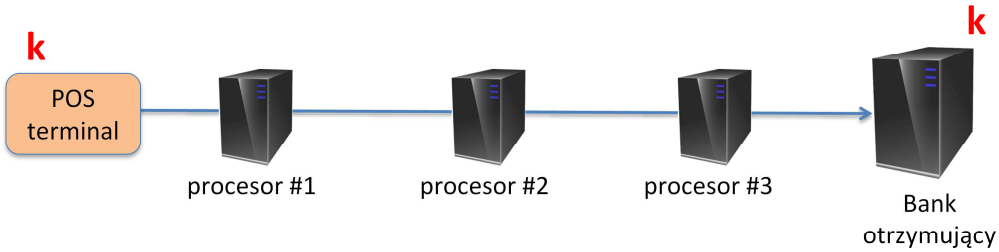
Podsumowanie (szyfrowanie blokowe z dostrajaniem)

- Szyfrowanie blokowe z dostrajaniem stosuje się, gdy potrzebujemy wielu niezależnych PRP wyprowadzonych z jednego klucza
- Konstrukcja XTS jest bardzie efektywna od podstawowego rozwiązania
  - Obie wykorzystują niewielkie bloki : 16 bajtów dla AES
- EME jest metodą szyfrowania z dostrajaniem dla dużych bloków danych
  - Jest 2x wolniejsza niż XTS

27

Szyfrowanie numerów kart kredytowych

Format karty kredytowej: **bbbb bbnn nnnn nnnc** (≈ 42 biów)



Cel: szyfrowanie punkt - punkt

Pośrednie procesory spodziewają się „widzieć” numer karty kredytowej

⇒ zaszyfrowana karta kredytowa powinna wyglądać jak karta kredytowa

28

Kolejnym zagadnieniem powiązanym z szyfrowaniem symetrycznym jest szyfrowanie zachowujące format ukrywanych danych. Jest to kolejne zagadnienie kryptograficzne mające zastosowanie praktyczne, a mianowicie jest używane do szyfrowania numerów kart płatniczych.
Typowa karta kredytowa ma numer składający się z 16 cyfr, podzielonych na 4 bloki po cztery liczby. Pierwsze 6 liczb to numer BIN opisujący wystawcę karty. Przykładowo, karty wydane przez VISA zawsze zaczynają się od 4. Wszystkie karty wydane przez MasterCard zaczynają się od liczb 51 lub 55 itd. Następne dziewięć cyfr koduje numer konta powiązany z danym klientem. Ostatnia liczba to suma kontrolna z wcześniejszych 15 cyfr. Jest więc około 20 000 numerów BIN. Jeśli chcemy zapisać numery kart kredytowych w kompaktowej formie co okazuje się potrzebujemy około 2^{41} wartości, co przekłada się na 42-bitów informacji do zakodowania.
Załóżmy, że chcemy zaszyfrować numer karty kredytowej po dokonaniu transakcji w terminalu. Sposób szyfrowania musi być specyficzny. Ten numer będzie przechodził przez wiele komputerów, które go będą przetwarzały. Chcemy, żeby sam numer był ukryty, ale żeby poszczególne procesory dalej uważały dostarczone dane jako zwyczajny numer karty kredytowej. Miejscem odszyfrowania będzie nasz bank lub system VISA. Przy zastosowaniu zwyczajnego deterministycznego szyfrowania AES nie osiągamy zamierzonego celu, bo rezultatem szyfrowania są 128 bitowe bloki (16 bajtów danych) i wszystkie „procesory” przetwarzające operacje na kartach kredytowych musiałby być przeprogramowane. W naszym wypadku chcemy spowodować ew. przeprogramowanie tylko punktów końcowych: terminala i komputera w banku/systemie VISA. Postawiony problem nazywa się szyfrowaniem z zachowaniem formatu.

Szyfrowanie z zachowaniem formatu danych (ang. Format preserving Encryption <FPE>)

$S = 2^{42}$

Formalnie: mając $0 < s \leq 2^n$, trzeba zbudować PRP na $\{0,\ldots,s-1\}$

używając bezpiecznego PRF $\mathbf{F: K \times \{0,1\}^n \longrightarrow \{0,1\}^n}$ (np. AES)

Wtedy, żeby zaszyfrować numer karty:
(s = całkowita liczba kart kredytowych)

1. Mapuj dany numer karty kredytowej na $\{0,\ldots,s-1\}$
2. Zastosuj PRP aby uzyskać wyjście z zakresu $\{0,\ldots,s-1\}$
3. Mapuj wyjście do formatu numeru karty kredytowej

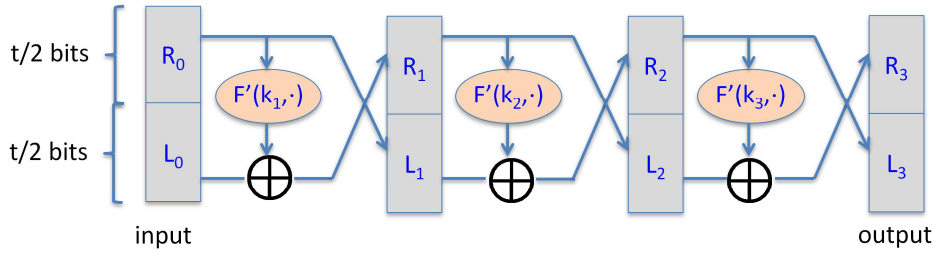
29

Formalnie, chcemy zbudować PRP na ustalonym zbiorze wartości. Zbiór wartości dla numerów kart kredytowych nie może przekroczyć 2^{42} (około). Permutacja musi się mieścić w zakresie $\{0,\ldots,s-1\}$. Do dyspozycji musimy mieć PRF (taka jak AES), ale działając na krótszym bloku danych. Jeśli to będziemy mieli, to szyfrowanie z zachowaniem formatu będzie polegało na mapowaniu numeru karty kredytowej na ciąg bitowy z zakresu 0…1-s, następnie wykonanie na tym ciągu pseudolosowej permutacji, i na koniec zamiana otrzymanej permutacji z powrotem na format numeru karty kredytowej. Trudnością tu jest zbudowanie bezpiecznego systemu szyfrowania pracującego na bardzo krótkich (z punktu widzenia alg. szyfrowania) wartościach danych.

Krok 1: od $\{0,1\}^n$ do $\{0,1\}^t$ (t<n)

Chcemy PRP na **{0,…,s-1}**. Niech t będzie takie, że $2^{t-1} < s \le 2^t$.

Metoda: Luby-Rackoff z **F': $K \times \{0,1\}^{t/2} \longrightarrow \{0,1\}^{t/2}$** (przycinanie F)



(lepiej stosować 7 rund a la Patarin, Crypto'03)

30

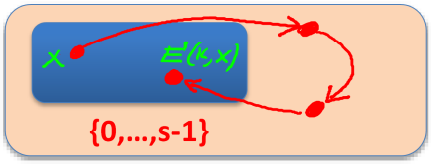
Pierwszy krok w celu rozwiązania problemu polega na opracowaniu bezpiecznej metody „skracania" rezultatu PRP. Mamy więc ciąg pseudolosowy np. o długości 128 bitów (AES), a chcemy go skrócić do t bitów, tak, żeby spełnić zależność: $2^{t-1} < s \le 2^t$. Do skrócenia może posłużyć konstrukcja Luby-Rackoff'a. Do wykonania przycinania potrzebujemy PRF, która może być wykonywana na blokach $\{0,1\}^{t/2}$. Jeśli założymy, że dla kart kredytowych t = 42 bitów, to potrzebujemy funkcji F' działającej na 21 bitach. Do skracania będziemy używać AES, w którym będziemy odcinać wszystkie bity powyżej 21. Aby otrzymać PRP o ograniczonej długości z AES tworzymy konstrukcję Luby-Rackoffa, która jest również konstrukcją Feistel'a (omawianą przy okazji algorytmu DES). Wiemy, że taka konstrukcja przekształca bezpieczną PRF z bezpieczne PRP o rozmiarze bloku dwa razy większym. Startujemy od bloki o 21 bitach, a otrzymujemy 42 – bitowy PRF. Po głębszej analizie tej konstrukcji okazało się, że lepsze parametry PRP uzyskuje się po wykonaniu 7 rund (każda z innym kluczem) a nie tylko trzech. Ostatecznie uzyskujemy PRP o długości 2^s. Teraz jesteśmy zainteresowani przekształceniem tej permutacji na zbiór (0,…,s-1)

Krok 2: od $\{0,1\}^t$ do $\{0,\ldots,s-1\}$

Mając PRP **$(E,D): K \times \{0,1\}^t \longrightarrow \{0,1\}^t$**

budujemy **$(E',D'): K \times \{0,\ldots,s-1\} \longrightarrow \{0,\ldots,s-1\}$**

E'(k, x): na wejściu jest $x \in \{0,\ldots,s-1\}$ wykonuj:
y⟵x; wykonuj { y ⟵ E(k, y) } aż do $y \in \{0,\ldots,s-1\}$; zwróć y



$\{0,1\}^t$

Spodziewana # iteracji: 2

Udowodniono bezpieczeństwo rozwiązania na CPA. Rozwiązanie nie zapewnia integralności.

31

Dysponujemy „skróconą” PRP przekształcającą $K \times \{0,1\}^t \longrightarrow \{0,1\}^t$. Chcemy otrzymać przekształcenie szyfrujące liczby z zakresu {0,…, s-1} na liczby z takiego samego zakresu {0,…,s-1}.
Wejściem naszego algorytmu jest liczba x z ustalonego przedziału. Zmiennej y przypisujemy wartość x. Następnie szyfrujemy algorytmem E z kluczem k wartość y i wynik zapisujemy do y. Jeśli wartość y mieści się w zadanym przedziale, to przerywamy obliczenia. Jeśli nie, to ponawiamy szyfrowanie y (wyniku z poprzedniej iteracji). Taki proces jest odwracalny, można więc „odszyfrować” zaszyfrowany numer karty kredytowej. Zwykle wystarczają 2 iteracje algorytmu, aby otrzymać szyfrowanie numeru karty kredytowej.

Literatura uzupełniająca

- Cryptographic Extraction and Key Derivation: The HKDF Scheme. H. Krawczyk, Crypto 2010
- Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Keywrap Problem. P. Rogaway, T. Shrimption, Eurocrypt 2006
- A Parallelizable Enciphering Mode. S. Halevi, P. Rogaway, CT-RSA 2004
- Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. P. Rogaway, Asiacrypt 2004
- How to Encipher Messages on a Small Domain: Deterministic Encryption and the Thorp Shuffle. B. Morris, P. Rogaway, T. Stegers, Crypto 2009

32

Wiadomości można uzupełnić z podanych prac. Pierwsza dotyczy technik ekstrakcji kluczy. Druga praca omawia deterministyczne szyfrowanie z uwierzytelnieniem (SIV). EME jest przedstawione w pracy trzeciej. Z kolei praca czwarta omawia szyfrowanie z dostrajaniem. Ostatnia prac dyskutuje szyfrowanie z zachowanym formatem danych.