

WPROWADZENIE DO
**NOWOCZESNY
KRYPTOGRAFIA**

Druga edycja

Jonathana Katza
Jehuda Lindell



CRC Press
Taylor & Francis Group

A CHAPMAN & HALL BOOK

WPROWADZENIE DO
**NOWOCZESNY
KRYPTOGRAFIA**
Druga edycja

Machine Translated by Google

CHAPMAN & HALL/CRC
KRYPTOGRAFIA I BEZPIECZEŃSTWO SIECI

Redaktor serii
Douglasa R. Stinsona

Opublikowane tytuły

Lidong Chen i Guang Gong, bezpieczeństwo systemu komunikacji

Shiu-Kai Chin i Susan Older, Kontrola dostępu, bezpieczeństwo i zaufanie:
Logiczne podejście

M. Jason Hinek, Kryptanaliza RSA i jej warianty

Antoine Joux, Algorytmiczna kryptoanaliza

Jonathan Katz i Yehuda Lindell, Wprowadzenie do nowoczesności
Kryptografia, wydanie drugie

Sankar K. Pal, Alfredo Petrosino i Lucia Maddalena, Podręcznik dotyczący
miękkiego przetwarzania danych w nadzorze wideo

Burton Rosenberg, Podręcznik kryptografii finansowej i bezpieczeństwa

Nadchodzące tytuły

Maria Isabel Vasco, Spyros Magliveras i Rainer Steinwandt,
Kryptografia teoretyczna grupowa

Machine Translated by Google

Chapmana i Halla/CRC
KRYPTOGRAFIA I BEZPIECZEŃSTWO SIECI

WPROWADZENIE DO
**NOWOCZESNY
KRYPTOGRAFIA**
Druga edycja

Jonathana Katza
Uniwersytet Marylandu
College Park, Maryland, USA

Jehuda Lindell
Uniwersytet Bar-Ilan
Ramat Gan, Izrael



CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group an **informa** business
A CHAPMAN & HALL BOOK

CRC Prasa

Grupa Taylora i Francisa
6000 Broken Sound Parkway NW, apartament 300
Boca Raton, Floryda 33487-2742

© 2015 by Taylor & Francis Group, LLC
CRC Press jest wydawnictwem Taylor & Francis Group, firmy informatycznej

Brak roszczeń do oryginalnych dzieł rządu USA
Data wersji: 20140915

Miejsz dzynarodowy Standard Book Number-13: 978-1-4665-7027-6 (eBook - PDF)

Książka ta zawiera informacje uzyskane z autentycznych i cenionych źródeł. Dołożono wszelkich starań, aby opublikować wiarygodne dane i informacje, jednak autor i wydawca nie mogą ponosić odpowiedzialności za ważność wszystkich materiałów ani konsekwencje ich wykorzystania. Autorzy i wydawcy podjęli próbę odnalezienia właścicieli praw autorskich do wszystkich materiałów reprodukowanych w tej publikacji i przepraszają właścicieli praw autorskich, jeśli nie uzyskali zgody na publikację w tej formie. Jeśli jakikolwiek materiał objęty prawami autorskimi nie został uznany, napisz i daj nam znać, abyśmy mogli poprawić go w przyszłym przedruk.

Z wyjątkiem przypadków dozwolonych na mocy amerykańskiego prawa autorskiego, żadna część tej książki nie może być przedrukowywana, powielana, przesyłana ani wykorzystywana w jakikolwiek formie za pomocą jakichkolwiek środków elektronicznych, mechanicznych lub innych, znanych obecnie lub wynalezionych w przyszłości, w tym poprzez fotokopiowanie, mikrofilmowanie i nagrywanie, lub w jakikolwiek systemie przechowywania lub wyszukiwania informacji, bez pisemnej zgody wydawców.

Aby uzyskać pozwolenie na kopiowanie lub wykorzystywanie materiałów z tej pracy w formie elektronicznej, należy wejść na stronę www.copy-right.com (<http://www.copy-right.com/>) lub skontaktować się z Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC jest organizacją non-profit, która zapewnia licencje i rejestrację różnym użytkownikom. Dla organizacji, które uzyskały licencję na kserokopię od CCC, ustalono odrębny system płatności.

Uwaga dotycząca znaków towarowych: Nazwy produktów lub firm mogą być znakami towarowymi lub zastrzeżonymi znakami towarowymi i są używane wyłącznie w celu identyfikacji i wyjaśnienia, bez zamiaru naruszenia.

Odwiedź witrynę internetową firmy Taylor & Francis pod adresem
<http://www.taylorandfrancis.com>

oraz witryna internetowa CRC Press pod adresem
<http://www.crcpress.com>

Zawartość

Przedmowa

xv

I Wprowadzenie i kryptografia klasyczna

1. Wstęp	3
1.1 Kryptografia i współczesna kryptografia.	3
1.2 Ustawianie szyfrowania klucza prywatnego .	4
1.3 Szyfry historyczne i ich kryptoanaliza.	8
1.4 Zasady współczesnej kryptografii.	16
1.4.1 Zasada 1 – Definicje formalne .	17
1.4.2 Zasada 2 – Precyzyjne założenia .	20
1.4.3 Zasada 3 – Dowody zabezpieczenia .	22
1.4.4 Bezpieczeństwo możliwe do udowodnienia i bezpieczeństwo w świecie rzeczywistym .	22
Referencje i dodatkowe lektury.	23
Ćwiczenia .	24
 2 Szyfrowanie doskonale tajne	25
2.1 Definicje .	25
2.2 Jednorazowa podkładka .	26
2.3 Ograniczenia doskonałej tajemnicy .	32
2.4 *Twierdzenie Shannona .	35
Referencje i dodatkowe lektury.	36
Ćwiczenia .	37
 38	38

II Kryptografia klucza prywatnego (symetrycznego).

3 Szyfrowanie kluczem prywatnym	43
3.1 Bezpieczeństwo obliczeniowe .	43
3.1.1 Konkretnie podejście .	44
3.1.2 Podejście asymptotyczne .	45
3.2 Definiowanie bezpiecznego szyfrowania obliczeniowego .	52
3.2.1 Podstawowa definicja bezpieczeństwa .	53
3.2.2 *Bezpieczeństwo semantyczne .	56
3.3 Konstruowanie schematów bezpiecznego szyfrowania .	60
3.3.1 Generatory pseudolosowe i szyfry strumieniowe .	60
3.3.2 Dowody redukcyjne .	65
3.3.3 Bezpieczny schemat szyfrowania o stałej długości.	66

3.4 Wzmocnione koncepcje bezpieczeństwa	71
3.4.1 Bezpieczeństwo przy wielokrotnym szyfrowaniu	71
3.4.2 Ataki wybranym tekstem jawnym i bezpieczeństwo CPA	73
3.5 Konstruowanie schematów szyfrowania z zabezpieczeniem CPA	77
3.5.1 Funkcje pseudolosowe i szyfry blokowe	77
3.5.2 Szyfrowanie bezpieczne CPA z funkcji pseudolosowych	82
3.6 Tryby pracy	86
3.6.1 Tryby działania szyfru strumieniowego	86
3.6.2 Tryby działania szyfru blokowego	88
3.7 Ataki za pomocą wybranego tekstu zaszyfrowanego	96
3.7.1 Definicja bezpieczeństwa CCA	96
3.7.2 Ataki dopełniające-Oracle	98
Referencje i dodatkowe lektury	101
Ćwiczenia	102
 4 Kody uwierzytelniające wiadomości 4.1	107
Integralność wiadomości	107
4.1.1 Tajemnica a uczciwość	107
4.1.2 Szyfrowanie a uwierzytelnianie wiadomości	108
4.2 Kody uwierzytelniające wiadomości – definicje	110
4.3 Konstruowanie bezpiecznych kodów uwierzytelniających wiadomości	116
4.3.1 MAC o stałej długości	116
4.3.2 Rozszerzenie domeny dla komputerów MAC	118
4.4 CBC-MAC	122
4.4.1 Podstawowa konstrukcja	123
4.4.2 *Dowód bezpieczeństwa	125
4.5 Uwierzytelnione szyfrowanie	131
4.5.1 Definicje	131
4.5.2 Konstrukcje ogólne	132
4.5.3 Sesje bezpiecznej komunikacji	140
4.5.4 Szyfrowanie bezpieczne CCA	141
4.6 *Mac z teorii informacji	142
4.6.1 Konstruowanie MAC w teorii informacyjnej	143
4.6.2 Ograniczenia adresów MAC z teorii informacyjnej	145
Referencje i dodatkowe lektury	146
Ćwiczenia	147
 5 funkcji skrótu i zastosowań	153
5.1 Definicje	153
5.1.1 Odporność na kolizje	154
5.1.2 Ślabsze pojęcia bezpieczeństwa	156
5.2 Rozszerzenie domeny: transformacja Merkle'a-Damgård'a	156
5.3 Uwierzytelnianie wiadomości za pomocą funkcji skrótu	158
5.3.1 Hash i MAC	159
5.3.2 HMAC	161

5.4 Ogólne ataki na funkcje mieszające	164
5.4.1 Ataki urodzinowe w celu znalezienia kolizji	164
5.4.2 Ataki urodzinowe na małej przestrzeni	166
5.4.3 *Kompromisy czasu i przestrzeni dla funkcji odwracających	168
5.5 Model Random-Oracle	174
5.5.1 Szczegóły modelu Random-Oracle	175
5.5.2 Czy metodologia Random-Oracle jest rozsądna?	179
5.6 Dodatkowe zastosowania funkcji skrótu	182
5.6.1 Odciski palców i deduplikacja	182
5.6.2 Drzewa Merkle'a	183
5.6.3 Mieszanie hasła	184
5.6.4 Wyprowadzenie klucza	186
5.6.5 Schematy zobowiązania	187
Referencje i dodatkowe lektury.	189
Ćwiczenia	189
 6 Praktyczne konstrukcje prymitywów z kluczem symetrycznym	193
6.1 Szyfry strumieniowe	194
6.1.1 Rejestry przesunięcia ze sprzężeniem zwrotnym liniowym	195
6.1.2 Dodawanie nieliniowości	197
6.1.3 Tryrium	198
6.1.4 RC4	199
6.2 Szyfry blokowe	202
6.2.1 Sieci podstawień i permutacji	204
6.2.2 Sieci Feistela	211
6.2.3 DES – Standard szyfrowania danych	212
6.2.4 3DES: Zwiększenie długości klucza szyfru blokowego .	220
6.2.5 AES – zaawansowany standard szyfrowania .	223
6.2.6 *Kryptoanaliza różnicowa i liniowa	225
6.3 Funkcje mieszające	231
6.3.1 Funkcje skrótu z szyfrów blokowych	232
6.3.2 MD5	234
6.3.3 SHA-0, SHA-1 i SHA-2	234
6.3.4 SHA-3 (Keccak)	235
Referencje i dodatkowe lektury.	236
Ćwiczenia	237
 7 *Teoretyczne konstrukcje prymitywów o kluczu symetrycznym	241
7.1 Funkcje jednokierunkowe	242
7.1.1 Definicje	242
7.1.2 Kandydackie funkcje jednokierunkowe	245
7.1.3 Predykaty podstawowe	246
7.2 Od funkcji jednokierunkowych do pseudolosowości	248
7.3 Predykaty twardze z funkcji jednokierunkowych	250
7.3.1 Prosty przypadek	250

7.3.2 Bardziej zawiły przypadek ..	251	· · · · ·
7.3.3 Pełny dowód ..	254	· · · · ·
7.4 Konstruowanie generatorów pseudolosowych ..	257	· · · · ·
7.4.1 Generatorzy pseudolosowe z minimalnym rozszerzeniem ..	258	· · · · ·
7.4.2 Zwiększanie współczynnika rozszerzenia ..	259	· · · · ·
7.5 Konstruowanie funkcji pseudolosowych ..	265	· · · · ·
7.6 Konstruowanie (silnych) permutacji pseudolosowych ..	269	· · · · ·
7.7 Założenia dotyczące kryptografii klucza prywatnego ..	273	· · · · ·
7.8 Nierozróżnialność obliczeniowa ..	276	· · · · ·
Referencje i dodatkowe lektury ..	278	· · · · ·
Ćwiczenia ..	279	· · · · ·

III Kryptografia klucza publicznego (asymetryczna).

8 Teoria liczb i założenia dotyczące twardości kryptograficznej	285	
8.1 Wstęp, podstawowa teoria grup ..	287	· · · · ·
8.1.1 Liczby pierwsze i podzielność ..	287	· · · · ·
8.1.2 Arytmetyka modułowa ..	289	· · · · ·
8.1.3 Grupy ..	291	· · · · ·
8.1.4 Grupa Z_N ..	295	· · · · ·
8.1.5 *Izomorfizmy i chińskie twierdzenie o resztach ..	297	
8.2 Liczby pierwsze, faktoring i RSA ..	302	
8.2.1 Generowanie losowych liczb pierwszych ..	303	
8.2.2 *Testowanie pierwszości ..	306	
8.2.3 Założenia Faktoringu ..	311	
8.2.4 Założenie RSA ..	312	
8.2.5 *Powiązanie założeń RSA i faktoringu ..	314	
8.3 Założenia kryptograficzne w grupach cyklicznych ..	316	
8.3.1 Grupy cykliczne i generatory ..	316	· · · · ·
8.3.2 Logarytm dyskretny/Założenia Diffiego–Hellmana	319	
8.3.3 Praca w ($\text{podgrupach } Z_p$) ..	322	
8.3.4 Krzywe eliptyczne ..	325	
8.4 *Aplikacje kryptograficzne ..	332	
8.4.1 Funkcje jednokierunkowe i permutacje ..	332	
8.4.2 Konstruowanie funkcji skrótu odpornych na kolizje ..	335	
Referencje i dodatkowe lektury ..	337	
Ćwiczenia ..	338	
9 *Algorytmy faktoringu i obliczania logarytmów dyskretnych – rytm 9.1	341	
Algorytmy faktoringu ..	342	
9.1.1 Algorytm p-1 Pollarda ..	343	
9.1.2 Algorytm Rho Pollarda ..	344	
9.1.3 Algorytm sita kwadratowego ..	345	
9.2 Algorytmy obliczania logarytmów dyskretnych ..	348	

9.2.1 Algorytm Pohliga-Hellmana	350
9.2.2 Algorytm Baby-Step/Giant-Step	352
9.2.3 Logarytmy dyskretnie ze zderzeń	353
9.2.4 Algorytm rachunku indeksowego	354
9.3 Zalecane długości kluczy	356
Referencje i dodatkowe lektury.	357
Ćwiczenia	358
 10 Zarządzanie kluczami i rewolucja w zakresie klucza publicznego 10.1	359
Dystrybucja kluczy i zarządzanie kluczami	359
10.2 Rozwiązywanie czę ściowe: Centra dystrybucji kluczy	361
10.3 Wymiana kluczy i protokół Diffiego-Hellmana	363
10.4 Rewolucja klucza publicznego.	370
Referencje i dodatkowe lektury.	372
Ćwiczenia	373
 11 Szyfrowanie kluczem publicznym	375
11.1 Szyfrowanie kluczem publicznym – przegląd	375
11.2 Definicje	378
11.2.1 Zabezpieczenie przed atakami w postaci wybranego tekstu jawnego.	379
11.2.2 Wielokrotne szyfrowanie	381
11.2.3 Zabezpieczenie przed atakami z wykorzystaniem wybranego tekstu zaszyfrowanego	387
11.3 Szyfrowanie hybrydowe i paradygmat KEM/DEM	389
11.3.1 Bezpieczeństwo CPA	393
11.3.2 Bezpieczeństwo CCA	398
11.4 Szyfrowanie oparte na CDH/DDH	399
11.4.1 Szyfrowanie El Gamala	400
11.4.2 Hermetyzacja kluczy w oparciu o DDH	404
11.4.3 *KEM oparty na CDH w modelu Random-Oracle	406
11.4.4 Bezpieczeństwo wybranego tekstu zaszyfrowanego i DHIES/ECIES	408
11.5 Szyfrowanie RSA	410
11.5.1 Zwykły RSA	410
11.5.2 Wyściece RSA i PKCS #1 v1.5	415
11.5.3 * Szyfrowanie zabezpieczone CPA bez losowych Oracle	417
11.5.4 OAEP i RSA PKCS #1 v2.0	421
11.5.5 *KEM bezpieczny dla CCA w modelu Random-Oracle	425
11.5.6 Problemy i pułapki związane z wdrażaniem RSA	429
Referencje i dodatkowe lektury.	432
Ćwiczenia	433
 12 Schematy podpisu cyfrowego 12.1	439
Podpisy cyfrowe – przegląd	439
12.2 Definicje	441
12.3 Paradygmat hash-and-sign	443
12.4 Podpisy RSA	444

12.4.1 Zwykły RSA	444
12.4.2 RSA-FDH i PKCS #1 v2.1	446
12.5 Podpisy z problemu logarytmu dyskretnego	451
12.5.1 Schemat podpisu Schnorra	451
12.5.2 DSA i ECDSA	459
12.6 *Podpisy z funkcji skrótu	461
12.6.1 Schemat podpisu Lamporta	461
12.6.2 Podpisy łańcuchowe	465
12.6.3 Podpisy oparte na drzewie	468
12.7 *Certyfikaty i infrastruktura klucza publicznego	473
12.8 Łączenie wszystkiego – SSL/TLS	479
12.9 *Szyfrowanie podpisu	481
Referencje i dodatkowe lektury.	483
Ćwiczenia	484
 13 *Zaawansowane tematy dotyczące szyfrowania klucza publicznego	487
13.1 Szyfrowanie kluczem publicznym z permutacji zapadni	487
13.1.1 Permutacje zapadni	488
13.1.2 Szyfrowanie kluczem publicznym z permutacji zapadni .	489
13.2 Schemat szyfrowania Paillier	491
13.2.1 Struktura Z N2	492
13.2.2 Schemat szyfrowania Paillier	494
13.2.3 Szyfrowanie homomorficzne	499
13.3 Udostępnianie sekretów i szyfrowanie progowe	501
13.3.1 Udostępnianie sekretu	501
13.3.2 Sprawdzalne udostępnianie sekretów	503
13.3.3 Szyfrowanie progowe i głosowanie elektroniczne	505
13.4 Schemat szyfrowania Goldwassera-Micali	507
13.4.1 Reszty kwadratowe Modulo a Prime	507
13.4.2 Reszty kwadratowe Modulo a Composite	510
13.4.3 Założenie resztowości kwadratowej	514
13.4.4 Schemat szyfrowania Goldwassera-Micali	515
13.5 Schemat szyfrowania Rabina	518
13.5.1 Obliczanie modułowych pierwiastków kwadratowych	518
13.5.2 Permutacja zapadni oparta na faktoringu	523
13.5.3 Schemat szyfrowania Rabina	527
Referencje i dodatkowe lektury.	528
Ćwiczenia	529
 Indeks wspólnej notacji	533

Dodatek A Podstawy matematyczne A.1 Tożsamości i nierówności.	537
A.2 Notacja asymptotyczna .	537
A.3 Podstawowe prawdopodobieństwo .	538
A.4 Problem „urodzin” .	542
A.5 *Pola skończone .	544
Dodatek B Podstawowa algorytmiczna teoria liczb	547
B.1 Arytmetyka liczb całkowitych .	549
B.1.1 Podstawowe operacje .	549
B.1.2 Algorytmy euklidesowe i rozszerzone algorytmy euklidesowe .	550
B.2 Arytmetyka modułowa .	552
B.2.1 Podstawowe operacje .	552
B.2.2 Obliczanie odwrotności modułowych .	552
B.2.3 Potęgowanie modułowe .	553
B.2.4 *Mnożenie Montgomery'ego .	556
B.2.5 Wybór jednolitego elementu grupy .	557
B.3 *Wyszukiwanie generatora grupy cyklicznej.	559
B.3.1 Podstawy teorii grup.	559
B.3.2 Efektywne algorytmy .	561
Referencje i dodatkowe lektury.	562
Ćwiczenia .	562
Bibliografia	563
Indeks	577

Machine Translated by Google

Przedmowa

Cel naszej książki pozostaje ten sam, co w pierwszym wydaniu: przedstawienie podstawowych paradygmatów i zasad współczesnej kryptografii szerokiemu gronu odbiorców posiadających podstawową wiedzę matematyczną. Zaprojektowaliśmy tę książkę tak, aby służyła jako podręcznik do kursów licencjackich i magisterskich z zakresu kryptografii (na wydziałach informatyki, elektrotechniki lub matematyki), jako ogólne wprowadzenie odpowiednie do samodzielnej nauki (szczególnie dla początkujących doktorantów), oraz jako punkt odniesienia dla studentów, badaczy i praktyków.

Obecnie dostępnych jest wiele innych podręczników do kryptografii i czytelnik może słusznie zadać sobie pytanie, czy potrzebna jest kolejna książka na ten temat. Nie napisalibyśmy tej książki – ani nie pracowali nad jej poprawieniem do drugiego wydania – gdyby odpowiedź na to pytanie była inna niż jednoznaczne „tak”. To, co naszym zdaniem odróżnia naszą książkę od innych dostępnych książek, to rygorystyczne omówienie współczesnej kryptografii w przystępny sposób, odpowiedni dla wprowadzenia w temat.

Skupiamy się na nowoczesnej kryptografii (po latach 80. XX wieku), która różni się od kryptografii klasycznej naciskiem na definicje, precyzyjne założenia i rygorystyczne dowody bezpieczeństwa. Pokrótko omówimy każdą z nich po kolei (zasady te zostaną omówione bardziej szczegółowo w rozdziale 1):

- Centralna rola definicji: Kluczowym intelektualnym wkładem współczesnej kryptografii było uznanie, że formalne definicje bezpieczeństwa stanowią istotny pierwszy krok w projektowaniu dowolnego algorytmu lub protokołu kryptograficznego. Z perspektywy czasu powód jest prosty: jeśli nie wiesz, co chcesz osiągnąć, jak możesz mieć nadzieję, że dowiesz się, kiedy to osiągniesz? Jak zobaczymy w tej książce, kryptograficzne definicje bezpieczeństwa są dość mocne i – na pierwszy rzut oka – mogą wydawać się niemożliwe do osiągnięcia. Jednym z najbardziej zdumiewających aspektów kryptografii jest to, że można udowodnić istnienie wydajnych konstrukcji spełniających tak mocne definicje (przy raczej łagodnych założeniach).
- Znaczenie dokładnych założeń: Jak zostało wyjaśnione w rozdziałach 2 i 3, obecnie nie można udowodnić bezpieczeństwa wielu konstrukcji kryptograficznych w sensie bezwarunkowym. Zamiast tego bezpieczeństwo często opiera się na powszechnie uznawanych (choć niepotwierdzonych) założeniach. Współczesne podejście kryptograficzne wymaga, aby każde takie założenie zostało jasno określone i jednoznacznie zdefiniowane. Pozwala to nie tylko na obiektywną ocenę założeń, ale, co ważniejsze, umożliwia rygorystyczne dowody bezpieczeństwa, jak opisano poniżej.

- Możliwość dowodów bezpieczeństwa: Dwie poprzednie zasady służą jako podstawa poglądu, że konstrukcje kryptograficzne można udowodnić jako bezpieczne w odniesieniu do jasno określonych definicji bezpieczeństwa i w odniesieniu do dobrze zdefiniowanych założeń kryptograficznych. Koncepcja ta stanowi esencję współczesnej kryptografii i to ona przekształciła tę dziedzinę ze sztuki w naukę .

Nie da się przecenić wagi tej idei. Historycznie rzecz biorąc, schematy kryptograficzne były projektowane w dużej mierze ad hoc i uznawano je za bezpieczne, jeśli sami projektanci nie mogli znaleźć żadnych ataków. Z kolei współczesna kryptografia zaleca projektowanie schematów z formalnymi, matematycznymi dowodami bezpieczeństwa w dobrze zdefiniowanych modelach. Takie schematy gwarantują bezpieczeństwo, chyba że leżące u ich podstaw założenie jest fałszywe (lub definicja bezpieczeństwa nie odzwierciedla odpowiednio rzeczywistych problemów związanych z bezpieczeństwem). Opierając się na utrwalonych założeniach (np. założeniu, że „faktoring jest trudny”), można zatem uzyskać schematy, których złamania jest niezwykle mało prawdopodobne.

Jednolite podejście. Powyższe zasady współczesnej kryptografii dotyczą nie tylko społeczności „teorii kryptografii”. Znaczenie precyzyjnych definicji jest obecnie szeroko rozumiane i doceniane przez programistów i inżynierów bezpieczeństwa, którzy wykorzystują narzędzia kryptograficzne do budowania bezpiecznych systemów, a rygorystyczne dowody bezpieczeństwa stały się jednym z wymogów standaryzacji schematów kryptograficznych.

Zmiany w wydaniu drugim

Przygotowując drugą edycję , podjęliśmy świadomego wysiłek, aby uwzględniliśmy bardziej praktyczną perspektywę (bez rezygnacji z rygorystycznego podejścia). Znajduje to odzwierciedlenie w szeregu zmian i uzupełnień, które wprowadziliśmy:

- Zwiększyliśmy zakres szyfrów strumieniowych, wprowadzając je jako wariant generatorów pseudolosowych w sekcji 3.3.1, omawiając tryby działania szyfru strumieniowego w sekcji 3.6.1 i opisując nowoczesne zasady i przykłady projektowania szyfrów strumieniowych w sekcji Sekcja 6.1.
- Podkreślimy znaczenie uwierzytelnionego szyfrowania (patrz sekcja 4.5) i dodaliśmy sekcję dotyczącą bezpiecznych sesji komunikacyjnych.
- Przenieśliśmy opis funkcji skrótu do osobnego rozdziału (rozdział 5), uwzględniliśmy kilka standardowych zastosowań kryptograficznych funkcji skrótu (rozdział 5.6) i dodaliśmy sekcję dotyczącą zasad projektowania funkcji skrótu i szeroko stosowanych konstrukcji (rozdział 6.3). Poprawiliśmy także sposób traktowania ataków urodzinowych (omawiając ataki urodzinowe na małą przestrzeń w sekcji 5.4.2) i dodaliśmy omówienie tą czwórką tabel oraz kompromisów w czasie i przestrzeni (sekcja 5.4.3).

- Uwzględniliśmy kilka ważnych ataków na implementacje kryptografii, które pojawiają się w praktyce, w tym ataki wybranym tekstem jawnym na szyfrowanie łańcuchowe CBC (sekcja 3.6.2), ataki typu padding-oracle na szyfrowanie w trybie CBC (sekcja 3.7.2) oraz ataki czasowe na weryfikację MAC (sekcja 4.2).
- Po długich naradach zdecydowaliśmy się wprowadzić model losowej wyroczni znacznie wcześniej w książce (rozdział 5.5). Pozwala nam to na właściwe, zintegrowane potraktowanie standardowych, szeroko stosowanych schematów szyfrowania i podpisów z użyciem klucza publicznego w późniejszych rozdziałach, zamiast spychania ich do rangi drugiej kategorii w rozdziale na końcu książki.
- Wzmocniliśmy naszą ofertę dotyczącą kryptografii krzywej eliptycznej (sekcja 8.3.4) i dodaliśmy dyskusję na temat jej wpływu na zalecane długości kluczy (sekcja 9.3).
- W rozdziale poświęconym szyfrowaniu klucza publicznego przedstawiamy paradygmat KEM/DEM jako formę szyfrowania hybrydowego (patrz rozdział 11.3). Oprócz standardów RSA PKCS #1 zajmujemy się również DHIES/ECIES.
- W rozdziale poświęconym podpisom cyfrowym opisujemy teraz konstrukcję podpisów ze schematów identyfikacyjnych z wykorzystaniem transformaty Fiat-Shamira, na prototypowym przykładzie schematu podpisu Schnorra. Poprawiliśmy także zakres naszych usług DSA/ECDSA. Zamieszczamy krótkie omówienie protokołu SSL/TLS i szyfrowania znaków, które stanowią zwieńczenie wszystkiego, co zostało omówione do tego momentu.
- W rozdziale „Tematy zaawansowane” rozszerzyliśmy podejście do szyfrowania homomorficznego i dodaliśmy sekcje dotyczące udostępniania sekretów i szyfrowania progowego.

Oprócz powyższego zredagowaliśmy także całą książkę, wprowadzając obszerne poprawki, a także mniejsze poprawki, w tym wiele opracowanych przykładów, w celu ulepszenia ekspozycji. Dodano także kilka dodatkowych ćwiczeń.

Przewodnik po korzystaniu z tej książki

Ta część jest przeznaczona przede wszystkim dla instruktorów, którzy chcą zastosować tę książkę w swoich kursach, chociaż student samodzielnie się gający po tej książce może również uznać ją za przydatny przegląd.

Wymagane tło. Stworzyliśmy tę książkę w taki sposób, że jedynym formalnym warunkiem wstępny jest kurs matematyki dyskretnej. Nawet tutaj opieramy się na bardzo małej ilości materiału: zakładamy znajomość podstawowego (dyskretnego) prawdopodobieństwa i arytmetyki modułowej. Oczekuje się również, że uczniowie czytający tę książkę mieli pewien kontakt z algorytmami, głównie po to, aby móc swobodnie czytać pseudokod i znać notację dużego O. Wiele z tych koncepcji omówiono w Dodatku A i/lub w momencie ich pierwszego użycia w książce.

XVIII

Niezależnie od powyższego, książka zawiera definicje, dowody i abstrakcyjne pojęcia matematyczne, dlatego wymaga pewnej dojrzałości matematycznej. W szczególności zakłada się, że czytelnik miał kontakt z dowodami na poziomie uczelni, czy to na kursie matematyki wyższego poziomu, czy na kursie matematyki dyskretnej, algorytmów lub teorii obliczalności.

Sugestie dotyczące organizacji kursu. Podstawowy materiał tej książki, który zalecamy uwzględniać dnia w każdym kursie wprowadzającym do kryptografii, składa się z następujących elementów (w wszystkich przypadkach wykluczone są sekcje oznaczone gwiazdką; więcej na ten temat poniżej):

- Wprowadzenie i kryptografia klasyczna: Rozdziały 1 i 2 omawiają kryptografię klasyczną i przygotowują grunt pod współczesną kryptografię.
- Kryptografia klucza prywatnego (symetryczna): Rozdział 3 o szyfrowaniu klucza prywatnego, Rozdział 4 o uwierzytelnianiu wiadomości i Rozdział 5 o funkcjach skrótu zapewniają dokładne omówienie tych tematów.
Zdecydowanie zalecamy również omówienie sekcji 6.2, która dotyczy projektowania szyfrów blokowych; Z naszego doświadczenia wynika, że uczniowie naprawdę lubią ten materiał, dzięki czemu abstrakcyjne idee, których nauczyli się w poprzednich rozdziałach, stają się bardziej konkretne. Choć bierzemy pod uwagę ten podstawowy materiał, nie jest on używany w pozostałej części książki, więc w razie potrzeby można go bezpiecznie pominąć.
- Kryptografia klucza publicznego (asymetryczna): Rozdział 8 stanowi samodzielne wprowadzenie do całej teorii liczb potrzebnej w pozostałej części książki. Materiał zawarty w rozdziale 9 nie jest później wykorzystywany; zalecamy jednak przynajmniej omówienie sekcji 9.3 na temat zalecanych długości kluczy. Rewolucję klucza publicznego opisano w rozdziale 10. Najlepiej byłoby, gdyby omówiono wszystkie rozdziały 11 i 12; ci, którym zależy na czasie, mogą wybierać właściwie.

Zazwyczaj jesteśmy w stanie omówić więcej niż powyższych zagadnień w ramach semestralnego (35-godzinnego) kursu licencjackiego (z pominięciem niektórych dowodów i pominięciem niektórych tematów, jeśli to konieczne) lub, z pewnymi zmianami polegającymi na dodaniu więcej ilości materiału z podstaw teoretycznych, w pierwsze trzy czwarte semestralnego kursu magisterskiego. Instruktorzy dysponujący więcej ilością czasu mogą kontynuować zajęcia w wolniejszym tempie lub uwzględniać dodatkowe tematy, jak omówiono poniżej.

Ci, którzy chcą omówić dodatkowy materiał w ramach dłuższego kursu lub szybszego kursu dla absolwentów, przekonają się, że struktura książki pozwala na elastyczne włączenie innych tematów, jeśli pozwala na to czas (i w zależności od zainteresowań instruktora). W szczególności sekcje i rozdziały oznaczone gwiazdką (*) można omówić w dowolnej kolejności lub całkowicie pominąć, bez wpływu na ogólny tok książki. Zadbaliśmy o to, aby żaden materiał rdzenia nie był zależny od żadnego materiału oznaczonego gwiazdką i w większości sekcje oznaczone gwiazdką nie były od siebie zależne. (Kiedy tak się dzieje, zależność ta jest wyraźnie odnotowana.)

Dla tych, którzy chcą nadać swojemu kursowi szczególny charakter, spośród tematów oznaczonych gwiazdką proponujemy następujące:

- Teoria: Kurs o bardziej teoretycznym charakterze mógłby obejmować materiał z Sekcji 3.2.2 (bezpieczeństwo semantyczne); Rozdział 7 (Funkcje jednokierunkowe i predykaty twardie oraz konstruowanie generatorów, funkcji i permutacji pseudolosowych na podstawie permutacji jednokierunkowych); Sekcja 8.4 (funkcje jednokierunkowe i odporne na kolizje funkcje mieszające na podstawie założeń teorii liczb); Sekcja 11.5.3 (szifrowanie RSA bez losowych wyciągów); i Sekcja 12.6 (podpisy bez przypadkowych wyciągów).
- Matematyka: Kurs skierowany do studentów z solidną wiedzą matematyczną – lub prowadzony przez kogoś, kto lubi ten aspekt kryptografii – mógłby obejmować sekcję 4.6 (MAC z teorii informacji w polach skończonych); niektóre z bardziej zaawansowanych teoriów liczb z rozdziału 8 (np. chińskie twierdzenie o resztach i test pierwszości Millera-Rabina); i cały rozdział 9.

W obu przypadkach można również uwzględnić wybór zaawansowanych tematów z rozdziału 13.

Informacje zwrotne i errata

Naszym celem przy pisaniu tej książki było udostępnienie współczesnej kryptografii szerokiemu gronu odbiorców spoza społeczności „informatyki teoretycznej”. Mamy nadzieję, że poinformujesz nas, czy nam się udało. Wiele entuzjastycznych e-maili, które otrzymaliśmy w odpowiedzi na nasze pierwsze wydanie, sprawiło, że cały proces pisania tej książki był wartościowy.

Zawsze chętnie otrzymujemy informację zwrotną. Mamy nadzieję, że w książce nie ma błędów ani literówek; jeśli jednak jakieś znajdziesz, będziemy bardzo wdzięczni, jeśli nas o tym poinformujesz. (Lista znanych errat będzie dostępna pod adresem <http://www.cs.umd.edu/~jkatz/imc.html>.) Swoje komentarze i erraty możesz przesyłać e-mailem na adres jkatz@cs.umd.edu i lindell@biu.ac.il; w tytule proszę wpisać „Wprowadzenie do współczesnej kryptografii”.

Podziękowania

Za drugie wydanie: Jesteśmy wdzięczni wielu czytelnikom pierwszego wydania, którzy przesłali nam komentarze, sugestie i poprawki, które pomogły w znacznym udoskonaleniu książki. Szczególnie owocne były dyskusje z Claude'em Cr'peau, Billem Gasarchem, Gene Itkisem, Leonidem Reyzinem, Tomem Shrimptonem i Salilem Vadhanem na temat treści i ogólnej „filozofii” książki. Dziękujemy także Bar Alonowi, Giladowi Asharovowi, Giuseppe Ateniese, Amirowi Azodiemu, Omerowi Berkmanowi, Sergio de Biasi, Aurorze Bristor, Richardowi Changowi, Qingfeng Chengowi, Kwan Tae Cho, Kyliah Clarkson, Ran Cohen, Nikolas Coukouma, Dana Dachman-Soled, Michael Fang, Michael Farcasin, Pooya Farshim, Marc Fischlin, Lance

Fortnow, Michael Fuhr, Bill Gasarch, Virgil Gligor, Carmit Hazay, Andreas Hübner, Karst Koymans, Eyal Kushilevitz, Steve Lai, Ugo Dal Lago, Armand Makowski, Tal Malkin, Steve Myers, Naveen Nathan, Ariel Nof, Eran Omri, Ruy de Queiroz, Eli Quiroz, Tal Rabin, Charlie Rackoff, Yona Raekow, Tzachy Reinman, Wei Ren, Ben Riva, Volker Roth, Christian Schaffner, Joachim Schipper, Dominique Schröder, Randy Shull, Nigel Smart, Christoph Sprenger, Aravind Srinivasan, John Steinberger, Aishwarya Thiruvengadam, Dave Tuller, Poorvi Vora, Avishai Yanai, Rupeng Yang, Arkady Yerukhi-movich, Dae Hyun Yum, Hila Zarosim i Konstantin Ziegler za pomocne poprawki do pierwszego wydania i/lub wcześniejszych projekty drugiego wydania.

Za pierwsze wydanie: Dziękujemy Zoe Bermant za wykonanie figurek; Davidowi Wagnerowi za udzielenie odpowiedzi na pytania dotyczące szyfrów blokowych i ich kryptoanalizy; oraz Salilowi Vadhanowi i Alonowi Rosenowi za eksperymentowanie z wczesną wersją naszego tekstu podczas kursu wprowadzającego na Uniwersytecie Harverda oraz za przekazanie nam cennych informacji zwrotnych. Chcielibyśmy także wyrazić naszą wdzięczność tym, którzy przeczytali i skomentowali wcześniejsze wersje tej książki, a także tym, którzy przesłali nam poprawki: Adamowi Benderowi, Chiu-Yuen Koo, Yairowi Domboowi, Michaelowi Fuhrowi, Williamowi Glennowi, S. Dovowi Gordonowi, Carmit Hazay, Eyal Kushilevitz, Avivit Levy, Matthew Mah, Ryan Murphy, Steve Myers, Martin Paraskevov, Eli Quiroz, Jason Rogers, Rui Xue, Dicky Yan, Arkady Yerukhimovich i Hila Zarosim. Jesteśmy niezmiernie wdzięczni wszystkim, którzy zachęcali nas do napisania tej książki i zgodzili się z nami, że taka książka jest bardzo potrzebna.

Na koniec dziękujemy naszym żonom i dzieciom za całe ich wsparcie i zrozumienie podczas wielu godzin, dni, miesięcy cy, a teraz lat, które spędziliśmy nad tym projektem.

Część I

Wprowadzenie i klasyka

Kryptografia

Machine Translated by Google

Rozdział 1

Wstęp

1.1 Kryptografia i współczesna kryptografia

Concise Oxford English Dictionary definiuje kryptografię jako „sztukę pisania lub rozwijywania kodów”. Jest to zgodne z prawdą historyczną, ale nie oddaje aktualnego zakresu tej dziedziny ani jej współczesnych podstaw naukowych. Definicja skupia się wyłącznie na kodach używanych od stuleci w celu umożliwienia tajnej komunikacji. Jednak obecnie kryptografia obejmuje znacznie więcej: zajmuje się mechanizmami zapewniającymi integralność, technikami wymiany tajnych kluczy, protokołami uwierzytelniania użytkowników, aukcjami i wyborami elektronicznymi, cyfrową gotówką i nie tylko. Nie próbując przedstawić pełnej charakterystyki, powiedzielibyśmy, że współczesna kryptografia obejmuje badanie technik matematycznych służących do zabezpieczania informacji cyfrowych, systemów i obliczeń rozproszonych przed atakami kontradyktoryjnymi.

Definicja słownikowa odnosi się także do kryptografii jako sztuki. Do końca XX wieku kryptografia była rzeczywiście w dużej mierze sztuką. Konstruowanie dobrych kodów lub łamanie istniejących opierało się na kreatywności i rozwiniętym wyczuciu działania kodów. Niewiele było teorii, na których można było oprzeć, i przez długi czas nie istniała działająca definicja tego, co stanowi dobry kod. Począwszy od lat 70. i 80. XX w. obraz kryptografii radykalnie się zmienił. Zaczęła wyłaniać się bogata teoria, umożliwiająca rygorystyczne badanie kryptografii jako nauki i dyscypliny matematycznej. Ta perspektywa z kolei wpłynęła na sposób, w jaki badacze myślą o szerszej dziedzinie bezpieczeństwa komputerowego.

Kolejna bardzo ważna różnica między klasyczną kryptografią (powiedzmy sprzed lat 80. XX wieku) a współczesną kryptografią dotyczy jej przyjęcia. Historycznie rzecz biorąc, głównymi odbiorcami kryptografii były organizacje wojskowe i rządy. Dziś kryptografia jest wszędzie! Jeśli kiedykolwiek uwierzytelniałeś się poprzez wpisanie hasła, kupiłeś coś kartą kredytową w Internecie lub pobrałeś zweryfikowaną aktualizację dla swojego systemu operacyjnego, niewątpliwie korzystałeś z kryptografii. Coraz częściej programiści ze stosunkowo niewielkim doświadczeniem proszeni są o „zabezpieczenie” pisanych przez siebie aplikacji poprzez włączenie mechanizmów kryptograficznych.

Krótko mówiąc, kryptografia przeszła od heurystycznego zestawu narzędzi zapewniających tajną komunikację wojsku do nauki, która pomaga zabezpieczać systemy zwykłym ludziom na całym świecie. Oznacza to również, że kryptografia stała się bardziej centralnym tematem w informatyce.

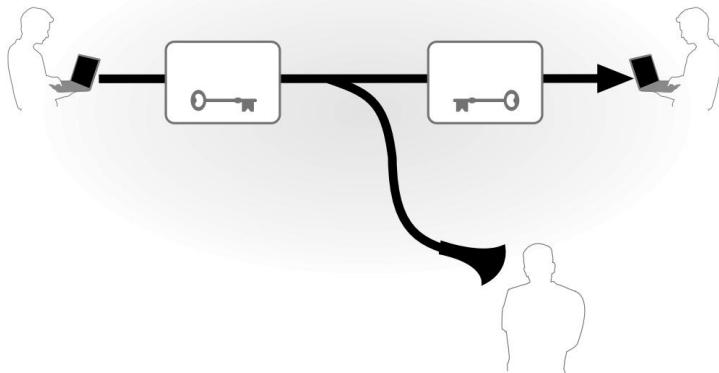
Cele tej książki. Naszym celem jest uczynienie podstawowych zasad nowoczesnością kryptografia dostępna dla studentów informatyki, elektrotechniki lub matematyki; dla profesjonalistów, którzy chcą zastosować kryptografię w systemach lub oprogramowaniu, które opracowują; i dla każdego, kto ma poziom podstawowy dojrzałości matematycznej, która jest zainteresowana zrozumieniem tego fascynującego zjawiska pole. Po przeczytaniu tej książki czytelnik powinien docenić gwarancje bezpieczeństwa, jakie mają zapewniać powszechnie stosowane prywatne kryptograficzne; Być świadomy standardowych (bezpiecznych) konstrukcji takich prywatnych; i móc dokonać podstawowej oceny nowych systemów w oparciu o dowody ich bezpieczeństwa (lub ich braku) oraz założenia matematyczne leżące u podstaw tych dowodów. Nie jest naszą intencją, aby czytelnicy stali się ekspertami – ani potrafili projektować nowe kryptosystemy – po ukończeniu tej książki, ale próbowałibyśmy zapewnić terminologię i podstawowy materiał potrzebny zainteresowanym czytelnikowi, aby mógł następnie zapoznać się z bardziej zaawansowanymi źródłami w tej dziedzinie.

Ten rozdział. Książka ta koncentruje się na formalnym badaniu współczesnej kryptografii, ale zaczniemy w tym rozdziale od bardziej nieformalnego omówienia „klasycznej” kryptografii. Poza tym, że pozwala nam to na łatwe wnikanie w materiał, nasze podejście omówione w tym rozdziale będzie miało motywację do przyjęcia bardziej rygorystycznego podejścia, które będziemy stosować w pozostałej części książki. Naszym zamiarem tutaj nie jest być wyczerpujące i jako taki nie należy traktować tego rozdziału jako reprezentatywnego opisu historycznego. Czytelnik zainteresowany historią kryptografii zachęcamy do zapoznania się z odnośnikami znajdującymi się na końcu tego rozdziału.

1.2 Ustawienie szyfrowania klucza prywatnego

Kryptografia klasyczna zajmowała się projektowaniem i używaniem kodów (także zwane szyframi), które umożliwiają dwóm stronom tajną komunikację w obecności podsłuchującego, który może monitorować całą komunikację między nimi. We współczesnym języku kody nazywane są schematami szyfrowania i takiej terminologii będziemy tutaj używać. Bezpieczeństwo wszystkich klasycznych schematów szyfrowania polegało na tajemniczy – kluczu – udostępnionym wcześniej przez komunikujące się strony i nieznanemu podsłuchującemu. Ten scenariusz jest znany jako klucz prywatny (lub ustawienie wspólnego/tajnego klucza), a szyfrowanie kluczem prywatnym to tylko jeden z przykładów prywatnych kryptograficznych używanych w tym ustawieniu. Zanim opiszymy niektóre historyczne schematy szyfrowania, omówimy bardziej ogólnie szyfrowanie kluczem prywatnym.

W przypadku szyfrowania kluczem prywatnym dwie strony dzielą klucz i go używają klucz, gdy chcą komunikować się w tajemniczy. Jedna strona może wysłać wiadomość, lub zwykły tekst, do drugiego za pomocą wspólnego klucza do szyfrowania (lub „szyfrowania”) wiadomość i w ten sposób uzyskać zaszyfrowany tekst, który jest przesyłany do odbiorcy. Odbiorca używa tego samego klucza do odszyfrowania (lub „rozszyfrowania”) tekstu zaszyfrowanego i odzyska oryginalną wiadomość. Należy pamiętać, że ten sam klucz służy do konwersji



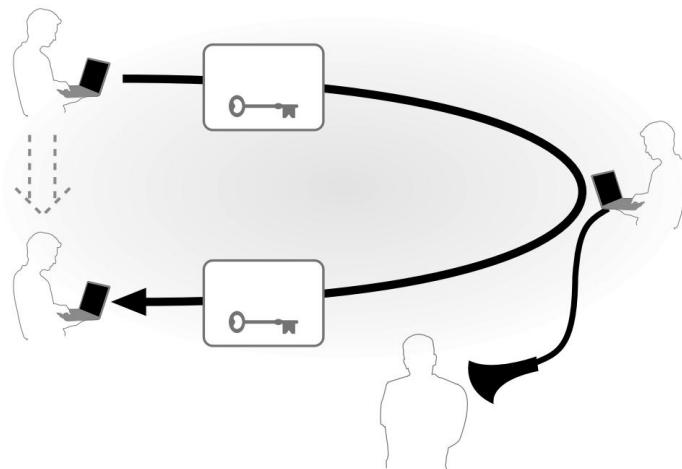
RYSUNEK 1.1: Jedno wspólne ustawienie kryptografii klucza prywatnego (w tym przypadku szyfrowanie): dwie strony dzielą klucz, którego używają do bezpiecznej komunikacji.

tekst jawnego na tekst zaszyfrowany i z powrotem; dlatego jest to również znane jako ustawienie klucza symetrycznego, gdzie symetria polega na tym, że obie strony posiadają ten sam klucz używany do szyfrowania i deszyfrowania. Różni się to od szyfrowania asymetrycznego, czyli z kluczem publicznym (przedstawionego w rozdziale 10), gdzie szyfrowanie i deszyfrowanie wykorzystują różne klucze.

Jak już wspomniano, celem szyfrowania jest ukrycie tekstu jawnego przed podsłuchującym, który może monitorować kanał komunikacyjny i obserwować zaszyfrowany tekst. Omówimy to bardziej szczegółowo w dalszej części tego rozdziału, a w rozdziałach 2 i 3 spędziemy dużo czasu na formalnym zdefiniowaniu tego celu.

Istnieją dwa kanoniczne zastosowania kryptografii klucza prywatnego. W pierwszym przypadku mamy do czynienia z dwiema odrębnymi stronami oddzielonymi przestrzenią, np. pracownik w Nowym Jorku komunikujący się ze swoim kolegą z Kalifornią; patrz rysunek 1.2. Zakłada się, że ci dwaj użytkownicy mogli bezpiecznie udostępnić klucz przed nawiązaniem komunikacji. (Należy pamiętać, że jeśli jedna z stron po prostu wyśle klucz do drugiej za pośrednictwem publicznego kanału komunikacji, wówczas podsłuchujący również uzyska klucz!) Często można to łatwo osiągnąć, organizując fizyczne spotkanie stron w bezpiecznym miejscu w celu udostępnienia klucza przed oddzieleniem; w podanym przykładzie współpracownicy mogą umówić się na wspólne korzystanie z klucza, gdy obaj będą w biurze w Nowym Jorku. W innych przypadkach bezpieczne udostępnienie klucza jest trudniejsze. W kolejnych kilku rozdziałach po prostu zakładamy, że możliwe jest współdzielenie klucza; wróćmy do tego zagadnienia w rozdziale 10.

Drugie powszechnie zastosowanie kryptografii klucza prywatnego polega na tym, że ta sama strona komunikuje się ze sobą w miarę upływu czasu. (Patrz rysunek 1.2.) Rozważmy np. szyfrowanie dysku, gdzie użytkownik szyfruje zwykły tekst i przechowuje wynikowy tekst zaszyfrowany na swoim dysku twardym; ten sam użytkownik powróci później



RYSUNEK 1.2: Inne popularne ustawienie kryptografii klucza prywatnego (ponownie szyfrowanie): pojedynczy użytkownik bezpiecznie przechowuje dane przez dłuższy czas.

w odpowiednim momencie na odszyfrowanie tekstu zaszyfrowanego i odzyskanie oryginalnych danych. Dysk twardy służy tutaj jako kanał komunikacyjny, na którym osoba atakująca może podsłuchać, uzyskując dostęp p do dysku twardego i czytając jego zawartość.

„Udostępnianie” klucza jest teraz banalne, chociaż użytkownik nadal potrzebuje bezpiecznego i niezawodnego sposobu na zapamiętanie/przechowanie klucza do wykorzystania w późniejszym czasie.

Składnia szyfrowania. Formalnie schemat szyfrowania klucza prywatnego definiuje się poprzez określenie przestrzeni wiadomości M wraz z trzema algorytmami: procedurą generowania kluczy (Gen), procedurą szyfrowania (Enc) i procedurą deszyfrowania (Dec). Przestrzeń komunikatów M definiuje zbiór komunikatów „legalnych”, czyli takich, które są obsługiwane przez schemat. Algorytmy posiadają następującą funkcjonalność:

1. Algorytm generowania klucza Gen jest algorymem probabilistycznym, który generuje klucz k wybrany według pewnego rozkładu.
2. Algorytm szyfrowania Enc przyjmuje jako dane wejściowe klucz k i wiadomość m , a na wyjściu wysyła zaszyfrowany tekst c . Oznaczamy przez $\text{Enc}_k(m)$ szyfrowanie tekstu jawnego m za pomocą klucza k .
3. Algorytm deszyfrujący Dec przyjmuje jako dane wejściowe klucz k i tekst zaszyfrowany c , a na wyjściu podaje tekst jawny m . Oznaczamy odszyfrowanie zaszyfrowanego tekstu c za pomocą klucza k przez $\text{Dec}_k(c)$.

Schemat szyfrowania musi spełniać następujące warunki poprawności: dla każdego klucza k wprowadzanego przez Gen i każdej wiadomości $m \in M$, utrzymuje się, że

$$\text{Pokład}(\text{Enc}_k(m)) = m.$$

Słowem: zaszyfrowanie wiadomości, a następnie odszyfrowanie powstałego tekstu zaszyfrowanego (przy użyciu tego samego klucza) daje oryginalną wiadomość.

Zbiór wszystkich możliwych kluczy wyrowadzanych przez algorytm generowania kluczy nazywany jest przestrzenią kluczy i jest oznaczony przez K. Prawie zawsze Gen po prostu wybiera jednolity klucz z przestrzeni kluczy; w rzeczywistości można założyć bez utraty ogólności, że tak jest (patrz ćwiczenie 2.1).

Przeglądając naszą wcześniejszą dyskusję, schemat szyfrowania może być używany przez dwie strony, które chcą komunikować się w następstwie pujący sposób. Najpierw uruchamiany jest Gen w celu uzyskania klucza k, który jest wspólny dla stron. Później, gdy jedna ze stron chce wysłać drugi tekst jawny m, oblicza c := Enck(m) i wysyła wynikowy tekst zaszyfrowany c kanałem publicznym do drugiej strony.¹ Po otrzymaniu c druga strona oblicza m := Deck(c), aby odzyskać oryginalny tekst jawny.

Zasada Keysa i Kerckhoffa. Jak wynika z powyższego, jeśli podsłuchujący przeciwnik zna algorytm Dec oraz klucz k współdzielony przez obie komunikujące się strony, wówczas przeciwnik ten będzie w stanie odszyfrować każdy szyfrogram przesyłany przez te strony. Z tego powodu komunikujące się strony muszą bezpiecznie dzielić klucz k i zachować k w całkowitej tajemnicy przed wszystkimi innymi. Być może powinni także zachować w tajemnicy algorytm deszyfrujący Dec? A jeśli o to chodzi, czy nie byłoby dla nich lepiej zachować w tajemnicy wszystkie szczegóły schematu szyfrowania?

Pod koniec XIX wieku Auguste Kerckhoff w swoim artykule wyjaśniającym kilka zasad projektowania szyfrów wojskowych argumentował coś przeciwnego. Jedna z najważniejszych z nich, obecnie znana po prostu jako zasada Kerckhoffa, brzmiała:

Nie można wymagać, aby metoda szyfrowania była tajna i mogła bez przeszkoły wpąść w ręce wroga.

Oznacza to, że schemat szyfrowania powinien być zaprojektowany tak, aby był bezpieczny, nawet jeśli osoba podsłuchująca zna wszystkie szczegóły schematu, o ile osoba atakująca nie zna używanego klucza. Inaczej mówiąc, bezpieczeństwo nie powinno polegać na tajności schematu szyfrowania; zamiast tego zasada Kerckhoffa wymaga, aby bezpieczeństwo opierało się wyłącznie na tajemnicy klucza.

Istnieją trzy główne argumenty przemawiające za zasadą Kerckhoffa. Po pierwsze, stronom znacznie łatwiej jest zachować w tajemnicy krótki klucz, niż zachować w tajemnicy (bardziej skomplikowany) algorytm, którego używają. Jest to szczególnie prawdziwe, jeśli wyobrażymy sobie szyfrowanie w celu zabezpieczenia komunikacji pomiędzy wszystkimi parami pracowników w jakiejś organizacji. O ile każda para stron nie używa własnego, unikalnego algorytmu, niektóre strony będą znać algorytm używany przez inne. Informacje na temat algorytmu szyfrowania mogą zostać ujawnione przez jednego z tych pracowników (powiedzmy po zwolnieniu) lub uzyskane przez osobę atakującą przy użyciu inżynierii wstępnej. Krótko mówiąc, założenie, że algorytm szyfrowania pozostanie tajny, jest po prostu nierealistyczne.

¹Używamy „:=” do oznaczenia przypisania deterministycznego i na razie zakładamy, że Enc jest deterministyczne. Listę typowych zapisów można znaleźć na końcu książki.

Po drugie, w przypadku ujawnienia tajnych informacji uczciwych stron, znacznie łatwiej będzie im zmienić klucz niż wymienić

schemat szyfrowania. (Rozważ aktualizację pliku zamiast instalowania nowego programu.) Co więcej, wygenerowanie nowego losowego sekretu jest stosunkowo proste,

mając na uwadze, że zaprojektowanie nowego schematu szyfrowania byłoby ogromnym przedsięwzięciem.

Wreszcie, w przypadku wdrożenia na dużą skalę, jest to znacznie łatwiejsze dla wszystkich użytkowników polegają na tym samym algorytmie/oprogramowaniu szyfrującym (z różnymi kluczami) co aby każdy mógł używać własnego, niestandardowego algorytmu. (Oznosi się to nawet do pojedynczy użytkownik, który komunikuje się z kilkoma różnymi stronami.) W rzeczywistości tak jest pożądane jest, aby schematy szyfrowania były ujednolicone, aby zapewnić (1) kompatybilność jest zapewnione domyślnie i (2) użytkownicy będą dążyli korzystać ze schematu szyfrowania, który ma zostały poddane kontroli publicznej i w których nie stwierdzono żadnych uchybień.

Obecnie zasadę Kerckhoffa rozumie się jako opowiadającą się za całkowitym upublicznieniem projektów kryptograficznych, co jest jaskrawym przeciwieństwem koncepcji „bezpieczeństwa przez zaciemnienie”, co sugeruje, że utrzymywanie algorytmów w tajemnicy zwiększa bezpieczeństwo. Używanie zastrzeżonego, „domowego” napoju jest bardzo niebezpieczne. algorytm (tj. niestandardowy algorytm zaprojektowany w tajemnicy przez jakąś firmę). Natomiast opublikowane projekty podlegają publicznej ocenie i dlatego są prawdopodobnie silniejszy. Wieloletnie doświadczenie pokazuje, że tak bardzo trudno skonstruować dobre schematy kryptograficzne. Dlatego nasza pewność co do bezpieczeństwa programu jest znacznie większa, jeśli był on realizowany na szeroką skalę zbadane (przez ekspertów innych niż twórcy programu) i nie wykazują żadnych słabych punktów zostało znalezione. Choć może się to wydawać proste i oczywiste, zasada otwartości Projekt kryptograficzny (tj. zasada Kerckhoffa) był wielokrotnie ignorowany ponownie z katastrofalnymi skutkami. Na szczęście dzisiaj jest wystarczająco dużo bezpiecznych, zestandardyzowanych i powszechnie dostępnych kryptosystemów, do których nie ma powodu użyć czegokolwiek innego.

1.3 Szyfry historyczne i ich kryptoanaliza

W naszym badaniu „klasycznej” kryptografii przeanalizujemy pewne elementy historyczne schematy szyfrowania i pokazać, że są one niebezpieczne. Naszym głównymi celami w prezentacji tego materiału są: (1) uwypuklenie słabości „ad hoc” podejścia do kryptografii, a tym samym zmotywowanie nowoczesnego, rygorystycznego podejścia zostanie to omówione w dalszej części książki oraz (2) zademonstrowanie tego prostego podejścia do osiągnięcia bezpiecznego szyfrowania raczej nie zakończy się sukcesem. Wzdłuż W ten sposób przedstawimy kilka głównych zasad kryptografii inspirowanych przez słabości tych historycznych schematów.

W tej sekcji znaki zwykłego tekstu są pisane małymi literami i szyfrowane. znaki tekstowe są pisane WIELKIMI LITERAMI dla przejrzystości typograficznej.

Szyfr Cezara. Jeden z najstarszych zarejestrowanych szyfrów, znany jako szyfr Cezara

szyfr, opisany jest w De Vita Caesarum, Divus Iulius („Życie Cezar, Deifikowany Juliusz”), napisany około 110 roku n.e.:

Są też jego listy do Cicerona i do bliskich w sprawach prywatnych, a w tym ostatnim, jeśli miał coś poufnego do powiedzenia, pisał to szyfrem, to znaczy zmieniając w ten sposób kolejność liter alfabetu, tak że nie można było rozróżnić ani słowa. . .

Juliusz Cezar zaszyfrował, przesuwając litery alfabetu o 3 miejsca do przodu: a zastąpiono D, b E i tak dalej. Na samym końcu alfabetu litery zawijają się, więc z zostało zastąpione przez C, y przez B, a x przez A. Na przykład szyfrowanie wiadomości rozpoczyna atak teraz, po usunięciu spacji, daje:

EHJLQWKHDWWDFNQRZ.

Bezpośrednim problemem związanym z tym szyfrem jest to, że metoda szyfrowania jest ustalona; nie ma klucza. Zatem każdy, kto dowie się, w jaki sposób Cezar zaszyfrował swoje wiadomości, będzie mógł bez wysiłku je odszyfrować.

Co ciekawe, odmiana tego szyfru o nazwie ROT-13 (gdzie przesunięcie wynosi 13 miejsc zamiast 3) jest nadal używana na różnych forach internetowych. Rozumie się, że nie zapewnia to żadnego bezpieczeństwa kryptograficznego; służy jedynie do zapewnienia, że tekst (powiedzmy, spoiler filmowy) będzie niezrozumiałym, chyba że czytelnik wiadomości świadomie zdecyduje się go odszyfrować.

Szyfr przesunięcia i zasada wystarczającej przestrzeni klucza. Szyfr z przesunięciem można postrzegać jako kluczową odmianę szyfru Cezara.² W szczególności w szyfrze z przesunięciem klucz k jest liczbą z zakresu od 0 do 25. Aby zaszyfrować, litery są przesuwane jak w szyfrze Cezara, ale teraz o k miejsc. Odwzorowując to na opisaną wcześniej składnię szyfrowania, przestrzeń wiadomości składa się z ciągów angielskich liter o dowolnej długości, z usuniętymi znakami interpunkcyjnymi, spacjami i cyframi, bez rozróżnienia na wielkie i małe litery. Algorytm Gen generuje jednolity klucz $k \in \{0, \dots, 25\}$; algorytm Enc pobiera klucz k i tekst jawnego i przesuwa każdą literę tekstu jawnego do przodu na pozycję k (zawijając na końcu alfabetu); a algorytm Dec pobiera klucz k i tekst zaszyfrowany i przesuwa każdą literę tekstu zaszyfrowanego o k pozycji wstecz.

Bardziej matematyczny opis uzyskuje się przez zrównanie angielskiego alfabetu ze zbiorzem $\{0, \dots, 25\}$ (więc $a = 0, b = 1$ itd.). Przestrzeń komunikatów M jest zatem dowolna skończona sekwencja liczb całkowitych z tego zbioru. Szyfrowanie wiadomości $m = m_1 \dots m_n$ (gdzie $m_i \in \{0, \dots, 25\}$) kluczem k wyraża się wzorem

$$\text{Enck}(m_1 \dots m_n) = c_1 \dots c_n, \text{ gdzie } c_i = [(m_i + k) \bmod 26].$$

(Zapis $[a \bmod N]$ oznacza resztę z dzielenia przez N , gdzie $0 \leq [a \bmod N] < N$. Mamy tu na myśli proces mapujący a na $[a \bmod N]$

²W niektórych książkach „szyfr Cezara” i „szyfr przesuwny” są używane zamiennie.

jako moduł redukcji N; bę dziemy mieli więcej do powiedzenia na ten temat na początku w rozdziale 8.) Odszyfrowanie szyfrogramu $c = c_1 \dots c$ przy użyciu klucza k jest dane przez

$$\text{Pokład}(c_1 \dots c) = m_1 \dots m, \text{ gdzie } m_i = [(c_i - k) \bmod 26].$$

Czy szyfr przesuwny jest bezpieczny? Zanim zaczniesz czytać dalej, spróbuj odszyfrować poniższe informacje tekst zaszyfrowany, który został wygenerowany przy użyciu szyfru przesunięcia i tajnego klucza k :

OVDTHUFWVZZPISLRLFZHYZAOLYL.

Czy można odzyskać wiadomość bez znajomości k ? Właściwie to banalne!

Powodem jest to, że możliwych jest tylko 26 kluczy. Można więc spróbować odszyfrować tekst zaszyfrowany przy użyciu każdego możliwego klucza i uzyskać w ten sposób listę 26 kandydujących tekstu jawnych. Poprawny tekst jawnny z pewnością będzie dzie na tej liście; co więcej, jeśli zaszyfrowany tekst jest „wystarczająco długi”, wówczas poprawny tekst jawnny będzie prawdopodobnie jedynym kandydatem na liście, który „ma sens”. (To drugie niekoniecznie jest prawdą, ale będzie prawdą w większości przypadków. Nawet jeśli tak nie jest, atak zawiera zbiór potencjalnych tekstu jawnych do maksymalnie 26 możliwości.) Skanując listę kandydatów, jest to łatwe aby odzyskać oryginalny tekst jawnny.

Atak polegający na wypróbowaniu każdego możliwego klucza nazywany jest atakiem brute-force lub wyczerpującym wyszukiwaniem. Oczywiście, aby schemat szyfrowania był bezpieczny, nie może być podatny na taki atak.³ Ta obserwacja jest znana jako zasada wystarczającej przestrzeni klucza:

Każdy bezpieczny schemat szyfrowania musi mieć wystarczająco dużą przestrzeń kluczy, aby uniemożliwić atak z wykorzystaniem wyczerpującego wyszukiwania.

Można debatować, ile wysiłku sprawia, że zadanie jest „niewykonalne”, a dokładne określenie wykonalności zależy zarówno od zasobów potencjalnego atakującego, jak i od tego, jak długo nadawca i odbiorca chcą zapewnić tajność swojej komunikacji. Obecnie napastnicy mogą wykorzystywać superkomputery, dziesiątki tysięcy komputerów osobistych lub procesorów graficznych (GPU), aby przyspieszyć ataki brute-force. Aby chronić przed takimi atakami, przestrzeń kluczy musi być bardzo duża —powiedzmy co najmniej 270, a nawet więcej, jeśli zależy nam na długoterminowym bezpieczeństwie przed dobrze finansowanym napastnikiem.

Zasada wystarczającej przestrzeni klucza daje warunek konieczny bezpieczeństwa, ale niewystarczający. Pokazuje to następujący przykład.

Monoalfabetyczny szyfr podstawieniowy. W szyfrze przesunięcia klucz definiuje mapę od każdej litery alfabetu (tekstu jawnego) do jakiejs litery alfabetu (tekstu zaszyfrowanego), gdzie mapa jest stałą przesunięciem określonym przez klucz. W monoalfabetycznym szyfrze podstawieniowym klucz definiuje również mapę alfabetu, ale teraz mapa może być dowolna, pod warunkiem, że jest ona jedyńka do jednego, aby możliwe było odszyfrowanie. Klucz

³Technicznie rzecz biorąc, jest to prawdą tylko wtedy, gdy przestrzeń komunikatu jest więcej niż spacja kluczy; powrócimy do tego punktu w Rozdziale 2. Stosowane w praktyce schematy szyfrowania mają tę właściwość.

przestrzeń składa się zatem ze wszystkich bijekcji, czyli permutacji alfabetu. Na przykład klucz definiujący następującą permutację

ABCDEFGHIJKLMNOPQRSTUVWXYZ	VWXYZ
XEUADNBKVMROCFSYHWGL ZIJPT	

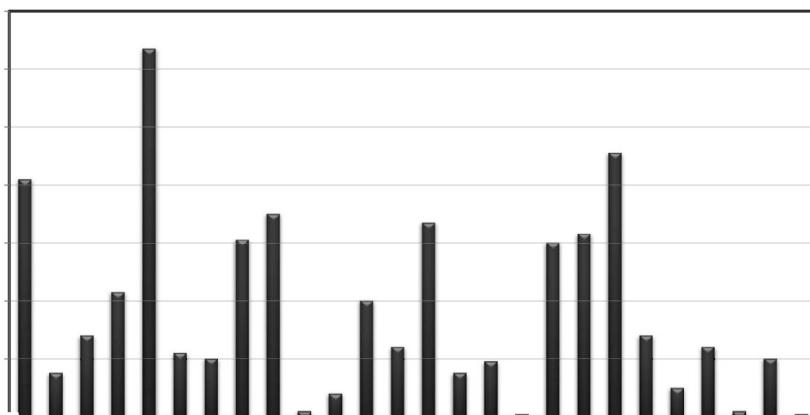
(w którym mapuje się do X itp.) zaszyfruje wiadomość tellhimaboutme do GDOOKVCXEFLGCD. Nazwa tego szyfru wynika z faktu, że klucz określa (stałe) podstawienie poszczególnych znaków tekstu jawnego.

Zakładając, że używany jest alfabet angielski, spacja klawiszy ma rozmiar $26! = 26 \cdot 25 \cdot 24 \cdots 2 \cdot 1$, czyli w przybliżeniu 288, a atak brute-force jest niemożliwy.

Nie oznacza to jednak, że szyfr jest bezpieczny! W rzeczywistości, jak pokażemy dalej, łatwo jest złamać ten schemat, mimo że ma on dużą przestrzeń na klucze.

Założymy, że tekst w języku angielskim jest szyfrowany (tzn. jest to tekst poprawny gramatycznie w języku angielskim, a nie tylko tekst napisany przy użyciu znaków alfabetu angielskiego). Monoalfabetyczny szyfr podstawieniowy można nastąpić zaatakować, wykorzystując wzorce statystyczne języka angielskiego. (Oczywiście ten sam atak działa w przypadku dowolnego języka.) Atak opiera się na faktach, że:

1. Dla dowolnego klucza mapowanie każdej litery jest ustalone, więc jeśli e jest mapowane na D, to każde pojawienie się e w tekście jawnym spowoduje pojawienie się D w tekście zaszyfrowanym.
2. Znany jest rozkład częstości występowania liter w języku angielskim (patrz rysunek 1.3). Oczywiście bardzo krótkie teksty mogą odbiegać od tego rozkładu, ale nawet teksty składające się zaledwie kilku zdani mają rozkłady bardzo zbliżone do średnich.



RYSUNEK 1.3: Średnia częstość występowania liter w tekście angielskim.

Atak polega na zestawieniu rozkładu częstotliwości znaków w zaszyfrowanym tekście, tj. zarejestrowaniu, że A pojawiło się 11 razy, B pojawiło się 4 razy i tak dalej. Częstotliwości te są następnie porównywane ze znaną częstotliwością liter w normalnym tekście angielskim. Można następnie pominąć porównywanie ze znakami, które nie występują w normalnym tekście angielskim. Na przykład, ponieważ e jest najczęstszą literą w języku angielskim, można się domyślić, że najczęstszy znak w zaszyfrowanym tekście odpowiada znakowi e w tekście jawnym i tak dalej. Niektóre domysły mogą być błędne, ale wystarczająca liczba domysłów będzie poprawna, aby umożliwić stosunkowo szybkie odszyfrowanie (zwłaszcza przy wykorzystaniu innej znajomości języka angielskiego), na przykład faktu, że u na ogół następuje po q i że h prawdopodobnie pojawi się pomiędzy dziesiątym a dwunastym znakiem. Dochodzimy do wniosku, że chociaż monoalfabetyczny szyfr podstawieniowy ma dużą przestrzeń kluczową, nadal jest niepewny.

Nie powinno dziwić, że monoalfabetyczny szyfr podstawieniowy można szybko złamać, skoro zagadki oparte na tym szyfrze pojawiają się w gazetach (a niektórzy rozwiązują je przed poranną kawą!). Zalecamy próbę odszyfrowania następującego tekstu zaszyfrowanego — powinno to przekonać Cię, jak łatwo jest przeprowadzić atak. (Skorzystaj z rysunku 1.3, który może Ci pomóc.)

JGRMQOYGHMVBWJWRWQFPWHGFFDQGFZRKBEEBJIZQQOCIBZKLFAFGQVFZFWWE
OGWOPFGFHOLPHLRLOLFDMDFGQWBWLWBWQOLKFWBLYLFSFLJGRMQBOLWJVFP
FWQVHQWFFFQOQVFPQOCFPOGFWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVPLIK
OGQILHQFQGIQVWOSFAFBQWBQVHQWIJVWJVFPFWHGIWIHZZRQGBABHZQOCGFHX

Ulepszony atak na szyfr przesuwny. Możemy użyć tabel częstotliwości liter, aby zapewnić lepszy atak na szyfr przesuwny. Nasz poprzedni atak na szyfr przesuwny wymagał odszyfrowania tekstu zaszyfrowanego przy użyciu każdego możliwego klucza, a następnie sprawdzenia, który klucz daje w wyniku zwykły tekst, który „ma sens”. Wadą tego podejścia jest to, że jest dość trudne do zautomatyzowania, ponieważ komputerowi trudno jest sprawdzić, czy dany tekst jawnny „ma sens”. (Nie twierdzimy, że byłoby to niemożliwe, ponieważ atak można zautomatyzować przy użyciu słownika ważnych angielskich słów. Twierdzimy jedynie, że zautomatyzowanie nie byłoby trywialne.) Co więc cej, mogą się zdarzyć przypadki — zobaczymy je później — gdzie znaki tekstu jawnego są rozmieszczone tak samo jak tekst w języku angielskim, mimo że sam tekst jawnny nie jest prawidłowym tekstem angielskim, w takim przypadku sprawdzanie tekstu jawnego, który „ma sens” nie będzie działało.

Opiszmy teraz atak, który nie ma tych wad. Tak jak poprzednio, skojarz literę alfabetu angielskiego z 0, ..., 25. Niech p_i oznacza częstotliwość występowania i-tej litery w normalnym tekście angielskim (bez spacji, znaków interpunkcyjnych itp.).

Obliczenia na podstawie rysunku 1.3 dają

$$\begin{array}{ccc} 25 & 2 & 0,065 \\ \text{Liczba} & & \\ \text{j}=0 & & \end{array} \quad (1.1)$$

Założymy teraz, że otrzymaliśmy jakiś zaszyfrowany tekst i niech q_i oznacza częstotliwość występowania i-tej litery alfabetu w tym zaszyfrowanym tekście; tj. q_i to po prostu liczba

i-tej litery alfabetu w szyfrogramie podzielonej przez długość szyfrogramu. Jeśli kluczem jest k, wówczas pi powinno być w przybliżeniu równe q_i+k dla wszystkich i, ponieważ i-ta litera jest odwzorowana na ($i + k$ -tą) literę. (Używamy $i+k$ zamiast bardziej kłopotliwego $[i+k \text{ mod } 26]$.) Zatem, jeśli obliczymy

$$I_j \stackrel{\text{def}}{=} \sum_{j=0}^{25} p_i \cdot q_i + j$$

dla każdej wartości j $\{0, \dots, 25\}$, to spodziewamy się, że $I_k = 0,065$ (gdzie k jest rzeczywistym kluczem), natomiast I_j dla $j = k$ będzie różne od 0,065.

Prowadzi to do łatwego do zautomatyzowania ataku polegającego na odzyskaniu klucza: oblicz I_j dla wszystkich j, a następnie wypisz wartość k, dla której I_k jest najbliższe 0,065.

Szyfr Vigen`ere'a (z przesunięciem polialfabetycznym). Atak statystyczny na monoalfabetyczny szyfr podstawieniowy można przeprowadzić, ponieważ klucz definiuje stałe mapowanie, które jest stosowane litera po literze do tekstu jawnego. Taki atak można udaremnić za pomocą polialfabetycznego szyfru podstawieniowego, w którym klucz zamiast tego definiuje mapowanie stosowane w blokach znaków zwykłego tekstu. Tutaj, na przykład, klucz może odwzorowywać 2-znakowy blok ab na DZ podczas odwzorowywania ac na TY; zauważ, że znak tekstu jawnego a nie jest mapowany na stały znak tekstu zaszyfrowanego. Szyfry podstawieniowe polialfabetyczne „wyglądzają” rozkład częstości znaków w zaszyfrowanym tekście i utrudniają analizę statystyczną.

Szyfr Vigen`ere'a, szczególny przypadek powyższego, zwany także szyfrem z przesunięciem polialfabetycznym, działa poprzez zastosowanie kilku niezależnych instancji szyfru z przesunięciem po kolei. Klucz jest teraz postrzegany jako ciąg liter; szyfrowanie odbywa się poprzez przesunięcie każdego znaku zwykłego tekstu o wielkość wskazywaną przez następny znak klucza, w razie potrzeby zawijając klucz.

(To degeneruje się do szyfru przesunięcia, jeśli klucz ma długość 1.) Na przykład szyfrowanie wiadomości „powiedz mu o mnie” za pomocą kawiarni z kluczem będzie działać w następujący sposób:

Tekst jawny:	powiedz mu o mnie
Klucz (powtórzony):	cafecafecafeca
Tekst zaszyfrowany:	VEQPJIREDOZXOE

(Kluczem nie musi być angielskie słowo.) Jest to dokładnie to samo, co szyfrowanie pierwszego, piątego, dziewiątego, ... znaki z szyfrem shift i kluczem c; drugi, szósty, dziesiąty, ... znaki z klawiszem a; trzeci, siódmy, ... znaki z f; i czwarty, ósmy, ... znaki z e. Zauważ, że w powyższym przykładzie l jest odwzorowywane raz na Q, a raz na P. Co więcej, znak szyfrogramu E jest czasami uzyskiwany z e, a czasami z a. W ten sposób częstości znaków w zaszyfrowanym tekście są „wyglądzane” zgodnie z potrzebami.

Jeśli klucz jest wystarczająco długi, złamanie tego szyfru wydaje się trudne. Rzeczywiście, przez wielu uważano go za „niezniszczalny” i choć wynaleziono go w XVI wieku, systematyczny atak na ten schemat wymyślono dopiero setki lat później.

Atakowanie szyfru Vigen`ere'a. Pierwszą obserwacją dotyczącą atakowania szyfru Vi-gen`ere jest to, że jeśli znana jest długość klucza, atak na szyfr jest stosunkowo łatwy. W szczególności powiedzmy, że długość klucza, zwana także kropką, wynosi t. Zapisz klucz k jako $k = k_1 \dots k_t$, gdzie każde k_i jest literą alfabetu.

Zaobserwowany zaszyfrowany tekst $c = c_1 c_2 \dots$ można podzielić na t części, przy czym każdą część można uznać za zaszyfrowaną przy użyciu szyfru przesuwnego. W szczególności dla wszystkich $j \in \{1, \dots, t\}$ znaki szyfrogramu

$$c_j, c_{j+1}, c_{j+2}, \dots$$

wszystko to wynikało z przesunięcia odpowiednich znaków tekstu jawnego o pozycję k_j . Powyższą sekwencję znaków nazywamy j -tym strumieniem.

Pozostaje tylko określić dla każdego z t strumieni, które z 26 możliwych przesunięć zostało zastosowane. Nie jest to tak trywialne, jak w przypadku szyfru przesunięcia, ponieważ nie jest już możliwe po prostu wypróbowanie różnych przesunięć w celu ustalenia, kiedy odszyfrowanie strumienia „ma sens”. (Przypomnijmy, że strumień nie odpowiada kolejnym literom tekstu jawnego.)

Co więc cej, próba odgadnięcia całego klucza k na raz wymagałaby przeszukania metodą brute-force przez 26^t różnych możliwości, co jest niewykonalne w przypadku dużego t.

Niemniej jednak nadal możemy używać analizy częstotliwości liter do niezależnej analizy każdego strumienia. Mianowicie, dla każdego strumienia zestawiamy częstotliwość każdego znaku tekstu zaszyfrowanego, a następnie sprawdzamy, które z 26 możliwych przesunięć daje „właściwy” rozkład prawdopodobieństwa dla tego strumienia. Ponieważ można to przeprowadzić niezależnie dla każdego strumienia (tj. dla każdego znaku klucza), atak ten wymaga czasu $26 \cdot t$, a nie czasu 26^t .

Bardziej zasadowym i łatwiejszym do zautomatyzowania podejściem jest użycie ulepszonej metody atakowania szyfru przesuwnego omówionej wcześniej. Atak ten nie polegał na sprawdzeniu, czy tekst jawnny ma „sens”, ale opierał się jedynie na podstawowym rozkładzie częstotliwości znaków w tekście jawnym.

Każde z powyższych podejść zapewnia skuteczny atak przy określonej długości klucza, jest znana. Co się stanie, jeśli długość klucza jest nieznana?

Zauważmy najpierw, że o ile maksymalna długość T klucza nie jest zbyt duża, możemy po prostu powtórzyć powyższy atak T razy (dla każdej możliwej wartości $t \in \{1, \dots, T\}$). Prowadzi to do co najwyżej T różnych kandydujących tekstów jawnych, spośród których prawdziwy tekst jawnego będzie prawdopodobnie łatwy do zidentyfikowania. Zatem nieznana długość klucza nie jest poważną przeszkodą.

Istnieją również bardziej wydajne sposoby określenia długości klucza na podstawie zaobserwowanego tekstu zaszyfrowanego. Jednym z nich jest metoda Kasiskiego, opublikowana w połowie XIX wieku. Pierwszym krokiem jest identyfikacja powtarzających się wzorców o długości 2 lub 3 w zaszyfrowanym tekście. Są one prawdopodobnie wynikiem pewnych bigramów lub trygramów, które często pojawiają się w tekście jawnym. Rozważmy na przykład popularne słowo „the”. To słowo zostanie odwzorowane na różne znaki szyfrogramu, w zależności od jego pozycji w tekście jawnym. Jeśli jednak pojawi się dwukrotnie w tej samej pozycji względem dnej, zostanie odwzorowany na te same znaki tekstu zaszyfrowanego. W przypadku wystarczająco długiego tekstu jawnego istnieje zatem duża szansa, że „the” zostanie wielokrotnie odwzorowane na te same znaki tekstu zaszyfrowanego.

Rozważmy następujący konkretny przykład z kluczowymi koralikami (spacje mają dodano dla przejrzystości):

Tekst jawnny: mę życzna i kobieta odebrali list z poczty bea dsb ead sbe adsbe adsbeadsb ead sbeadsb
Klucz: koralik sbe adsb eadsbe
Tekst zaszyfrowany: ULE PSO ENG LII WREBR RHLSMEYWE XHH DFXTHJ GVOP LII PRKU SFIADI

Słowo the jest czasami mapowane na ULE, czasami na LII, a czasami na XHH. Jednakże jest on odwzorowywany dwukrotnie na LII, a w wystarczająco długim tekście jest prawdopodobne, że zostałby przypisany wielokrotnie do każdej z tych możliwości.

Kasiski zauważył, że odległość między takimi powtarzającymi się zjawiskami (zakładając, że nie są przypadkowe) musi być wielokrotnością okresu.

(W powyższym przykładzie okres wynosi 5, a odległość między dwoma wystąpieniami II wynosi 30, co stanowi 6-krotność okresu.) Zatem najwiekszy wspólny dzielnik

LH wynosi 30, co stanowi 3-krotność okresu. Zatem najwięcej kasy wspólny dzierżawik odległości pomiędzy powtarzającymi się ciągami (zakładając, że nie są one przypadkowe) da długość klucza t lub jej wielokrotność.

Alternatywne podejście, zwane metodą indeksu koincydencji, jest bardziej metodyczne i dlatego łatwiejsze do zautomatyzowania. Przypomnijmy, że jeśli długość klucza wynosi t , to znaki szyfrogramu

$c_1, c_1+t, c_1+2t, \dots$

w pierwszym strumieniu wszystko wynikało z szyfrowania przy użyciu tego samego przesunięcia. Oznacza to, że oczekuje się, że częstość występowania znaków w tej sekwencji będzie identyczna z częstością znaków w standardowym tekście angielskim w jakiejś przesuniętej kolejności. Bardziej szczegółowo: niech q_i oznacza obserwowaną częstość występowania i -tej angielskiej litery w tym strumieniu; jest to po prostu liczba wystąpień i -tej litery alfabetu podzielona przez całkowitą liczbę liter w strumieniu. Jeśli zastosowanym tutaj przesunięciem jest j (tzn. jeśli pierwszy znak klucza jest równy j), to dla wszystkich i oczekujemy $q_{i+j} = p_i$, gdzie p_i jest częstością i -tej litery alfabetu w standardzie Angielski tekst. (Po raz kolejny używamy q_{i+j} zamiast $q[i+j \bmod 26]$.) Oznacza to jednak, że ciąg q_0, \dots, q_{25} jest po prostu ciągiem p_0, \dots, p_{25} przesunięty o j miejsc. W konsekwencji (por. Równanie (1.1)):

$$\begin{array}{ccc} 25 & 25 & \\ \text{q}_-^2 & & 2 p_- \\ |a=0 & & |a=0 \end{array}$$

Prowadzi to do dobrego sposobu określenia długości klucza t . Dla $\tau = 1, 2, \dots$, spójrz na sekwencję znaków szyfrogramu $c_1, c_{1+\tau}, c_{1+2\tau}, \dots$ i zestawić q_0, \dots, q_{25} dla tej sekwencji. Następnie oblicz

$$\begin{array}{ccc} & 25 & \\ \text{St} & \xrightarrow{\text{def}} & q \text{ ja .} \\ & & \text{j a = 0} \end{array}$$

Gdy $\tau = t$, oczekujemy $S\tau = 0,065$, jak omówiono powyżej. Z drugiej strony, jeśli τ nie jest wielokrotnością t , spodziewamy się, że wszystkie znaki będą występuwać mniej więcej w jednolitej liczbie

prawdopodobieństwo w ciągu $c_1, c_1+\tau, c_1+2\tau, \dots$, więc oczekujemy, że $1/26$ dla wszystkich i . W tym przypadku otrzymamy

$$\begin{array}{r} 25 \\ \text{St} \\ \hline 1 & 262 \\ ja=0 & 0,038. \end{array}$$

Najmniejsza wartość τ , dla której $\text{St} = 0,065$ jest zatem prawdopodobnie długością klucza. Można dodatkowo zweryfikować przypuszczenie τ , przeprowadzając podobne obliczenia przy użyciu drugiego strumienia $c_2, c_2+\tau, c_2+2\tau, \dots$.

Długość szyfrrogramu i ataki kryptoanalytyczne. Powyższe ataki na szyfr Vigen`ere wymagają dłuższego tekstu zaszyfrowanego niż ataki na poprzednie schematy. Na przykład metoda indeksu koincydencji wymaga, aby $c_1, c_1+\tau, c_1+2\tau$ (gdzie τ jest rzeczywistą długością klucza) były wystarczająco długie, aby zapewnić, że obserwowane częstotliwości odpowiadają oczekiwaniom; sam szyfrrogram musi być wtedy mniej więcej taki razy więcej. Podobnie, pokazany przez nas atak na monoalfabetyczny szyfr podstawieniowy wymaga dłuższego tekstu zaszyfrowanego niż atak na szyfr przesuwny (który może działać w przypadku sztuffowania nawet pojedynczego słowa). To pokazuje, że dłuższy klucz może, ogólnie rzecz biorąc, wymagać od kryptoanalytyka uzyskania więcej ilości tekstu zaszyfrowanego w celu przeprowadzenia ataku. (W istocie można wykazać, że szyfr Vigen`ere jest bezpieczny, jeśli klucz jest tak długi, jak długość zaszyfrowanego klucza.)

(Podobne zjawisko zobaczymy w następym rozdziale.)

Wnioski. Zaprezentowaliśmy jedynie kilka szyfrów historycznych. Oprócz ich zainteresowania historycznego, naszym celem podczas ich prezentacji było zilustrowanie kilku ważnych lekcji. Być może najważniejsze jest to, że projektowanie bezpiecznych szyfrów jest trudne.

Szyfr Vigen`era pozostawał nieprzerwany przez długi czas. Zastosowano również znacznie bardziej złożone schematy. Jednak złożony schemat niekoniecznie jest bezpieczny, a wszystkie historyczne schematy zostały złamane.

1.4 Zasady współczesnej kriptografii

Jak powinno być jasne z poprzedniej sekcji, kriptografia była historycznie bardziej sztuką niż nauką. Schematy opracowywano ad hoc i oceniano na podstawie ich postrzeganej złożoności lub sprytu. Schemat zostanie przeanalizowany pod kątem wykrycia jakichkolwiek ataków; jeśli tak, schemat zostanie „załatwany” w celu udaremnenia tego ataku i proces się powtórzy. Chociaż mogła panować zgoda co do tego, że niektóre systemy nie są bezpieczne (o czym świadczy szczególnie szkodliwy atak), nie było uzgodnionej koncepcji, jakie wymagania powinien spełniać „bezpieczny” system, ani nie było możliwości przedstawienia dowodu, że jakikolwiek konkretny program było bezpieczne.

W ciągu ostatnich kilku dekad kriptografia stała się bardziej nauką. Schematy są obecnie opracowywane i analizowane w bardziej systematyczny sposób.

sposób, a ostatecznym celem jest przedstawienie rygorystycznego dowodu, że dana konstrukcja jest bezpieczna. Aby sformułować takie dowody, potrzebujemy najpierw formalnych definicji, które dokładnie określają, co oznacza „bezpieczny”; takie definicje są przydatne i interesujące same w sobie. Jak się okazuje, wiek ksość dowodów kryptograficznych opiera się na obecnie nieudowodnionych założeniach dotyczących trudności algorytmicznej niektórych problemów matematycznych; wszelkie tego typu założenia muszą być wyraźnie i precyzyjnie określone. Nacisk na definicje, założenia i dowody odróżnia współczesną kryptografię od klasycznej kryptografii; omówimy te trzy zasady bardziej szczegółowo w kolejnych sekcjach.

1.4.1 Zasada 1 – Definicje formalne

Jednym z kluczowych osiągnięć współczesnej kryptografii było uznanie, że formalne definicje bezpieczeństwa są niezbędne do prawidłowego projektowania, badania, oceny i wykorzystania prymitywów kryptograficznych. Mówiąc wprost:

Jeśli nie rozumiesz, co chcesz osiągnąć, skąd możesz wiedzieć, kiedy (i czy) to osiągnąłeś?

Formalne definicje zapewniają takie zrozumienie, dając jasny opis, jakie zagrożenia wchodzą w zakres i jakie gwarancje bezpieczeństwa są pożądane. Jako takie, definicje mogą pomóc w projektowaniu schematów kryptograficznych. Rzeczywiście, znacznie lepiej jest sformalizować wymagania przed rozpoczęciem procesu projektowania, niż tworzyć definicję post factum, gdy projekt jest już ukończony.

To drugie podejście stwarza ryzyko zakończenia fazy projektowania w momencie wyczerpania się cierpliwości projektantów (a nie w momencie osiągnięcia celu) lub może skutkować osiągnięciem przez konstrukcję więcej celów, niż jest to konieczne, kosztem wydajności.

Definicje umożliwiają także ocenę i analizę tego, co jest skonstruowane. Mając już definicję, można przestudiować proponowany program, aby sprawdzić, czy zapewnia on pożądane gwarancje; w niektórych przypadkach można nawet udowodnić, że dana konstrukcja jest bezpieczna (patrz podrozdział 1.4.3), wykazując, że spełnia ona definicję. Z drugiej strony definicje mogą służyć do jednoznacznego wykazania, że dany schemat nie jest bezpieczny, o ile nie spełnia on definicji. W szczególności należy pamiętać, że ataki opisane w poprzedniej sekcji nie wykazują automatycznie, że którykolwiek z pokazanych tam schematów jest „niebezpieczny”. Na przykład atak na szyfr Vigen`ere zakładał, że szyfrowany jest wystarczająco długi tekst w języku angielskim, ale czy szyfr Vigen`ere może być „bezpieczny”, jeśli jest to krótki tekst w języku angielskim lub tekst skompresowany (który będzie miał mniej więcej jednakową częstotliwość liter), jest zaszyfrowany? Bez formalnej definicji trudno to stwierdzić.

Definicje umożliwiają znaczące porównanie schematów. Jak zobaczymy, istnieje wiele (poprawnych) sposobów definiowania bezpieczeństwa; „właściwy” zależy od kontekstu, w jakim schemat jest używany. Schemat spełniający słabszą definicję może być bardziej efektywny niż inny schemat spełniający silniejszą definicję; dzięki temu precyzyjnym definicjom możemy właściwie ocenić kompromisy między tymi dwoma schematami. Podobnie definicje umożliwiają bezpieczne korzystanie ze schematów. Rozważ kwestię wyboru schematu szyfrowania

użyć do jakiegoś większego zastosowania. Rozsądny sposobem podejścia do problemu jest najpierw zrozumienie, jakie pojęcie bezpieczeństwa jest wymagane dla danej aplikacji, a następnie znalezienie schematu szyfrowania spełniającego to założenie. Dodatkową korzyścią tego podejścia jest modułowość: projektant może „zamienić” jeden schemat szyfrowania i zastąpić go innym (który również spełnia niezbędną definicję bezpieczeństwa), nie martwiąc się o wpływ na bezpieczeństwo całej aplikacji.

Napisanie formalnej definicji zmusza do zastanowienia się, co jest istotne dla rozpatrywanego problemu, a jakie właściwości są obce. Przejście przez ten proces często ujawnia subtelności problemu, które na pierwszy rzut oka nie były oczywiste. Zilustrujemy to dalej w przypadku szyfrowania.

Przykład: bezpieczne szyfrowanie. Często stym błędem jest myślenie, że formalne definicje nie są potrzebne lub są łatwe do wymyślenia, ponieważ „każdy ma intuicyjne pojęcie o tym, co oznacza bezpieczeństwo”. Nie o to chodzi.

Jako przykład rozważmy przypadek szyfrowania. (Czytelnik może zatrzymać się w tym miejscu, aby pomyśleć o tym, jak formalnie zdefiniowałby, co to znaczy, że schemat szyfrowania jest bezpieczny.) Chociaż formalną definicję bezpiecznego szyfrowania przekładamy na następujące dwa rozdziałы, opisujemy tutaj nieformalnie, co takie definicja powinna uchwycić.

Ogólnie rzecz biorąc, definicja bezpieczeństwa składa się z dwóch elementów: gwarancji bezpieczeństwa (lub, z punktu widzenia atakującego, co stanowi skuteczny atak na schemat) oraz modelu zagrożenia. Gwarancja bezpieczeństwa określa, przed czym schemat ma uniemożliwić atakującemu, natomiast model zagrożenia opisuje siłę przeciwnika, tj. jakie działania, jak zakłada się, jest w stanie wykonać atakujący.

Zaczniemy od pierwszego z nich. Jaki powinien być bezpieczny schemat szyfrowania gwarancja? Oto kilka myśli:

- Osoba atakująca nie powinna mieć możliwości odzyskania klucza. Zaobserwowałyśmy wcześniej, że jeśli atakujący może określić klucz wspólnie przez dwie strony przy użyciu jakiegoś schematu, wówczas schemat ten nie może być bezpieczny. Łatwo jednak wymyślić schematy, w przypadku których odzyskanie klucza jest niemożliwe, a mimo to są one zarówno niebezpieczne. Rozważmy np. schemat, w którym $Enck(m) = m$. Zaszyfrowany tekst nie ujawnia żadnych informacji o kluczu (w związku z tym klucz nie może zostać odzyskany, jeśli jest wystarczająco długi), a mimo to wiadomość jest wysyłana w sposób jawnym! Widzimy zatem, że niemożność odzyskania klucza nie jest wystarczająca dla bezpieczeństwa. Ma to sens: celem szyfrowania jest ochrona wiadomości; klucz jest środkiem do osiągnięcia tego celu, ale sam w sobie jest nieistotny.
- Osoba atakująca nie powinna mieć możliwości odzyskania całego tekstu jawnego z tekstu zaszyfrowanego. Definicja ta jest lepsza, lecz wciąż daleka od zadowalającej. W szczególności definicja ta uznałaby schemat szyfrowania za bezpieczny, gdyby jego zaszyfrowane teksty ujawniały 90% tekstu jawnego, o ile 10% tekstu jawnego pozostawało trudne do rozszeryfrowania. Jest to wyraźnie niedopuszczalne w większości powszechnych zastosowań szyfrowania; na przykład podczas szyfrowania

bazy danych o wynagrodzeniach, słusznie bylibyśmy zdenerwowani, gdyby ujawniono 90% wynagrodzeń pracowników!

- Osoba atakująca nie powinna mieć możliwości odzyskania jakiegokolwiek znaku tekstu jawnego z tekstu zaszyfrowanego. Wygląda na to, że jest to dobra definicja, ale wciąż niewystarczająca. Wracając do przykładu szyfrowania bazy danych o wynagrodzeniach, nie uznalibyśmy schematu szyfrowania za bezpieczny, jeśli ujawni on, czy wynagrodzenie pracownika jest wyższe czy mniejsze niż 100 000 USD, nawet jeśli nie ujawnia żadnej konkretnej cyfry wynagrodzenia tego pracownika.
- Podobnie nie chcielibyśmy, aby schemat szyfrowania ujawnił, czy pracownik A zarabia więcej niż pracownik B.

Inną kwestią jest to, jak sformalizować, co dla przeciwnika oznacza „odzyskanie charakteru tekstu jawnego”. Co się stanie, jeśli atakujący prawidłowo odgadnie, dzięki której szczegółowi lub zewnętrzne trzymać informacjom, że najmniej znacząca cyfra czyjego wynagrodzenia to 0? Oczywiście nie powinno to powodować braku bezpieczeństwa schematu szyfrowania, dlatego każda realna definicja musi w jakiś sposób wykluczać takie zachowanie jako udany atak.

- „Właściwa” odpowiedź: niezależnie od informacji, które atakujący już posiada, zaszyfrowany tekst nie powinien powodować wycieku żadnych dodatkowych informacji na temat ukrytego tekstu jawnego. Ta nieformalna definicja uwzględnia wszystkie problemy opisane powyżej. Należy w szczególności zauważać, że nie ma tu próby zdefiniowania, jakie informacje w tekście jawnym są „znaczące”; wymaga po prostu, aby żadne informacje nie wyciekły. Jest to ważne, ponieważ oznacza, że bezpieczny schemat szyfrowania jest odpowiedni dla wszystkich potencjalnych zastosowań, w których wymagana jest tajemnica.

Brakuje tu precyzyjnego, matematycznego sformułowania definicji. Jak powinniśmy przechwycić wcześniejszą wiedzę osoby atakującej na temat tekstu jawnego? A co to znaczy (nie) ujawniać informacji? Do tych pytań powrócimy w następnych dwóch rozdziałach; patrz zwłaszcza Definicje 2.3 i 3.12.

Teraz, gdy ustaliliśmy cel bezpieczeństwa, pozostaje określić model zagrożenia. Określa to, jaką „moc” zakłada się, że atakujący ma, ale nie nakłada żadnych ograniczeń na strategię przeciwnika. To ważne rozróżnienie: określamy, co zakładamy na temat zdolności przeciwnika, ale nie zakładamy niczego na temat tego, jak on wykorzystuje te zdolności. Nie da się przewidzieć, jakie strategie można zastosować w ataku, a historia pokazała, że próby takiego zastosowania są skazane na niepowodzenie.

Istnieje kilka prawdopodobnych opcji modelu zagrożenia w kontekście szyfrowania; standardowe, w kolejności rosnącej siły atakującego, to:

- Atak wykorzystujący wyłącznie zaszyfrowany tekst: Jest to najbardziej podstawowy atak i odnosi się do scenariusza, w którym przeciwnik po prostu obserwuje zaszyfrowany tekst (lub wiele zaszyfrowanych tekstów) i próbuje uzyskać informacje o leżącym u jego podstaw tekście jawnym (lub tekstach jawnych). To jest model zagrożenia, jaki mieliśmy

pośrednio zakładając podczas omawiania klasycznych schematów szyfrowania w poprzedniej sekcji.

- Atak znanym tekstem jawnym: W tym przypadku przeciwnik może poznać jedną lub wiele cej par tekst jawnego/tekst zaszyfrowany wygenerowanych przy użyciu jakiegoś klucza. Celem przeciwnika jest następne wydedukowanie informacji na temat tekstu jawnego leżącego u podstaw innego tekstu zaszyfrowanego utworzonego przy użyciu tego samego klucza.

Wszystkie klasyczne schematy szyfrowania, które widzieliśmy, są łatwe do złamania przy użyciu ataku ze znanym tekstem jawnym; demonstrację zostawiamy jako ćwiczenie.

- Atak wybranym tekstem jawnym: W tym ataku przeciwnik może uzyskać pary tekst jawnego/tekst zaszyfrowany (jak powyżej) dla wybranego przez siebie tekstu jawnego.
- Atak z wybranym tekstem zaszyfrowanym: Ostatni typ ataku to taki, w którym przeciwnik jest w stanie dodatkowo uzyskać (pewne informacje na temat) odszyfrowania wybranego przez siebie tekstu zaszyfrowanego, np. czy odszyfrowanie jakiegoś tekstu zaszyfrowanego wybranego przez atakującego daje prawidłowy angielski wiadomość. Celem przeciwnika jest po raz kolejny poznanie informacji o tekście jawnym innego zaszyfrowanego tekstu (którego odszyfrowanie nie jest w stanie bezpośrednio uzyskać).

Żaden z tych modeli zagrożeń nie jest z natury lepszy od innych; wybór odpowiedniego zależy od środowiska, w którym wdrożono schemat szyfrowania.

Pierwsze dwa rodzaje ataków są najłatwiejsze do przeprowadzenia. W przypadku ataku wykorzystującego wyłącznie szyfrogramy jedyne, co musi zrobić przeciwnik, to podsłuchać publiczny kanał komunikacyjny, przez który przesyłane są zaszyfrowane wiadomości. W ataku ze znanym tekstem jawnym zakłada się, że przeciwnik w jakiś sposób uzyskuje również teksty zaszyfrowane odpowiadające znanym tekstem jawnym. Często jest to łatwe do osiągnięcia, ponieważ nie wszystkie zaszyfrowane wiadomości są poufne, przynajmniej nie na czas nieokreślony. Jako trywialny przykład, dwie strony mogą zawsze szyfrować wiadomość „część” za każdym razem, gdy rozpoczynają komunikację. Bardziej złożonym przykładem może być szyfrowanie, które pozwala zachować tajemnicę kwartalnych raportów o zarobkach aż do daty ich publikacji; w takim przypadku każdy, kto podsłucha zaszyfrowany tekst, uzyska później odpowiedni tekst jawnego.

W dwóch ostatnich atakach zakłada się, że przeciwnik jest w stanie uzyskać szyfrowanie i/lub deszyfrowanie wybranych przez siebie tekstów jawnych/zaszyfrowanych. Na pierwszy rzut oka może się to wydawać dziwne, dlatego bardziej szczegółowe omówienie tych ataków i ich praktyczności odkładamy do sekcji 3.4.2 (w przypadku ataków z wybranym tekstem jawnym) i sekcji 3.7 (w przypadku ataków z wybranym tekstem zaszyfrowanym).

1.4.2 Zasada 2 – Precyzyjne założenia

Większość nowoczesnych konstrukcji kryptograficznych nie można bezwarunkowo udowodnić bezpieczeństwa; takie dowody wymagałyby rozwiązania pytań z teorii złożoności obliczeniowej, które wydają się dziś dalekie od odpowiedzi. Wynik

ten niefortunny stan rzeczy polega na tym, że dowody bezpieczeństwa zazwyczaj opierają się na założeniach. Współczesna kryptografia wymaga, aby wszelkie tego typu założenia były jednoznaczne i matematycznie precyzyjne. Na najbardziej podstawowym poziomie dzieje się tak po prostu dlatego, że wymagają tego matematyczne dowody bezpieczeństwa. Ale są też inne powody:

1. Weryfikacja założeń: Założenia ze swojej natury są stwierdzeniami, które nie są udowodnione, ale zamiast tego przyjmuje się, że są prawdziwe. Aby umocnić naszą wiarę w jakieś założenie, konieczne jest jego zbadanie. Im częściej założenie jest sprawdzane i testowane bez obalenia, tym bardziej jesteśmy pewni, że jest ono prawdziwe. Co więcej, badanie założenia może dostarczyć dowodów na jego zasadność, pokazując, że wynika ono z innego założenia, w które również powszechnie się wierzy.

Jeżeli założenie, na którym się opierasz, nie jest precyzyjnie określone, nie można go zbadać ani (potencjalnie) obalić. Zatem warunkiem wstępnym zwieńczenia naszej pewności co do danego założenia jest dokładne określenie tego, co dokładnie się zakłada.

2. Porównanie schematów: Często w kryptografii mamy do czynienia z dwoma schematami, co do których można wykazać, że spełniają pewną definicję, a każdy opiera się na innym założeniu. Zakładając, że wszystko inne jest takie samo, który schemat powinien być preferowany? Jeżeli założenie, na którym opiera się pierwszy schemat, jest słabsze od założenia, na którym opiera się drugi schemat (tzn. drugie założenie implikuje pierwsze), to pierwszy schemat jest preferowany, gdyż może się okazać, że drugie założenie jest fałszywe podczas gdy pierwsze założenie jest prawdziwe. Jeżeli założenia stosowane w obu schematach nie są porównywalne, ogólną zasadą jest preferowanie schematu opartego na lepiej zbadanych założeniach, co do których istnieje większa pewność.

3. Zrozumienie niezbędnych założeń: Schemat szyfrowania może opierać się na pewnych podstawowych elementach składowych. Jeśli później w bloku konstrukcyjnym zostaną wykryte pewne słabe punkty, w jaki sposób możemy stwierdzić, czy schemat szyfrowania jest nadal bezpieczny? Jeżeli podstawowe założenia dotyczące modułu konstrukcyjnego zostaną jasno określone w ramach udowadniania bezpieczeństwa programu, wystarczy jedynie sprawdzić, czy na wymagane założenia mają wpływ nowe, wykryte słabe punkty.

Czasami pojawia się pytanie: zamiast udowadniać bezpieczeństwo systemu w oparciu o inne założenia, dlaczego nie założyć po prostu, że sama konstrukcja jest bezpieczna? W niektórych przypadkach —np. gdy system skutecznie opiera się na atakom przez wiele lat —może to być rozsądne podejście. Jednak takie podejście nigdy nie jest preferowane i jest wręcz niebezpieczne, gdy wprowadzany jest nowy system. Powyższe powody pomagają wyjaśnić dlaczego. Po pierwsze, lepsze jest założenie, które było testowane przez kilka lat, niż nowe, doraźne założenie

który zostaje wprowadzony wraz z nową konstrukcją. Po drugie, ogólnie preferuje się założenia, które są prostsze do sformułowania, ponieważ łatwiej je zbadać i (potencjalnie) obalić. Na przykład założenie, że jakiś problem matematyczny jest trudny do rozwiązania, jest prostsze do zbadania i oceny niż założenie, że schemat szyfrowania spełnia złożoną definicję bezpieczeństwa. Kolejną zaletą polegania na założeniach „niższego poziomu” (a nie tylko założenia, że konstrukcja jest bezpieczna) jest to, że te założenia niskiego poziomu można zwykle zastosować w innych konstrukcjach. Wreszcie założenia niskiego poziomu mogą zapewnić modułowość. Rozważmy schemat szyfrowania, którego bezpieczeństwo opiera się na zakładanej właściwości jednego z jego elementów składowych. Jeśli okaże się, że podstawowy element składowy nie spełnia podanych założeń, schemat szyfrowania można nadal utworzyć przy użyciu innego komponentu, który uważa się za spełniający niezbędną wymagania.

1.4.3 Zasada 3 – Dowody zabezpieczenia

Dwie zasady opisane powyżej pozwalają nam osiągnąć nasz cel, jakim jest dostarczenie rygorystycznego dowodu, że konstrukcja spełnia zadaną definicję przy pewnych określonych założeniach. Takie dowody są szczególnie ważne w kontekście kryptografii, gdzie atakujący aktywnie próbuje „złamać” jakiś schemat. Dowody bezpieczeństwa dają żelazną gwarancję – w odniesieniu do definicji i założeń – że żadnemu atakującemu się nie uda; jest to o wiele lepsze niż przyjęte pozbawionego zasad lub heurystycznego podejścia do problemu. Bez dowodu na to, że żaden przeciwnik dysponujący określonymi zasobami nie jest w stanie złamać jakiegoś schematu, pozostaje nam jedynie intuicja, że tak właśnie jest. Doświadczenie pokazało, że intuicja w kryptografii i bezpieczeństwie komputerów jest zgubna. Istnieje niezliczona ilość przykładów niepotwierdzonych schematów, które zostały złamane, czasami natychmiast, a czasem po latach od ich opracowania.

Podsumowanie: Rygorystyczne a doraźne podejście do bezpieczeństwa

Opieranie się na definicjach, założeniach i dowodach stanowi rygorystyczne podejście do kryptografii, które różni się od nieformalnego podejścia do kryptografii klasycznej. Niestety, pozbawione zasad, niestandardowe rozwiązania są nadal projektowane i wdrażane przez tych, którzy chcą uzyskać szybkie rozwiązanie problemu, lub przez tych, którzy po prostu nie mają wiedzy. Mamy nadzieję, że ta książka przyczyni się do zwiększenia świadomości rygorystycznego podejścia i jego znaczenia w opracowywaniu bezpiecznych systemów.

1.4.4 Bezpieczeństwo możliwe do udowodnienia i bezpieczeństwo w świecie rzeczywistym

Duża część współczesnej kryptografii opiera się obecnie na solidnych podstawach matematycznych. Nie oznacza to jednak, że dziedzina ta nie jest już po części sztuką. Rygorystyczne podejście pozostawia miejsce na kreatywność w opracowywaniu definicji dostosowanych do współczesnych zastosowań i środowisk oraz w proponowaniu nowych

założeń matematycznych lub projektowania nowych prymitywów, a także konstruowania nowych schematów i udowadniania ich bezpieczeństwa. Oczywiście zawsze będzie dzie istniała sztuka atakowania wdrożonych kryptosystemów, nawet jeśli zostaną udowodnione, że są bezpieczne. Dalej rozwijamy ten punkt.

Podejście stosowane we współczesnej kriptografii zrewolucjonizowało tę dziedzinę i pomaga zapewnić bezpieczeństwo schematów kryptograficznych wdrożonych w prawdziwym świecie. Ważne jest jednak, aby nie przeceniać tego, co oznacza dowód bezpieczeństwa. Dowód bezpieczeństwa zawsze odnosi się do rozważanej definicji i stosowanych założień. Jeśli gwarancja bezpieczeństwa nie odpowiada potrzebom lub model zagrożenia nie oddaje prawdziwych możliwości przeciwnika, dowód może być nieistotny. Podobnie, jeśli założenie, na którym się opieramy, okaże się fałszywe, wówczas dowód bezpieczeństwa nie będzie dzie miał żadnego znaczenia.

Wniosek jest taki, że możliwe do udowodnienia bezpieczeństwo programu niekoniecznie oznacza bezpieczeństwo tego programu w świecie rzeczywistym.⁴ Choć niektórzy postrzegali to jako wadę dającego się udowodnić bezpieczeństwa, my postrzegamy to optymistycznie jako ilustrację siły zbliżać się. Aby zaatakować możliwy do udowodnienia bezpieczny schemat w świecie rzeczywistym, wystarczy skupić uwagę na definicji (tj. zbadać, w jaki sposób wyidealizowana definicja różni się od rzeczywistego środowiska, w którym schemat jest wdrażany) lub leżących u jej podstaw założeniach (tj. zobacz czy trzymają). Z kolei zadaniem kryptografów jest ciągłe udoskonalanie swoich definicji, aby lepiej odpowiadały światu rzeczywistemu, a także sprawdzanie ich założzeń w celu sprawdzenia ich trafności. Możliwe do udowodnienia bezpieczeństwo nie kończy odwiecznej bitwy pomiędzy atakującym a obrońcą, ale zapewnia ramy, które pomagają przehylić szalę zwycięstwa na korzyść obrońcy.

Referencje i dodatkowe lektury

W tym rozdziale przestudiowaliśmy tylko kilka znanych szyfrów historycznych. Jest wiele innych o znaczeniu zarówno historycznym, jak i matematycznym, dlatego w celu uzyskania dalszych szczegółów odsyłamy czytelnika do podręczników Stinsona [168] lub Trappe i Washingtona [169]. Ważna rola, jaką kriptografia odegrała na przestrzeni dziejów, jest fascynującym tematem poruszonym w książkach Kahn [97] i Singha [163].

Zasady Kerckhoffa wyjaśniono w [103, 104]. Shannon [154] jako pierwszy zastosował rygorystyczne podejście do kriptografii oparte na precyzyjnych definicjach i dowodach matematycznych; omówimy jego pracę w następny razem rozdziale.

⁴Tutaj nawet nie rozważamy możliwości nieprawidłowej realizacji schematu. Źle zaimplementowana kriptografia jest poważnym problemem w prawdziwym świecie, ale problem ten wykracza poza zakres kriptografii jako takiej.

Ćwiczenia

- 1.1 Odszyfruj tekst zaszyfrowany podany na końcu sekcji dotyczącej monoalfabetycznych szyfrów podstawieniowych.
- 1.2 Podaj formalną definicję algorytmów Gen, Enc i Dec dla monoalfabetyczny szyfr podstawieniowy.
- 1.3 Podaj formalną definicję algorytmów Gen, Enc i Dec dla szyfru Vigen`ere'a. (Uwaga: istnieje kilka prawdopodobnych opcji dla Gen; wybierz jedną.)
- 1.4 Zaimplementuj ataki opisane w tym rozdziale dla szyfru przesuwnego i szyfru Vigen`ere.
- 1.5 Pokaż, że szyfry przesunięcia, podstawienia i Vigen`ere'a są łatwe do złamania przy użyciu ataku wybranego tekstu jawnego. Ile wybranego tekstu jawnego potrzeba do odzyskania klucza dla każdego z szyfrów?
- 1.6 Założmy, że atakujący wie, że hasło użytkownika to abcd lub bedg.
Założmy, że użytkownik szyfruje swoje hasło za pomocą szyfru przesuwnego, a osoba atakująca widzi wynikowy tekst zaszyfrowany. Pokaż, w jaki sposób atakujący może poznać hasło użytkownika lub wyjaśnij, dlaczego nie jest to możliwe.
- 1.7 Powtórz poprzednie ćwiczenie dla szyfru Vigen`ere'a, używając okresu 2, okresu 3 i okresu 4.
- 1.8 Szyfry przesunięcia, podstawienia i szyfry Vigen`ere'a można również zdefiniować w 128-znakowym alfabetie ASCII (zamiast 26-znakowego alfabetu angielskiego).
 - (a) W tym przypadku podać formalną definicję każdego z tych programów. (b) Omów, w jaki sposób ataki, które pokazaliśmy w tym rozdziale, można zmodyfikować, aby złamać każdy z tych zmodyfikowanych schematów.

Rozdział 2

Idealnie tajne szyfrowanie

W poprzednim rozdziale przedstawiliśmy historyczne schematy szyfrowania i pokazaliśmy, jak można je złamać przy niewielkim wysiłku obliczeniowym. W tym rozdziale przyjrzymy się innym skrajnym sposobom i przestudiujemy schematy szyfrowania, które są w sposób udowodniony bezpieczne nawet w przypadku przeciwnika z nieograniczoną mocą obliczeniową. Takie plany nazywane są doskonale tajnymi. Oprócz rygorystycznego zdefiniowania tego pojęcia, zbadamy warunki, w których można osiągnąć idealną tajemnicę. (Na początku tego rozdziału zakładamy znajomość podstawowej teorii prawdopodobieństwa. Przegląd odpowiednich pojęć znajduje się w Załączniku A.3.)

Materiał zawarty w tym rozdziale należy w pewnym sensie bardziej do świata kryptografii „klasycznej” niż do świata kryptografii „nowoczesnej”. Pomijając fakt, że cały przedstawiony tutaj materiał powstał przed rewolucją w kryptografii, która miała miejsce w połowie lat 70. i 80. XX w., konstrukcje, które badamy w tym rozdziale, opierają się wyłącznie na pierwszej i trzeciej zasadzie zarysowanych w podrozdziale 1.4. Oznacza to, że stosowane są dokładne definicje matematyczne i podawane są rygorystyczne dowody, ale nie będą dziekień poleganie na jakichkolwiek nieudowodnionych założeniach obliczeniowych. Zdecydowanie korzystne jest unikanie takich założeń; przekonamy się jednak, że takie postępowanie ma nieodłączne ograniczenia.

Zatem wyniki tego rozdziału nie tylko stanowią dobrą podstawę do zrozumienia zasad leżących u podstaw współczesnej kryptografii, ale także uzasadniają późniejsze przyjęcie przez nas wszystkich trzech wyżej wymienionych zasad.

Począwszy od tego rozdziału zdefiniujemy zabezpieczenia i przeanalizujemy schematy za pomocą eksperymentów probabilistycznych z udziałem algorytmów dokonujących losowych wyborów; podstawowym przykładem jest wybór losowego klucza przez strony komunikujące się. Zatem zanim powrócimy do tematu kryptografii per se, krótko omówimy kwestię generowania losowości odpowiedniej dla zastosowań kryptograficznych.

Generowanie losowości. W całej książce będziemy po prostu zakładać, że strony mają dostęp do nieograniczonej liczby niezależnych, bezstronnych, losowych bitów. W praktyce, skąd pochodzą te przypadkowe bity? W zasadzie można rzucić monetą. Ale takie podejście nie jest zbyt wygodne ani skalowalne.

Nowoczesne generowanie liczb losowych przebiega w dwóch etapach. Najpierw zbierana jest „pula” danych o wysokiej entropii. (Dla naszych celów nie jest potrzebna formalna definicja entropii i wystarczy pomyśleć o entropii jako o mierze nieprzewidywalności). Następnie dane o wysokiej entropii są przetwarzane w celu uzyskania sekwencji prawie niezależnych i nieobciążonych bitów. Ten drugi krok jest konieczny, ponieważ dane o wysokiej entropii niekoniecznie są jednolite.

Na pierwszym etapie potrzebne jest źródło nieprzewidywalnych danych. Istnieje kilka sposobów pozyskiwania takich danych. Jedną z technik jest poleganie na danych wejściowych zewnątrznych, na przykład opóźnieniach międuzy zdarzeniami sieciowymi, czasie dostępu do dysku twardego, naciśnięciem klawiszy lub ruchach myszą wykonywanych przez użytkownika i tak dalej. Takie dane prawdopodobnie nie będą jednolite, ale jeśli zostanie wykonana wystarczająca liczba pomiarów, oczekuje się, że uzyskana pula danych będzie miała wystarczającą entropię. Zastosowano również bardziej wyrafinowane podejścia, które z założenia ściślej uwzględniają generowanie liczb losowych w systemie na poziomie sprzętu. Opierają się one na zjawiskach fizycznych, takich jak szum termiczny/strzałowy lub rozpad radioaktywny. Firma In-tel opracowała niedawno procesor, który zawiera cyfrowy generator liczb losowych w chipie procesora i zapewnia dedykowaną instrukcję dostępu do wynikowych bitów losowych (po ich odpowiednim przetworzeniu w celu uzyskania niezależnych, bezstronnych bitów, jak omówiono poniżej).

Przetwarzanie potrzebne do „wygładzenia” danych o wysokiej entropii w celu uzyskania (prawie) jednolitych bitów nie jest trywialne i zostało pokrótko omówione w sekcji 5.6.4. Tutaj podajemy prosty przykład, aby dać wyobrażenie o tym, co zostało zrobione. Wyobraź sobie, że nasza pula o wysokiej entropii wynika z sekwencji stronniczych rzutów monetą, gdzie „reszka” pojawia się z prawdopodobieństwem p , a „gorzka” z prawdopodobieństwem $1-p$. (Zakładamy jednak, że wynik dowolnego rzutu monetą jest niezależny od wszystkich innych rzutów monetą.) W praktyce to założenie zazwyczaj nie jest uzasadnione.) Wynik 1000 takich rzutów monetą z pewnością ma wysoką entropię, ale nie jest zbliżony do jednolitego. Możemy uzyskać równomierny rozkład, rozważając rzuty monetą parami: jeśli zobaczymy reszkę, po której następuje reszka, wówczas wygenerujemy „0”, a jeśli zobaczymy gorzko, po której następuje reszka, wówczas wygenerujemy „1”. (Jeśli zobaczymy dwie reszki lub dwie gorskie w rzędzie, nic nie wyprowadza nas i po prostu przechodzimy do następnej pary.) Prawdopodobieństwo, że jakakolwiek para da „0” wynosi $p \cdot (1-p)$, co jest dokładnie równe równe prawdopodobieństwu, że dowolna para da „1” i w ten sposób otrzymujemy równomiernie rozłożony wynik z naszej początkowej puli o wysokiej entropii.

Należy zachować ostrożność podczas tworzenia losowych bitów, a użycie kiepskich generatorów liczb losowych może często narazić dobry kryptosystem na atak. Należy używać generatora liczb losowych przeznaczonego do zastosowań kryptograficznych, a nie generatora liczb losowych „ogólnego przeznaczenia”, który nie nadaje się do zastosowań kryptograficznych. W szczególności funkcja `Rand()` w bibliotece C `stdlib.h` nie jest bezpieczna kryptograficznie, a użycie jej w ustawieniach kryptograficznych może mieć katastrofalne skutki.

2.1 Definicje

Zaczniemy od przypomnienia i rozwinięcia składni wprowadzonej w poprzednim rozdziale. Schemat szyfrowania definiują trzy algorytmy `Gen`, `Enc` i `Dec`, a także specyfikacja (skończonej) przestrzeni wiadomości M

z $|M| > 1$. Algorytm generowania klucza Gen jest algorytmem probabilistycznym, który generuje klucz k wybrany według pewnego rozkładu. Oznaczamy przez K skończoną przestrzeń kluczy, tj. zbiór wszystkich możliwych kluczy, które mogą zostać wyprowadzone przez Gen. Algorytm szyfrowania Enc jako dane wejściowe przyjmuje klucz k $\in K$ oraz wiadomość $m \in M$ i wyprowadza tekst zaszyfrowany c. Pozwalamy teraz, aby algorytm szyfrowania był probabilistyczny (aby $Enck(m)$ mógł wygenerować inny tekst zaszyfrowany po wielokrotnym uruchomieniu) i zapisujemy $c = Enck(m)$, aby oznaczyć możliwie probabilistyczny proces, w wyniku którego wiadomość m jest szyfrowana przy użyciu klucza k podać szyfrogram c. (W przypadku, gdy Enc jest deterministyczny, możemy to podkreślić, pisząc $c := Enck(m)$). Patrząc w przyszłość, czasami używamy również notacji $x \in S$ do oznaczenia jednolitego wyboru x ze zbioru S.) Niech C oznacza zbiór wszystkich możliwych szyfrogramów, które może wyprowadzić $Enck(m)$, dla wszystkich możliwych wyborów $k \in K$ i $m \in M$ (oraz dla wszystkich losowych wyborów Enc w przypadku, gdy jest to losowe). Algorytm deszyfrujący Dec przyjmuje na wejściu klucz k $\in K$ oraz tekst zaszyfrowany c $\in C$ i wysyła wiadomość $m \in M$. Zakładamy doskonałą poprawność, co oznacza, że dla wszystkich $k \in K$, $m \in M$ i dowolnego tekstu zaszyfrowanego c wysyłanego przez $Enck(m)$ utrzymuje, że $Deck(c) = m$ z prawdopodobieństwem 1. Doskonała poprawność oznacza, że możemy założyć, że Dec jest deterministyczny bez utraty ogólności, ponieważ $Deck(c)$ musi dawać ten sam wynik przy każdym uruchomieniu. W ten sposób napiszemy $m := Deck(c)$, aby oznaczyć proces deszyfrowania tekstu zaszyfrowanego c przy użyciu klucza k w celu uzyskania wiadomości m.

W definicjach i twierdzeniach poniżej odwołujemy się do rozkładów prawdopodobieństwa po K, M i C. Rozkład po K jest zdefiniowany przez uruchomienie Gen i pobranie wyniku. (Prawie zawsze jest tak, że Gen wybiera klucz równomiernie z K i w zasadzie możemy to założyć bez utraty ogólności; patrz ćwiczenie 2.1.) Niech K będzie zmienną losową oznaczającą wartość klucza wyjściowego产生的 Gen ; zatem dla dowolnego $k \in K$ $Pr[K = k]$ oznacza prawdopodobieństwo, że klucz wyjściowy Gen jest równy k. Podobnie pozwalamy, aby M było zmienną losową oznaczającą zaszyfrowaną wiadomość, zatem $Pr[M = m]$ oznacza prawdopodobieństwo, że wiadomość przyjmie wartość m $\in M$. Rozkład prawdopodobieństwa wiadomości nie jest zdeterminowany samym schematem szyfrowania , ale zamiast tego odzwierciedla prawdopodobieństwo wysłania różnych wiadomości przez strony korzystające z schematu, a także niepewność przeciwnika co do tego, co zostanie wysłane. Na przykład przeciwnik może wiedzieć, że wiadomość zostanie dzisiaj zaatakowana lub nie będzie dzisiaj atakowana. Przeciwnik może nawet wiedzieć (w inny sposób), że z prawdopodobieństwem 0,7 wiadomość będzie dzisiaj atakowana, a z prawdopodobieństwem 0,3 wiadomość nie będzie dzisiaj atakowana, aby nie atakować. W tym przypadku mamy $Pr[M = \text{atak dzisiaj}] = 0,7$ i $Pr[M = \text{nie atakuj}] = 0,3$.

Zakłada się , że K i M są niezależne, tzn. to, co komunikują strony, jest niezależne od wspólnego klucza. Ma to sens mię dzy innymi dlatego, że rozkład na K jest określony przez

¹Jeśli $|M| = 1$ jest tylko jedna wiadomość i nie ma sensu jej komunikować, a co dopiero szyfrować.

sam schemat szyfrowania (ponieważ jest zdefiniowany przez Gen), podczas gdy rozkład na M zależy od kontekstu, w którym schemat szyfrowania jest używany.

Ustalenie schematu szyfrowania i rozkładu po M wyznacza rozkład po przestrzeni zaszyfrowanych tekstów C dany poprzez wybór klucza $k \in K$ (wg Gen) i wiadomości M (wg zadanego rozkładu), oraz następnie obliczamy szyfrogram $c = \text{Enck}(m)$. Niech C będzie zmienną losową oznaczającą wynikowy tekst zaszyfrowany, więc dla $c \in C$ napiszemy $\Pr[C = c]$, aby oznaczyć prawdopodobieństwo, że zaszyfrowany tekst jest równy ustalonej wartości c .

Przykład 2.1

Pracujemy nad prostym przykładem szyfru przesywanego (por. sekcja 1.3). Tutaj z definicji mamy $K = \{0, \dots, 25\}$ przy $\Pr[K = k] = 1/26$ dla każdego $k \in K$.

Powiedzmy, że mamy następujący rozkład na M:

$$\Pr[M = a] = 0,7 \text{ i } \Pr[M = z] = 0,3.$$

Jakie jest prawdopodobieństwo, że zaszyfrowanym tekstem jest B? Może to nastąpić tylko na dwa sposoby: albo $M = a$ i $K = 1$, albo $M = z$ i $K = 2$. Dzięki niezależności M i K mamy

$$\begin{aligned} \Pr[M = a \mid K = 1] &= \Pr[M = a] \cdot \Pr[K = 1] \\ &= 0,7 \cdot \frac{1}{26}. \end{aligned}$$

Podobnie $\Pr[M = z \mid K = 2] = 0,3 \cdot \frac{1}{26}$. Dlatego,

$$\begin{aligned} \Pr[C = B] &= \Pr[M = a \mid K = 1] + \Pr[M = z \mid K = 2] \\ &= 0,7 \cdot \frac{1}{26} + 0,3 \cdot \frac{1}{26} = 1/26. \end{aligned}$$

Możemy również obliczyć prawdopodobieństwa warunkowe. Na przykład, jakie jest prawdopodobieństwo, że wiadomość a została zaszyfrowana, biorąc pod uwagę, że obserwujemy tekst zaszyfrowany B? Korzystając z twierdzenia Bayesa (Twierdzenie A.8) mamy

$$\begin{aligned} \Pr[M = a \mid C = B] &= \Pr[C = B \mid M = a] \cdot \Pr[M = a] \\ &= \frac{\Pr[C = B \mid M = a] \cdot \Pr[M = a]}{\Pr[C = B]} \\ &= \frac{0,7 \cdot \Pr[C = B \mid M = a]}{1/26}. \end{aligned}$$

Zauważ, że $\Pr[C = B \mid M = a] = 1/26$, ponieważ jeśli $M = a$, to jedyny sposób, w jaki może zaistnieć $C = B$, to $K = 1$ (co zachodzi z prawdopodobieństwem 1/26). Dochodzimy do wniosku, że $\Pr[M = a \mid C = B] = 0,7$.

Przykład 2.2

Rozważmy ponownie szyfr przesunięcia, ale z następującym rozkładem na M:

$$\Pr[M = \text{kim}] = 0,5, \Pr[M = \text{ann}] = 0,2, \Pr[M = \text{boo}] = 0,3.$$

Jakie jest prawdopodobieństwo, że $C = DQQ$? Jedyny sposób, w jaki ten zaszyfrowany tekst może wystąpić, to $M = \text{ann}$ i $K = 3$ lub $M = \text{boo}$ i $K = 2$, co ma miejsce z prawdopodobieństwem $0,2 \cdot 1/26 + 0,3 \cdot 1/26 = 1/52$.

Jakie jest zatem prawdopodobieństwo, że Ann została zaszyfrowana, pod warunkiem przestrzegania szyfrogramu DQQ ? Obliczenie jak powyżej przy użyciu twierdzenia Bayesa daje $\Pr[M = \text{ann} | C = DQQ] = 0,4$.

Doskonała tajemnica. Jesteśmy teraz gotowi zdefiniować pojęcie doskonałej tajemnicy. Wyobrażamy sobie przeciwnika, który zna rozkład prawdopodobieństwa w M ; oznacza to, że przeciwnik zna prawdopodobieństwo wysłania różnych wiadomości. Ten przeciwnik zna również używany schemat szyfrowania; jedyną rzeczą nieznaną przeciwnikowi jest klucz wspólny dla stron. Wiadomość jest wybierana przez jedną z uczciwych stron i szyfrowana, a powstały zaszyfrowany tekst przesyłany jest do drugiej strony. Przeciwnik może podsłuchać komunikację stron i w ten sposób zaobserwować ten zaszyfrowany tekst. (Oznacza to, że jest to atak wykorzystujący wyłącznie tekst zaszyfrowany, w którym atakujący otrzymuje tylko jeden tekst zaszyfrowany). Aby schemat był całkowicie tajny, przestrzeganie tego tekstu zaszyfrowanego nie powinno mieć wpływu na wiedzę przeciwnika na temat faktycznej wysłanej wiadomości; innymi słowy, prawdopodobieństwo a posteriori, że wysłano jakąś wiadomość $m \in M$, uwarunkowane zaobserwowanym tekstem zaszyfrowanym, nie powinno różnić się od prawdopodobieństwa apriorycznego, że m zostanie wysłane. Oznacza to, że zaszyfrowany tekst nie ujawnia niczego na temat ukrytego tekstu jawnego, a przeciwnik nie dowiaduje się absolutnie niczego na temat zaszyfrowanego tekstu jawnego. Formalnie:

DEFINICJA 2.3 Schemat szyfrowania (Gen , Enc , Dec) z przestrzenią wiadomości M jest doskonale tajny, jeśli dla każdego rozkładu prawdopodobieństwa w M , każdej wiadomości $m \in M$ i każdego tekstu zaszyfrowanego $c \in C$, dla którego $\Pr[C = c] > 0$:

$$\Pr[M = m | C = c] = \Pr[M = m].$$

(Wymóg, aby $\Pr[C = c] > 0$ jest wymogiem technicznym niezbędnym do zapobiegania warunkowaniu zdarzenia o zerowym prawdopodobieństwie.)

Podajemy teraz równoważne sformułowanie doskonałej tajemnicy. Nieformalnie to sformułowanie wymaga, aby rozkład prawdopodobieństwa szyfrogramu nie zależał od tekstu jawnego, tj. dla dowolnych dwóch wiadomości $m, m' \in M$ rozkład szyfrogramu, gdy m jest szyfrowane, powinien być identyczny z rozkładem tekstu zaszyfrowanego, gdy m' jest szyfrowane. Formalnie dla każdego $m, m' \in M$ i każdego $c \in C$,

$$\Pr[\text{Enc}_K(m) = c] = \Pr[\text{Enc}_K(m') = c] \tag{2.1}$$

(gdzie prawdopodobieństwa zależą od wyboru K i dowolnej losowości Enc). Oznacza to, że zaszyfrowany tekst nie zawiera informacji o tekście jawnym i że niemożliwe jest odróżnienie szyfrowania m od szyfrowania m' , ponieważ rozkłady w zaszyfrowanym tekście są w każdym przypadku takie same.

LEMAT 2.4 Schemat szyfrowania (Gen, Enc, Dec) z przestrzenią wiadomości M jest całkowicie tajny wtedy i tylko wtedy, gdy równanie (2.1) zachodzi dla każdego $m, m \in M$ i każdego $c \in C$.

DOWÓD Pokazujemy, że jeśli podany warunek jest spełniony, to plan jest całkowicie tajny; implikację odwrotną pozostawiamy ćwiczeniu 2.4. Napraw rozkład po M , wiadomości m i zaszyfrowanym tekście c , dla którego $\Pr[C = c] > 0$.

Jeśli $\Pr[M = m] = 0$, to trywialnie mamy

$$\Pr[M = m | C = c] = 0 = \Pr[M = m].$$

Założymy więc $\Pr[M = m] > 0$. Najpierw zauważcie to

$$\Pr[C = c | M = m] = \Pr[\text{Enc}K(M) = c | M = m] = \Pr[\text{Enc}K(m) = c],$$

gdzie pierwsza równość wynika z definicji zmiennej losowej C , oraz

po drugie, stawiamy warunek na przypadku, gdy M jest równe m . Ustaw &

def

$\Pr[\text{Enc}K(m) = c] = \Pr[C = c | M = m]$. Jeżeli warunek lematu jest spełniony, to dla każdego $m \in M$ mamy $\Pr[\text{Enc}K(m) = c] = \Pr[C = c | M = m] = \delta c$.

Korzystając z twierdzenia Bayesa (patrz Dodatek A.3), mamy zatem

$$\begin{aligned} \Pr[M = m | C = \text{do}] &= \frac{\Pr[C = c | M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{\Pr[C = c | M = m] \cdot \Pr[M = m]}{\Pr[m \in M] \cdot \Pr[C = c | M = m] + \Pr[m \notin M] \cdot \Pr[C = c]} \\ &= \frac{\delta c \cdot \Pr[M = m]}{\Pr[m \in M] + \Pr[m \notin M]} \\ &= \frac{\Pr[M = m]}{\Pr[m \in M] + \Pr[m \notin M]} = \Pr[M = m], \end{aligned}$$

gdzie sumowanie następuje po $m \in M$ z $\Pr[M = m] = 0$. Dochodzimy do wniosku, że dla każdych $m \in M$ i $c \in C$, dla których $\Pr[C = c] > 0$, utrzymuje się, że $\Pr[M = m | C = c] = \Pr[M = m]$, więc schemat jest całkowicie tajny. ■

Doskonała (kontradydaktywna) nieroróżnialność. Zakończymy tę sekcję przedstawieniem innej równoważnej definicji doskonałej tajemnicy. Definicja ta opiera się na eksperymencie, w którym przeciwnik biernie obserwuje zaszyfrowany tekst, a następnie próbuje odgadnąć, która z dwóch możliwych wiadomości została zaszyfrowana. Wprowadzamy to pojęcie, ponieważ będzie ono stanowić punkt wyjścia do zdefiniowania bezpieczeństwa obliczeniowego w następnych rozdziałach. Rzeczywiście, w dalszej części książki będziemy często używać tego rodzaju eksperymentów do definiowania bezpieczeństwa.

W tym kontekście rozważymy następujący eksperiment: przeciwnik A najpierw określa dwie dowolne wiadomości $m_0, m_1 \in M$. Jedna z nich

wiadomości są wybierane jednolicie losowo i szyfrowane przy użyciu losowego klucza; powstały tekst zaszyfrowany jest przekazywany A. Na koniec A wysyła „zgadnij”, która z dwóch wiadomości została zaszyfrowana; A udaje się, jeśli odgadnie poprawnie.

Schemat szyfrowania jest całkowicie nie do odróżnienia, jeśli żaden przeciwnik A nie może odnieść sukcesu z prawdopodobieństwem większym niż 1/2. (Należy zauważać, że w przypadku dowolnego schematu szyfrowania A może odnieść sukces z prawdopodobieństwem 1/2, wyprowadzając jednolite przypuszczenie; warunkiem jest po prostu to, że żaden atakujący nie może zrobić nic lepszego.) Podkreślamy, że nie nakłada się żadnych ograniczeń na możliwości obliczeniowe moc A.

Formalnie niech $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ będzie schematem szyfrowania z przestrzenią komunikatów M. Niech A będzie przeciwnikiem, co formalnie jest po prostu algorytmem (stanowym).

Definiujemy eksperyment PrivKeav w następujący sposób:

Eksperyment nieodróżnialności kontradykcyjnej $\text{PrivKeav}_{A,\Pi}$: 1. Przeciwnik A wysyła

parę komunikatów $m_0, m_1 \in M$.

2. Za pomocą Gen generowany jest klucz k i wybierany jest jednolity bit $b \in \{0, 1\}$.

Zaszyfrowany tekst $c = \text{Enc}_k(m_b)$ jest obliczany i przekazywany A. c nazywamy zaszyfrowanym tekstem wyzwania.

3. A wyprowadza trochę b.

4. Wynik eksperymentu definiuje się jako 1, jeśli $b = b$, i 0 w przeciwnym razie.

Zapisujemy $\text{PrivKeav} = 1$, jeśli wynik eksperymentu A, Π wynosi 1 i w tym przypadku mówimy, że A się udało.

Jak zauważono wcześniej, dla A odniesienie sukcesu z prawdopodobieństwem 1/2 w wyniku losowego odgadnięcia cia jest trywialne. Doskonała nierozróżnialność wymaga, aby jakiekolwiek A nie mogło działać lepiej.

DEFINICJA 2.5 Schemat szyfrowania $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ z przestrzenią wiadomości M jest całkowicie nierozróżnialny, jeśli dla każdego A zachodzi to

$$\Pr_{A, \Pi}[\text{PrivKeav} = 1] = \frac{1}{2}.$$

Poniższy lemat stwierdza, że Definicja 2.5 jest równoważna Definicji 2.3. Dowód lematu pozostawiamy jako ćwiczenie 2.5.

LEMMA 2.6 Schemat szyfrowania Π jest całkowicie tajny wtedy i tylko wtedy, gdy jest całkowicie nie do odróżnienia.

Przykład 2.7

Pokazujemy, że szyfr Vigen`ere'a nie jest całkowicie nierozróżnialny, przynajmniej dla niektórych parametrów. Konkretnie, niech Π oznacza szyfr Vigen`ere'a dla przestrzeni komunikatów dwuznakowych ciągów znaków, gdzie okres jest wybrany jednolicie w $\{1, 2\}$. Aby pokazać, że Π nie jest całkowicie nieodróżnialne, pokazujemy przeciwnika A, dla którego $\Pr_{A, \Pi}[\text{PrivKeav} = 1] > \Pr_{A, \Pi}[\text{PrivKeav} = 0]$

Przeciwnik A robi:

1. Wyjście $m_0 = aa$ i $m_1 = ab$.
2. Po otrzymaniu szyfrogramu wezwania $c = c_1c_2$ wykonaj następujące czynności: jeśli $c_1 = c_2$ wyprowadź 0; inaczej wyjście 1.

Obliczanie $\Pr[\text{PrivKeav}] = 1$ jest złożone, ale proste.

$$\begin{aligned} \Pr[\text{PrivKeav}] &= \frac{1}{22} \cdot \Pr[\text{PrivKeav} | b = 0] + \frac{1}{22} \cdot \Pr[\text{PrivKeav} | b = 1] \\ &= \frac{1}{22} \cdot \Pr[A \text{ wyjście } 0 | b = 0] + \frac{1}{22} \cdot \Pr[A \text{ wyjście } 1 | b = 1], \quad (2.2) \end{aligned}$$

gdzie b jest bitem jednolitym określającym, która wiadomość zostanie zaszyfrowana. A wyprowadza 0 wtedy i tylko wtedy, gdy dwa znaki tekstu zaszyfrowanego $c = c_1c_2$ są równe. Gdy $b = 0$ (więc $c_0 = aa$ jest szyfrowane), to $c_1 = c_2$, jeśli albo (1) zostanie wybrany klucz z okresu 1, albo (2) zostanie wybrany klucz z okresu 2, a oba znaki klucza będą równe. To pierwsze wstępuje z prawdopodobieństwem, a to drugie z prawdopodobieństwem

$\frac{1}{22} \cdot \frac{1}{26}$. Więc

$$\Pr[A \text{ wyjście } 0 | b = 0] = \frac{1}{22} \cdot \frac{1}{26} = 0,52.$$

Gdy $b = 1$ wtedy $c_1 = c_2$ tylko wtedy, gdy wybrany zostanie klucz z okresu 2 i pierwszy znak klucza jest o jeden więcej niż drugi znak klucza, co dzieje się z prawdopodobieństwem
 $\frac{1}{22} \cdot \frac{1}{26}$. Więc

$$\Pr[A \text{ wyjście } 1 | b = 1] = \frac{1}{2} \cdot \Pr[A \text{ wyjście } 0 | b = 1] = \frac{1}{2} \cdot \frac{1}{26} = 0,98.$$

Podstawienie do równania (2.2) daje

$$\Pr[\text{PrivKeav}] = \frac{1}{22} \cdot \frac{1}{26} + \frac{1}{22} \cdot \frac{1}{26} = \frac{1}{22} = 0,75 > 2 \cdot \frac{1}{26},$$

a schemat nie jest całkowicie nie do odróżnienia.

2.2 Jednorazowa podkładka

W 1917 roku Vernam opatentował całkowicie tajny schemat szyfrowania, zwany obecnie blokiem jednorazowym. W czasie, gdy Vernam zaproponował ten plan, nie było dowodów na to, że był on całkowicie tajny; w rzeczywistości nie było jeszcze pojęcia, czym jest doskonała tajemnica. Jednak około 25 lat później Shannon wprowadził definicję doskonalej tajemnicy i wykazał, że jednorazowy notes zapewnia taki poziom bezpieczeństwa.

KONSTRUKCJA 2.8

Napraw liczbę całkowitą > 0 . Przestrzeń wiadomości M , przestrzeń kluczy K i przestrzeń tekstu zaszyfrowanego C są równe $\{0, 1\}$ (zbiór wszystkich ciągów

- binarnych o długości l . • Gen: algorytm generowania klucza wybiera klucz spośród $K = \{0, 1\}^l$ zgodnie z rozkładem równomiernym (tzn. każdy z 2^l ciągów w przestrzeni jest wybierany jako klucz z prawdopodobieństwem dokładnie 2^{-l}).
- Enc: mając klucz $k \in \{0, 1\}^l$ i wiadomość $m \in \{0, 1\}^l$, algorytm szyfrowania wyprowadza tekst zaszyfrowany $c := k \oplus m$.
- Dec: mając klucz $k \in \{0, 1\}^l$ i tekst zaszyfrowany $c \in \{0, 1\}^l$, algorytm deszyfrujący wysyła wiadomość $m := k \oplus c$.

Schemat jednorazowego szyfrowania padu.

Opisując schemat, pozwalamy $a \oplus b$ oznaczać bitową wyłączność-lub (XOR) dwóch ciągów binarnych $a \oplus b$ (tzn. jeśli $a = a_1 \dots a_l$ i $b = b_1 \dots b_l$ są ciągami -bitowymi, to $a \oplus b$ jest ciągiem -bitowym podanym przez $a_1 \oplus b_1 \dots a_l \oplus b_l$). W schemacie szyfrowania jednorazowego kluczem jest jednolity串 o tej samej długości co wiadomość; zaszyfrowany串 jest obliczany poprzez proste XORowanie klucza i wiadomości. Formalna definicja jest podana jako Konstrukcja 2.8. Zanim omówimy bezpieczeństwo, najpierw sprawdzamy poprawność: dla każdego klucza k i każdej wiadomości m obowiązuje $Deck(Enc_k(m)) = k \oplus m = m$, a zatem jednorazowa podkładka stanowi prawidłowy schemat szyfrowania.

Idealną tajność szyfru jednorazowego można łatwo udowodnić wykorzystując Lemat 2.4 oraz fakt, że szyfrogram jest równomiernie rozłożony niezależnie od tego, jaką wiadomość jest szyfrowana. Dowód przedstawiamy bezpośrednio w oparciu o pierwotną definicję .

TWIERDZENIE 2.9 Schemat jednorazowego szyfrowania podkładki jest całkowicie tajny.

DOWÓD Najpierw obliczamy $\Pr[C = c | M = m]$ dla dowolnego $c \in C$ i $m \in M$. W przypadku podkładki jednorazowej

$$\begin{aligned}\Pr[C = c | M = m] &= \Pr[Enc_k(m) = c] = \Pr[m \oplus K = c] \\ &= \Pr[K = m \oplus c] \\ &= 2^{-l},\end{aligned}$$

gdzie zachodzi ostateczna równość, ponieważ klucz K jest jednolitym ciągiem bitowym. Napraw dowolny rozkład na M . Dla dowolnego $c \in C$ mamy

$$\begin{aligned}\Pr[C = c] &= \Pr[C = c | M = m] \cdot \Pr[M = m] \\ &= 2^{-l} \cdot \Pr[M = m] \\ &= 2^{-l},\end{aligned}$$

gdzie suma jest większa od $m - M$ przy $\Pr[M = m] = 0$. Twierdzenie Bayesa podaje:

$$\begin{aligned}\Pr[M = m \mid C = \text{do}] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{2^{-\frac{m}{k}} \cdot \Pr[M = m]}{2^{-\frac{m}{k}}} \\ &= \Pr[M = m].\end{aligned}$$

Dochodzimy do wniosku, że jednorazowy notes jest całkowicie tajny. ■

Jednorazowa podkładka była używana przez kilka agencji wywiadu narodowego w połowie XX wieku do szyfrowania wrażliwego ruchu. Być może najbardziej znany jest fakt, że „czerwony telefon” łączący Biały Dom z Kremllem podczas zimnej wojny był chroniony za pomocą jednorazowego szyfrowania, podczas którego rządy USA i ZSRR wymieniały niezwykle długie klucze za pośrednictwem zaufanych kurierów niosących papierowe teczkę na jakie losowe znaki zostały zapisane.

Niezależnie od powyższego, jednorazowe szyfrowanie podkładki jest już rzadko stosowane ze względu na szereg wad. Najważniejsze jest to, że klucz jest tak długi, jak wiadomość.

² Ogranicza to użyteczność schematu do wysyłania bardzo długich wiadomości (ponieważ bezpieczne udostępnianie i przechowywanie bardzo długiego klucza może być trudne) i jest problematyczne, gdy strony nie mogą z góry przewidzieć (górną granicą), jak długo będzie trwała wiadomość.

Co więcej, podkładka jednorazowa – jak sama nazwa wskazuje – jest bezpieczna tylko wtedy, gdy zostanie użyta raz (tym samym kluczem). Chociaż nie zdefiniowaliśmy jeszcze pojęcia tajemnicy w przypadku szyfrowania wielu wiadomości, łatwo zauważać, że szyfrowanie więcej niż jednej wiadomości tym samym kluczem powoduje wyciek wielu informacji. W szczególności powiedzmy, że dwie wiadomości m_1, m_2 są szyfrowane przy użyciu tego samego klucza k .

Przeciwnik, który otrzyma $c = m_1 \oplus k$ i $c = m_2 \oplus k$, może obliczyć

$$\text{do} \quad \text{do} = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$$

i w ten sposób dowiesz się, na czym polega wyłączność lub z dwóch wiadomości lub, równoważnie, dokładnie, gdzie te dwie wiadomości się różnią. Chociaż może to nie wydawać się zbyt istotne, wystarczy wykluczyć wszelkie twierdzenia o całkowitej tajemnicy podczas szyfrowania dwóch wiadomości przy użyciu tego samego klucza. Co więcej, jeśli komunikaty odpowiadają tekstowi w języku naturalnym, to biorąc pod uwagę wyłączność lub dwa wystarczająco długie komunikaty, możliwe jest przeprowadzenie analizy częstotliwości (jak w poprzednim rozdziale, choć bardziej złożonej) i odzyskanie samych komunikatów. Ciekawym historycznym przykładem jest projekt VENONA, w ramach którego Stanom Zjednoczonym i Wielkiej Brytanii udało się odszyfrować teksty zaszyfrowane wysypane przez Związek Radziecki, które zostałyomyłkowo zaszyfrowane za pomocą powtarzających się fragmentów jednorazowego bloku na prz.

²Nie czyni to jednorazowej klawiatury bezużyteczną, ponieważ dwóm stronom może być łatwiej dzielić się kluczem w pewnym momencie, zanim znana będzie wiadomość, która ma zostać przekazana.

2.3 Ograniczenia doskonałej tajemnicy

Zakończyliśmy poprzednią sekcję zwróceniem uwagi na pewne wady schematu jednorazowego szyfrowania padu. Tutaj pokazujemy, że te wady nie są specyficzne dla tego schematu, ale raczej są nieodłącznymi ograniczeniami doskonałej tajemnicy. W szczególności udowadniamy, że każdy doskonale tajny schemat szyfrowania musi mieć przestrzeń kluczów co najmniej tak dużą jak przestrzeń wiadomości. Jeśli wszystkie klucze mają tę samą długość, a przestrzeń wiadomości składa się ze wszystkich ciągów znaków o określonej długości, oznacza to, że klucz jest co najmniej tak długi, jak wiadomość. W szczególności długość klucza jednorazowej podkładki jest optymalna. (Drugie ograniczenie —mianowicie to, że klucz można użyć tylko raz —jest również nieodłączne, jeśli wymagana jest całkowita tajemnica; patrz ćwiczenie 2.13.)

TWIERDZENIE 2.10 Jeżeli (Gen, Enc, Dec) jest doskonale tajnym schematem szyfrowania z przestrzenią wiadomości M i przestrzenią kluczów K , to $|K| \geq |M|$.

DOWÓD Pokazujemy, że jeśli $|K| < |M|$ w takim razie plan nie może być całkowicie tajny. Założymy, że $|K| < |M|$. Rozważ równomierny rozkład w M i niech $c \in C$ będzie szyfrogramem występującym z niezerowym prawdopodobieństwem. Niech $M(c)$ będzie zbiorem wszystkich możliwych wiadomości, które są możliwymi deszyfrowaniami c ; to jest

$$M(c) \stackrel{\text{def}}{=} \{m \mid m = \text{pokład}(c) \text{ dla pewnego } k \in K\}.$$

Jasne $|M(c)| \leq |K|$. (Przypomnijmy, że możemy założyć, że Dec jest deterministyczny.) Jeśli $|K| < |M|$, istnieje pewne $m \in M$ takie, że $m \in M(c)$. Ale wtedy

$$\Pr[M = m \mid C = c] = 0 = \Pr[M = m],$$

więc plan nie jest całkowicie tajny. ■

Idealna tajemnica przy krótszych kluczach? Powyższe twierdzenie pokazuje nieodłączne ograniczenie schematów zapewniających doskonąłą tajemnicę. Mimo to osoby czasami twierdzą, że opracowały radykalnie nowy schemat szyfrowania, który jest „nie do złamania” i zapewnia bezpieczeństwo jednorazowej podkładki bez użycia kluczów, o ile to, co jest szyfrowane. Powyższy dowód pokazuje, że twierdzenia takie nie mogą być prawdziwe; każdy, kto wysuwa takie twierdzenia, albo wie bardzo mało o kryptografii, albo rażąco kłamie.

2.4 *Twierdzenie Shannona

W swojej pracy na temat doskonałej tajemnicy Shannon przedstawił także charakterystykę schematów szyfrowania doskonale tajnego. Charakterystyka ta mówi, że w pewnych warunkach algorytm generowania klucza Gen musi wybrać klucz w sposób jednolity ze zbioru wszystkich możliwych kluczy (jak w podkładce jednorazowej); co więcej, dla każdej wiadomości m i tekstu zaszyfrowanego c istnieje unikalny klucz mapujący m na c (znów, jak w bloku jednorazowym). Twierdzenie to jest poza tym, że jest interesujące samo w sobie, jest użytecznym narzędziem do udowadniania (lub obalania) całkowitej tajności sugerowanych schematów. Omówimy to dalej po dowodzie.

Twierdzenie podane tutaj zakłada $|M| = |K| = |C|$, co oznacza, że wszystkie zestawy tekstów jawnych, kluczy i tekstów zaszyfrowanych mają ten sam rozmiar. Widzieliśmy już, że dla doskonałej tajemnicy musimy mieć $|K| = |M|$. Łatwo zauważać, że prawidłowe odszyfrowanie wymaga $|C| = |M|$. Dlatego w pewnym sensie schematy szyfrowania z $|M| = |K| = |C|$ są „optymalne”.

TWIERDZENIE 2.11 (Twierdzenie Shannona) Niech $(\text{Gen}, \text{Enc}, \text{Dec})$ będzie schematem szyfrowania z przestrzenią wiadomości M , dla której $|M| = |K| = |C|$. Schemat jest całkowicie tajny wtedy i tylko wtedy, gdy:

1. Każdy klucz $k \in K$ jest wybierany z (równym) prawdopodobieństwem $1/|K|$ według algorytmu gen.
2. Dla każdego $m \in M$ i każdego $c \in C$ istnieje unikalny klucz $k \in K$ taki, że $\text{Enc}(m)$ wyrowadza c .

DOWÓD Intuicja stojąca za dowodem jest następująca. Aby zobaczyć, że podane warunki implikują całkowitą tajemnicę, zauważ, że warunek 2 oznacza, że dowolny zaszyfrowany tekst c może być wynikiem zaszyfrowania dowolnego możliwego tekstu jawnego m , ponieważ istnieje jakiś klucz k odwzorowujący m na c . Ponieważ istnieje unikalny taki klucz i każdy klucz jest wybierany z równym prawdopodobieństwem, obowiązuje pełna tajemnica, jak w przypadku jednorazowej podkładki. Z drugiej strony, doskonała tajemnica natychmiast oznacza, że dla każdego m i c istnieje co najmniej jeden klucz odwzorowujący m na c . Fakt, że $|M| = |K| = |C|$ oznacza ponadto, że na każde m i c przypada dokładnie jeden taki klucz. Biorąc to pod uwagę, każdy klucz musi zostać wybrany z równym prawdopodobieństwem, w przeciwnym razie nie uda się zachować doskonałej tajemnicy. Następnie formalny dowód.

Zakładamy dla uproszczenia, że Enc jest deterministyczny. (Można wykazać, że dzieje się to bez utraty ogólności.) Najpierw udowodnimy, że jeśli schemat szyfrowania spełnia warunki 1 i 2, to jest on całkowicie tajny. Dowód jest zasadniczo taki sam, jak dowód całkowitej tajemnicy dla jednorazowego notesu, więc będę dzielić się stosunkowo krótko. Ustal dowolne $c \in C$ i $m \in M$. Niech $k \in K$ będzie kluczem unikalnym, gwarantowanym przez warunek 2, dla którego $\text{Enc}(m) = c$. Następnie,

$$\Pr[C = c \mid M = m] = \Pr[K = k] = 1/|K|,$$

gdzie ostateczna równość zachodzi pod warunkiem 1. Zatem

$$\Pr_{m \in M}[C = c] = \Pr[EncK(m) = c] \cdot \Pr[M = m] = 1/|K|.$$

Dotyczy to dowolnego rozkładu na M . Zatem dla dowolnego rozkładu na M , dowolnego $m \in M$ z $\Pr[M = m] = 0$ i dowolnego $c \in C$ mamy:

$$\begin{aligned} \Pr[M = m \mid C = \text{do}] &= \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{\Pr[EncK(m) = c] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{|K|^{-1} \cdot \Pr[M = m]}{|K|^{-1}} = \Pr[M = m], \end{aligned}$$

a plan jest całkowicie tajny.

W drugim przypadku założymy, że schemat szyfrowania jest całkowicie tajny; pokazujemy, że zachodzą warunki 1 i 2. Ustal dowolne $c \in C$. Musi istnieć jakiś komunikat $m \in M$, dla którego $\Pr[EncK(m) = c] = 0$. Z lematu 2.4 wynika zatem, że $\Pr[EncK(m) = c] = 0$ dla każdego $m \in M$. Innymi słowy, jeśli pozwolimy, aby $M = \{m_1, m_2, \dots\}$, to dla każdego $m_i \in M$ mamy niepusty zbiór kluczy $K_i \subseteq K$ taki, że $EncK(m_i) = c$ wtedy i tylko wtedy, gdy $k_i \in K_i$. Co więcej, gdy $i = j$, wówczas K_i i K_j muszą być rozłączne, w przeciwnym razie poprawność nie będzie spełniona. Ponieważ $|K| = |M|$, widzimy, że każde K_i zawiera tylko jeden klucz k_i , zgodnie z wymogiem warunku 2. Lemat 2.4 pokazuje, że dla dowolnego $m_i \in M$ mamy

$$\Pr[K = k_i] = \Pr[EncK(m_i) = c] = \Pr[EncK(m_j) = c] = \Pr[K = k_j].$$

Ponieważ dotyczy to wszystkich $i, j \in M = |K|$ i $k_i = k_j$ dla $i = j$, oznacza to, że każdy klucz jest wybierany z prawdopodobieństwem $1/|K|$, zgodnie z wymogiem warunku 1. ■

Twierdzenie Shannona jest przydatne przy podejmowaniu decyzji, czy dany schemat jest całkowicie tajny. Warunek 1 jest łatwy do sprawdzenia, a warunek 2 można wykazać (lub mu zaprzeczyć) bez konieczności obliczania jakichkolwiek prawdopodobieństw (w przeciwieństwie do bezpośredniej pracy z Definicją 2.3). Na przykład doskonała tajemnica jednorazowego notesu jest banalna do udowodnienia przy użyciu twierdzenia Shannona. Podkreślamy jednak, że twierdzenie ma zastosowanie tylko wtedy, gdy $|M| = |K| = |C|$.

Referencje i dodatkowe lektury

Jednorazową podkładkę powszechnie przypisuje się Vernamowi [172], który złożył na nią patent, jednak ostatnie badania historyczne [25] pokazują, że została wynaleziona jakiś czas temu

35 lat wcześniej. Analiza notesu jednorazowego musiała poczekać na przełomową pracę Shannona [154], która wprowadziła pojęcie tajemnicy doskonałej.

W tym rozdziale przestudiowaliśmy doskonale tajne szyfrowanie. Niektóre inne problemy kryptograficzne można również rozwiązać za pomocą „doskonałego” bezpieczeństwa. Godnym uwagi przykładem jest problem uwierzytelniania wiadomości, którego celem jest uniemożliwienie przeciwnikowi (niewykwytalnej) modyfikacji wiadomości wysyłanej od jednej strony do drugiej. Przeanalizujemy ten problem szczegółowo w Rozdziale 4, omawiając „idealnie bezpieczne” uwierzytelnianie wiadomości w Podrozdziale 4.6.

Ćwiczenia

2.1 Udowodnij, że redefiniując przestrzeń kluczy, możemy założyć, że algorytm generowania klucza Gen wybiera klucz z przestrzeni kluczy równomiernie i losowo, bez zmiany $\Pr[C = c | M = m]$ dla dowolnego m, c .

Wskaźówka: Zdefiniuj przestrzeń kluczową jako zbiór wszystkich możliwych losowych taśm dla algorytmu losowego Gen.

2.2 Udowodnić, że poprzez redefinicję przestrzeni kluczy możemy założyć, że Enc jest deterministyczny, bez zmiany $\Pr[C = c | M = m]$ dla dowolnego m, c .

2.3 Udowodnić lub obalić: Schemat szyfrowania z przestrzenią komunikatów M jest całkowicie tajny wtedy i tylko wtedy, gdy dla każdego rozkładu prawdopodobieństwa w M i każdego $c_0, c_1 \in C$ mamy $\Pr[C = c_0] = \Pr[C = c_1]$.

2.4 Udowodnić drugi kierunek lematu 2.4.

2.5 Udowodnij lemat 2.6.

2.6 Dla każdego z poniższych schematów szyfrowania określ, czy jest on całkowicie tajny. W każdym przypadku uzasadnij swoją odpowiedź.

(a) Przestrzeń wiadomości to $M = \{0, \dots, 4\}$. Algorytm Gen wybiera klucz jednolity z przestrzeni kluczy $\{0, \dots, 5\}$. Enck(m) zwraca $[k + m \bmod 5]$, a Deck(c) zwraca $[c \bmod 5]$. (b) Przestrzeń wiadomości to $M = \{m \in \{0, 1\} \mid$
ostatni bit m to 0).

Gen wybiera jednolity klucz z $\{0, 1\}$ ¹. Enck(m) zwraca tekst zaszyfrowany $m \oplus k_0$, a Deck(c) zwraca $c \oplus k_0$.

2.7 Używając klawiatury jednorazowej z klawiszem $k = 0$ mamy $\text{Enck}(m) = k \oplus m = m$ i wiadomość jest wysyłana jawnie! Dlatego zasugerowano modyfikację jednorazowego uzupełnienia poprzez szyfrowanie jedynie przy użyciu $k = 0$ (tj. aby Gen wybierał k równomiernie ze zbioru niezerowych kluczy o długości n). Czy ten zmodyfikowany plan jest nadal całkowicie tajny? Wyjaśnić.

2.8 Niech Π oznacza szyfr Vigen`ere'a, w którym przestrzeń wiadomości składa się ze wszystkich 3-znakowych ciągów znaków (nad alfabetem angielskim), a klucz jest generowany poprzez najpierw wybór okresu t równomiernie z $\{1, 2, 3\}$, a następnie pozwolenie kluczem bę dzie jednolity ciąg o długości t.

(a) Zdefiniuj A w następujący sposób: A wyprowadza $m_0 = aab$ i $m_1 = abb$. Po podaniu szyfrogramu c zwraca 0, jeśli pierwszy znak c jest taki sam, jak drugi znak c, a w przeciwnym razie wyprowadza 1. Oblicz $\Pr[\text{PrivKeav} \mid A, \Pi]$. Skonstruuj i przeanalizuj przeciwnika A, dla $\Pr[\text{PrivKeav} \mid A, \Pi] = 1$.

którego $\Pr[\text{PrivKeav} \mid A, \Pi] = 1$
jest więc ksa niż odpowiedź z czę ści (a).

2.9 W tym ćwiczeniu przyjrzymy się różnym warunkom, w których przesunięcie, podstawienie monoalfabetyczne i szyfry Vigenere' a są doskonale tajne:

- (a) Udowodnić, że jeśli szyfrowany jest tylko jeden znak, to szyfr przesuwny jest całkowicie tajny.
- (b) Jaka jest najwięcej ksa przestrzeń komunikatów M, dla której występuje monoalfabetyczna szyfrowanie zapewnia doskonałą tajemnicę?
- (c) Udowodnić, że szyfr Vigen`ere'a wykorzystujący (stały) okres t jest całkowicie tajny, gdy jest używany do szyfrowania wiadomości o długości t.

Pogódź to z atakami pokazanymi w poprzednim rozdziale.

2.10 Udowodnić, że schemat spełniający definicję 2.5 musi mieć $|K| \geq |M|$ bez użycia Lematu 2.4. W szczególności niech Π bę dzie dowolnym schematem szyfrowania z $|K| < |M|$. Pokaż, dla którego $\Pr[\text{PrivKeav} \mid A, \Pi] = 1 > \frac{1}{|T^2|}$.

Wskazówka: Łatwiej może być losowanie A.

2.11 Założmy, że wymagamy jedynie, aby schemat szyfrowania (Gen, Enc, Dec) z przestrzenią komunikatów M spełniał następujące wymagania: Dla wszystkich $m \in M$ mamy $\Pr[\text{DecK}(\text{EncK}(m)) = m] = 1$ dla dowolnego klucza K . (To prawdopodobieństwo przejmuje wybór klucza jako również losowość zastosowana podczas szyfrowania.) Pokaż, że idealną tajemnicę można osiągnąć za pomocą $|K| < |M|$ gdy $t = 1$. Udowodnić dolną granicę wielkości K wyrażoną w t.

2.12 Niech $\epsilon > 0$ bę dzie stałą. Założmy, że schemat szyfrowania jest ϵ -idealnie tajny jeśli dla każdego przeciwnika A tak jest

$$\Pr[\text{PrivKeav} \mid A, \Pi] = \frac{1}{2} + \epsilon.$$

(Porównaj z definicją 2.5.) Pokaż, że ϵ -doskonałą tajemnicę można osiągnąć za pomocą $|K| < |M|$ gdy $\epsilon > 0$. Udowodnić dolną granicę wielkości K wyrażoną jako ϵ .

2.13 W tym zadaniu rozważamy definicje całkowitej tajemnicy przy szyfrowaniu dwóch wiadomości (przy użyciu tego samego klucza). Rozważamy tutaj rozkłady na pary wiadomości z przestrzeni wiadomości M ; pozwalamy M_1, M_2 są zmiennymi losowymi oznaczającymi odpowiednio pierwszy i drugi komunikat. (Podkreślamy, że nie zakłada się, że te zmienne losowe tak mają być niezależne.) Generujemy (pojedynczy) klucz k , próbujemy parę komunikatów (m_1, m_2) zgodnie z podanym rozkładem, a następnie obliczamy szyfrrogramy $c_1 = \text{Enc}_k(m_1)$ i $c_2 = \text{Enc}_k(m_2)$; indukuje to rozkład na pary szyfrrogramów i pozwalamy, aby C_1, C_2 były odpowiadającymi zmiennej losowej.

- (a) Powiedzmy, że schemat szyfrowania (Gen , Enc , Dec) jest całkowicie tajny dla dwojga komunikatów, jeśli dla wszystkich rozkładów po $M \times M$, wszystkie $m_1, m_2 \in M$, oraz wszystkie szyfrrogramy $c_1, c_2 \in C$ z $\Pr[C_1 = c_1 \cap C_2 = c_2] > 0$:

$$\begin{aligned} \Pr[M_1 = m_1 \cap M_2 = m_2 \mid C_1 = c_1 \cap C_2 = c_2] \\ = \Pr[M_1 = m_1 \cap M_2 = m_2]. \end{aligned}$$

Udowodnić, że żaden schemat szyfrowania nie spełnia tej definicji.

Wskazówka: weź $c_1 = c_2$.

- (b) Powiedzmy, że schemat szyfrowania (Gen , Enc , Dec) jest całkowicie tajny dla dwojga odrębnych komunikatów, jeśli dla wszystkich rozkładów w $M \times M$, gdzie gwarantuje się, że pierwsza i druga wiadomość będą różne (tj. rozkłady na pary różnych wiadomości), wszystkie $m_1, m_2 \in M$ i wszystkie $c_1, c_2 \in C$ z $\Pr[C_1 = c_1 \cap C_2 = c_2] > 0$:

$$\begin{aligned} \Pr[M_1 = m_1 \cap M_2 = m_2 \mid C_1 = c_1 \cap C_2 = c_2] \\ = \Pr[M_1 = m_1 \cap M_2 = m_2]. \end{aligned}$$

Pokaż schemat szyfrowania, który w sposób możliwy do udowodnienia spełnia tę definicję.

Wskazówka: Proponowany przez Ciebie schemat szyfrowania nie musi być skuteczny, chociaż możliwe jest skuteczne rozwiązanie.

część druga

Klucz prywatny (symetryczny)

Kryptografia

Machine Translated by Google

Rozdział 3

Szyfrowanie kluczem prywatnym

W poprzednim rozdziale widzieliśmy pewne podstawowe ograniczenia doskonałej tajemnicy. W tym rozdziale rozpoczynamy nasze badania współczesnej kryptografii od wprowadzenia słabszego (ale wystarczającego) pojęcia tajemnicy obliczeniowej. Następnie pokażemy, jak można wykorzystać tę definicję do oznaczenia pokazanych wcześniej wyników niemożliwości, a w szczególności, jak można użyć krótkiego klucza (powiedzmy o długości 128 bitów) do szyfrowania wielu długich wiadomości (powiedzmy łącznie gigabajtów).

Po drodze przestudiujemy podstawowe pojęcie pseudolosowości, które oddaje pogląd, że coś może „wyglądać” na całkowicie losowe, nawet jeśli tak nie jest. Ta potężna koncepcja leży u podstaw współczesnej kryptografii i ma zastosowania i implikacje także poza tą dziedziną.

3.1 Bezpieczeństwo obliczeniowe

W Rozdziale 2 wprowadziliśmy pojęcie doskonałej tajemnicy. Chociaż doskonała tajemnica jest wartościowym celem, jest również niepotrzebnie silna. Doskonała tajemnica wymaga, aby żadna informacja o zaszyfrowanej wiadomości nie wyciekła, nawet do osoby podsłuchującej z nieograniczoną mocą obliczeniową. Jednakże ze względu dów przaktycznych schemat szyfrowania nadal byłby uważany za bezpieczny, gdyby powodował wyciek jedynie niewielkiej ilości informacji do osób podsłuchujących o ograniczonej mocy obliczeniowej. Na przykład schemat polegający na wycieku informacji z prawdopodobieństwem co najwyżej 2-60 do podsłuchujących inwestujących do 200 lat obliczeń w najszybszy dostępny superkomputer jest odpowiedni dla każdego zastosowania w świecie rzeczywistym. Definicje bezpieczeństwa, które uwzględniają ograniczenia obliczeniowe atakującego i dopuszcza małe prawdopodobieństwo niepowodzenia, nazywane są obliczeniowymi, aby odróżnić je od pojęcia (takich jak doskonała tajemnica) o charakterze informacyjnym. Bezpieczeństwo obliczeniowe jest obecnie de facto sposobem definiowania bezpieczeństwa we wszystkich celach kryptograficznych.

Podkreślamy, że choć rezygnujemy z uzyskania doskonałego bezpieczeństwa, nie oznacza to, że rezygnujemy z rygorystycznego podejścia matematycznego. Definicje i dowody są nadal istotne, a jedną różnicą jest to, że obecnie rozważamy słabsze (ale wciąż znaczące) definicje bezpieczeństwa.

Bezpieczeństwo obliczeniowe obejmuje dwa zagadzenia w odniesieniu do informacji:

teoretyczne koncepcje bezpieczeństwa (w przypadku szyfrowania oba te złagodzenia są konieczne, aby wyjść poza ograniczenia doskonałej tajemnicy omówione w poprzednim rozdziale):

1. Bezpieczeństwo jest gwarantowane tylko w przypadku skutecznych przeciwników, którzy działają przez określony czas. Oznacza to, że mając wystarczająco dużo czasu (lub wystarczających zasobów obliczeniowych), atakujący może być w stanie złamać bezpieczeństwo. Jeśli uda nam się zwiąkszyć zasoby wymagane do złamania schematu od tych, którymi dysponuje jakkolwiek realistyczny atakujący, wówczas z praktycznych względów schemat będzie nie do złamania.
2. Przeciwnicy mogą potencjalnie odnieść sukces (tj. bezpieczeństwo może potencjalnie zanieść) z bardzo małym prawdopodobieństwem. Jeśli uda nam się zmniejszyć to prawdopodobieństwo, nie musimy się tym martwić.

Aby uzyskać sensowną teorię, musimy precyzyjnie zdefiniować powyższe relaksacje. Istnieją dwa ogólne podejścia do tego: podejście konkretne i podejście asymptotyczne. Zostały one opisane dalej.

3.1.1 Konkretnie podejście

Konkretnie podejście do bezpieczeństwa obliczeniowego określa ilościowo bezpieczeństwo schematu kryptograficznego poprzez wyraźne ograniczenie maksymalnego prawdopodobieństwa sukcesu dowolnego (randomizowanego) przeciwnika działającego przez określony czas lub, dokładniej, inwestującą określoną ilość wysiłku obliczeniowego.

Zatem konkretna definicja bezpieczeństwa przyjmuje z grubsza następującą formę :

Schemat jest (t, ϵ) -pewny, jeśli któremukolwiek przeciwnikowi działającemu przez czas co najwyżej t uda się złamać schemat z prawdopodobieństwem co najwyżej ϵ .

(Oczywiście powyższe służy jedynie jako ogólny szablon i aby powyższe stwierdzenie miało sens, musimy dokładnie zdefiniować, co to znaczy „złamać” dany schemat.) Jako przykład można mieć schemat z zagwarantowaną, że żadnemu przeciwnikowi działającemu przez co najwyżej 200 lat i korzystającemu z najszybszego dostępu superkomputera nie uda się złamać schematu z prawdopodobieństwem więcej niż 2^{-60} . Lub też wygodniejsze może być zmierzenie czasu działania w kategoriach cykli procesora i skonstruowanie takiego schematu, aby żaden przeciwnik używający co najwyżej 280 cykli nie mógł złamać schematu z prawdopodobieństwem więcej niż 2^{-60} .

Pouczające jest wyczucie dużych wartości t i małych wartości ϵ , które są typowe dla współczesnych schematów kryptograficznych.

Przykład 3.1

Ogólnie przyjmuje się, że współczesne schematy szyfrowania klucza prywatnego zapewniają niemal optymalne bezpieczeństwo w następującym sensie: gdy klucz ma długość n – a więc c przestrzeń na klucze ma rozmiar 2^n – przeciwnik działa przez czas t (mierzony, powiedzmy, w komputerze cykli) udaje się przełamać schemat co najwyżej z prawdopodobieństwem

$ct/2 n$ dla pewnej stałej stałej c . (Odpowiada to po prostu przeszukiwaniu przestrzeni klucza metodą brute-force i zakłada, że nie przeprowadzono żadnego przetwarzania wstępnego.)

Zakładając dla uproszczenia $c = 1$, klucz o długości $n = 60$ zapewnia odpowiednie zabezpieczenie przed przeciwnikiem korzystającym z komputera stacjonarnego. Rzeczywiście, na procesorze 4 GHz (który wykonuje 4×10^9 cykli na sekundę) 260 cykli procesora wymaga $2^{60}/(4 \times 10^9)$ sekund, czyli około 1 lat. Jednak najszybszy superkomputer w chwili pisania tego tekstu może wykonać około 2×10^{16} operacji zmiennoprzecinkowych na sekundę, a 260 takich operacji zajmuje na takim komputerze tylko około 1 minuty. Bardziej rozważnym wyborem byłoby przyjąć $c = 80$; nawet wspomniany komputer potrzebowałby około 2 lat na wykonanie 280 operacji.

(Powyższe liczby służą wyłącznie celom ilustracyjnym; w praktyce $c > 1$ i kilka innych czynników – takich jak czas potrzebny na dostęp p do pamięci i możliwość równoległych obliczeń w sieci komputerów – znaczco wpływają na skuteczność ataków brute-force.)

Jednak obecnie zalecana długość klucza może wynosić $n = 128$. Różnica między 280 a 2128 jest mnożnikiem wynoszącym 248. Aby zorientować się, jak duża to jest, zauważ, że według szacunków fizyków liczba sekund ponieważ Wielki Wybuch był rzędu 258.

Jeśli prawdopodobieństwo, że atakującemu uda się zdyskać zaszyfrowaną wiadomość w ciągu jednego roku, wynosi co najwyżej 2–60, wówczas znacznie bardziej prawdopodobne jest, że nadawca i odbiorca zostaną uderzeni piorunem w tym samym czasie. Można z grubsza oszacować, że zdarzenie, które ma miejsce raz na sto lat, wystąpi z prawdopodobieństwem 2–30 w dowolnej sekundzie. Coś, co zachodzi z prawdopodobieństwem 2–60 w dowolnej sekundzie, jest 230 razy mniej prawdopodobne i można się spodziewać, że wystąpi mniej więcej tej samej radości na 100 miliardów lat.

Konkretnie podejście jest ważne w praktyce, ponieważ konkretne gwarancje są tym, czym ostatecznie interesują się użytkownicy schematu kryptograficznego. Trudno jednak zapewnić dokładne, konkretne gwarancje. Ponadto należy zachować ostrożność przy interpretacji konkretnych roszczeń dotyczących bezpieczeństwa. Na przykład twierdzenie, że żaden przeciwnik działający przez 5 lat nie jest w stanie złamać danego schematu z prawdopodobieństwem mniej niż ϵ , nasuwa pytanie: jakiego rodzaju moc obliczeniowa (np. komputer stacjonarny, superkomputer, sieć setek komputerów) zakłada to? Czy uwzględnia to przyszły postęp p w mocy obliczeniowej (która zgodnie z prawem Moore'a podwaja się mniej więcej co 18 miesięcy)? Czy szacunki zakładają wykorzystanie „gotowych” algorytmów, czy też dedykowanych implementacji oprogramowania zoptymalizowanych pod kątem ataku? Co mniej więcej, taka gwarancja niewiele mówi o prawdopodobieństwie powodzenia przeciwnika działającego przez 2 lata (poza faktem, że może ona wynosić co najwyżej ϵ) i nie mówi nic o prawdopodobieństwie sukcesu przeciwnika działającego przez 10 lat.

3.1.2 Podejście asymptotyczne

Jak częściej zauważono powyżej, istnieją pewne trudności techniczne i teoretyczne w stosowaniu podejścia opartego na bezpieczeństwie betonu. Kwestie te należy uregulować w

praktyka, ale gdy konkretne bezpieczeństwo nie jest bezpośrednio przedmiotem zainteresowania, wygodniej jest zamiast tego zastosować asymptotyczne podejście do bezpieczeństwa; takie właśnie podejście przyjęto w tej książce. Podejście to, zakorzenione w teorii złożoności, wprowadza parametr bezpieczeństwa o wartości całkowitej (oznaczony przez n), który parametryzuje zarówno schematy kryptograficzne, jak i wszystkie zaangażowane strony (mianowicie strony uczciwe oraz atakującego). Kiedy uczciwe strony inicjują schemat (tj. generują klucze), wybierają wartość n dla parametru bezpieczeństwa; na potrzeby tej dyskusji można myśleć o parametrze bezpieczeństwa jako odpowiadającym długości klucza. Zakkadza się, że parametr bezpieczeństwa jest znany każdemu przeciwnikowi atakującemu schemat, a teraz czas działania przeciwnika, a także prawdopodobieństwo jego powodzenia, postrzegamy jako funkcje parametru bezpieczeństwa, a nie konkretne liczby. Następnie:

1. „Efektywnych przeciwników” utożsamiamy z randomizowanymi (tj. probabilistycznymi) algorytmami działającymi w wielomianu czasu w n . Oznacza to, że istnieje pewien wielomian p taki, że przeciwnik działa przez czas co najwyżej $p(n)$, gdy parametrem bezpieczeństwa jest n . Wymagamy również – aby zapewnić efektywność w świecie rzeczywistym – aby uczciwe strony działały w czasie wielomianowym, chociaż podkreślamy, że przeciwnik może być znacznie potężniejszy (i działać znacznie dłużej niż) uczciwe strony.
2. Pojęcie „małych prawdopodobieństw sukcesu” utożsamiamy z prawdopodobieństwami sukcesu mniejszymi niż jakikolwiek odwrotny wielomian w n (patrz Definicja 3.4). Takie prawdopodobieństwa nazywane są znikomymi.

Niech ppt oznacza „probabilistyczny czas wielomianowy”. Definicja bezpieczeństwa asymptotycznego przyjmuje wówczas następującą ogólną postać:

Schemat jest bezpieczny, jeśli którymkolwiek przeciwnikowi ppt uda się złamać schemat z co najwyżej znikomym prawdopodobieństwem.

To pojęcie bezpieczeństwa jest asymptotyczne, ponieważ bezpieczeństwo zależy od zachowania schematu dla wystarczająco dużych wartości n . Poniższy przykład wyjaśnia to.

Przykład 3.2

Załóżmy, że mamy schemat asymptotycznie bezpieczny. Wtedy może to być przypadek, że przeciwnikowi działającemu przez $n^{\frac{3}{2}}$ minut uda się „złamać schemat” z prawdopodobieństwem $240 \cdot 2^{-n}$ (co jest zaniedbywalną funkcją n). Gdy $n = 40$ oznacza to, że przeciwnik działający przez 403 minuty (około 6 tygodni) może złamać schemat z prawdopodobieństwem 1, zatem takie wartości n nie są zbyt przydatne. Nawet dla $n = 50$ przeciwnik działający przez 503 minuty (około 3 miesięcy) może złamać schemat z prawdopodobieństwem około 1/1000, co może być nie do przyjęcia. Z drugiej strony, gdy $n = 500$, przeciwnik działający przez 200 lat łamie schemat tylko z prawdopodobieństwem około 2-500.

Jak pokazał poprzedni przykład, możemy postrzegać parametr bezpieczeństwa jako mechanizm, który pozwala uczciwym stronom „dostroić” bezpieczeństwo schematu do pożdanego poziomu. (Zwięk kszenie parametru bezpieczeństwa zwię ksza również czas wymagany do uruchomienia schematu, a także długość klucza, więc uczciwe strony bę dą chciały ustawić parametr bezpieczeństwa na jak najmniejszym poziomie, z zastrzeżeniem obrony przed klasą ataków, których dotyczą około.) Postrzegając parametr bezpieczeństwa jako długość klucza, odpowiada to mniej wię cej faktowi, że czas wymagany do ataku polegającego na wyczerpującym przeszukiwaniu rośnie wykładniczo wraz ze wzrostem długości klucza. Możliwość „zwię kszenie bezpieczeństwa” poprzez zwię kszenie parametru bezpieczeństwa ma istotne konsekwencje praktyczne, gdyż umożliwia uczciwym stronom obronę przed wzrostem mocy obliczeniowej. Poniższy przykład pokazuje, jak mogłoby to wyglądać w praktyce.

Przykład 3.3

Zobaczmy, jaki wpływ na bezpieczeństwo może mieć dostęp pnoś szybszych komputerów w praktyce. Założmy, że mamy schemat kryptograficzny, w którym uczciwe strony działają przez $106 \cdot n$ cykli i mogą „złamać” schemat z 2^2 cykli, i dla którego przeciwnik biegnie przez $108 \cdot n$ prawdopodobieństwem co najwyżej $2^{-n/2}$. (Liczby mają na celu ułatwienie obliczeń i nie mają odpowiać żadnemu istniejącemu schematowi kryptograficznemu.)

Założmy, że wszystkie strony korzystają z komputerów o czę stotliwości 2 GHz, a uczciwe strony ustalają $n = 80$. Nastę pnie uczciwe strony biegą przez $106 \cdot 6400$ cykli, czyli 3,2 sekundy, a przeciwnik biegący przez $108 \cdot (80)4$ cykle, czyli około 3 tygodnie, może złamać schemat z prawdopodobieństwem zaledwie 2-40 .

Założmy, że komputery 8 GHz stają się dostęp pne i wszystkie strony dokonują aktualizacji. Uczciwi ludzie mogą zwię kszyć n do 160 (co wymaga wygenerowania nowego klucza) i utrzymać czas działania 3,2 sekundy (tj. $106 \cdot 1602$ cykli przy $8 \cdot 109$ cyklach/sekundę). Dla kontrastu, przeciwnik musi teraz biec przez ponad 8 milionów sekund, czyli ponad 13 tygodni, aby osiągnąć prawdopodobieństwo sukcesu wynoszące 2-80. Efektem szybszego komputera było utrudnienie pracy przeciwnikowi.

Nawet stosując podejście asymptotyczne, należy pamiętać, że ostatecznie, gdy kryptosystem zostanie wdrożony w praktyce, potrzebna bę dzie konkretna gwarancja bezpieczeństwa. (W końcu trzeba zdecydować się na jakąś wartość n.) Jednakże, jak wskazują powyższe przykłady, ogólnie rzecz biorąc, asymptotyczne oświadczenie dotyczące bezpieczeństwa można przełożyć na konkretne ograniczenie bezpieczeństwa dla dowolnej pożądanej wartości n.

Podejście asymptotyczne w szczegółach

Omówimy teraz bardziej formalnie pojęcia „algorytmy czasu wielomianowego” oraz „znikome prawdopodobieństwo sukcesu”.

Wydajne algorytmy. Zdefiniowaliśmy algorytm jako efektywny, jeśli działa w czasie wielomianowym. Algorytm A działa w czasie wielomianowym, jeśli istnieje a

wielomian p taki, że dla każdego wejścia $x \in \{0, 1\}^n$ obliczenia $A(x)$ kończą się w co najwyżej $p(|x|)$ krokach. (Tutaj $|x|$ oznacza długość łańcucha x .) Jak wspomnieliśmy wcześniej, interesują nas tylko przeciwnicy, których czas działania jest wielomianem w parametrze bezpieczeństwa n . Ponieważ czas działania algorytmu mierzymy długością danych wejściowych, czasami dostarczamy algorytmy z parametrem bezpieczeństwa zapisanym w postaci jednoargumentowej (tj. jako $1n$ lub ciąg n jedynek) jako dane wejściowe. Strony (a dokładniej uruchamiane przez nie algorytmy) mogą poza parametrem bezpieczeństwa przyjmować inne dane wejściowe —na przykład wiadomość, która ma zostać zaszyfrowana —a my pozwalamy, aby czas ich działania był wielomianem w stosunku do (całkowitej) długości ich danych wejściowych.

Domyślnie pozwalamy, aby wszystkie algorytmy były probabilistyczne (lub losowe). Każdy taki algorytm może „rzucić monetą” na każdym etapie jego wykonania; jest to metaforyczny sposób powiedzenia, że algorytm może uzyskać dostęp p do nieobciążonego losowego bitu na każdym kroku. Równoważnie możemy postrzegać algorytm losowy jako taki, który oprócz danych wejściowych otrzymuje równomiernie rozłożoną losową taśmę o wystarczającej długości¹, której bity może wykorzystać w razie potrzeby podczas wykonywania.

Domyślnie rozważamy algorytmy losowe z dwóch powodów. Po pierwsze, losowość jest niezbędna w kryptografii (np. w celu wyboru losowych kluczy itd.), dlatego uczciwe strony muszą kierować się probabilizmem; biorąc to pod uwagę, naturalnym jest, aby przeciwnicy również zachowywali się probabilistycznie. Po drugie, randomizacja jest praktyczna i — o ile nam wiadomo — daje atakującym dodatkową moc. Ponieważ naszym celem jest modelowanie wszystkich realistycznych ataków, wolimy bardziej liberalną definicję wydajnych obliczeń.

Znikome prawdopodobieństwo sukcesu. Funkcja pomijalna to taka, która jest asymptotycznie mniejsza niż jakakolwiek odwrotna funkcja wielomianowa. Formalnie:

DEFINICJA 3.4 Funkcja f łącząca liczby naturalne z nieujemnymi liczbami rzeczywistymi jest pomijalna, jeśli dla każdego dodatniego wielomianu p istnieje N taki, że dla wszystkich liczb całkowitych $n > N$ zachodzi zasada, że $f(n) < p(n)$.

W skrócie powyższe można również zapisać następująco: dla każdego wielomianu p i wszystkich dostatecznie dużych wartości n obowiązuje, że $\lim_{n \rightarrow \infty} f(n) / p(n) = 0$. Równoważne sformułowanie powyższego wymaga, aby dla wszystkich stałych c istniało N takie, że dla wszystkich $n > N$ utrzymuje się, że $f(n) < n^{-c}$. Zwykle oznaczamy dowolną, pomijalną funkcję przez negl.

Przykład 3.5

Funkcje $f(n) = \frac{1}{n}$ i $f(n) = \sqrt{n}$ są pomijalne. Jej wartości zbliżają się do zero w bardzo różnym tempie. Na przykład możemy sprawdzić minimalną wartość n , dla której każda funkcja jest mniejsza niż $1/n^5$:

¹ Jeżeli dany algorytm działa dla kroków $p(n)$ na wejściach o długości n , wówczas wystarczy losowa taśma o długości $p(n)$, ponieważ atakujący może odczytać co najwyżej jeden losowy bit na krok czasu.

1. Rozwiążując $2^{n-5} < n^5$ otrzymujemy $n > 5 \log n$. Najmniejsza wartość całkowita n , dla której to zachodzi, to $n = 23$.

2. Rozwiązywanie $2^{\frac{n}{5}} < n^5$ otrzymujemy $n > 25 \log_2 n$. Najmniejsza wartość całkowita n , dla którego to zachodzi, wynosi $n = 3500$.

3. Rozwiązanie $\log n < n^5$ otrzymujemy $\log n > 5$. Najmniejsza wartość całkowita n , dla którego zachodzi to, wynosi $n = 33$.

Z powyższego można odnieść wrażenie, że n szybciej niż $\log n$ zbliża się do zera. Jest to jednak błąd dne; dla wszystkich $n > 65536$ utrzymuje się, że $2^{\frac{n}{\log n}} > n$. Niemniej jednak pokazuje to, że dla wartości n w $\log n$ setek lub tysięcy cy, prawdopodobieństwo sukcesu przeciwnika równe n jest lepsze niż prawdopodobieństwo sukcesu przeciwnika wynoszące $2^{\frac{n}{\log n}}$.

jest

Techniczną zaletą pracy z znikomym prawdopodobieństwem sukcesu jest to, że spełniają one pewne właściwości domknięcia. Poniżej znajduje się łatwe ćwiczenie.

TWIERDZENIE 3.6 Niech negl_1 i negl_2 będą funkcjami pomijalnymi. Następuje,

1. Funkcja negl_3 zdefiniowana przez $\text{negl}_3(n) = \text{negl}_1(n) + \text{negl}_2(n)$ jest pomijalna.

2. Dla dowolnego dodatniego wielomianu p funkcja negl_4 określona przez $\text{negl}_4(n) = p(n) \cdot \text{zaniebanie}_1(n)$ jest nieistotne.

Z drugiej części powyższego twierdzenia wynika, że jeśli w pewnym eksperymencie jakieś zdarzenie wystąpi z prawdopodobieństwem znikomym, to zdarzenie to wystąpi z prawdopodobieństwem znikomym, nawet jeśli doświadczenie będzie powtarzane wielomianowo wiele razy. (To opiera się na połączeniu unijnym; patrz Twierdzenie A.7.)

Na przykład prawdopodobieństwo, że po n uczciwych rzutach monetą wypadnie reszka, jest znikome.

Oznacza to, że nawet jeśli wielokrotnie powtórzymy eksperiment polegający na rzucie n monetami wielomianowo, prawdopodobieństwo, że w którymkolwiek z tych eksperymentów wyrzucimy reszkę, jest nadal znikome.

Konsekwencją drugiej części powyższego twierdzenia jest to, że jeśli jest to funkcja $f(n)$ definiująca się dla n nie jest pomijalne, to funkcja $f(n) = g(n)/p(n)$ dla dowolnego

Bezpieczeństwo asymptotyczne: podsumowanie

Każda definicja bezpieczeństwa składa się z dwóch części: definicji tego, co jest uważane za „przerwanie” schematu, oraz określenia siły przeciwnika.

Siła przeciwnika może wiązać się z wieloma kwestiami (np. w przypadku szyfrowania, czy zakładamy atak wykorzystujący wyłącznie tekst zaszyfrowany, czy atak z wybranym tekstem jawnym). Jeśli jednak chodzi o moc obliczeniową przeciwnika, odtąd będzie dzielić modelować przeciwnika jako wydajnego i dlatego będzie dzielić brać pod uwagę tylko strategie kontradiktoryjne, które można wdrożyć w wielomianie probabilistycznym

czas. Definicje będą dążeć zawsze tak formułowane, że przerwa występująca z znikomym prawdopodobieństwem nie będzie uznawana za znaczącą. Zatem ogólne ramy dowolnej definicji bezpieczeństwa będą wyglądać następująco:

Schemat jest bezpieczny, jeśli dla każdego wielomianowego przeciwnika A przeprowadzającego atak pewnego formalnie określonego typu, prawdopodobieństwo, że A odniesie sukces w ataku (jeśli sukces jest również formalnie określony) jest znikome.

Taka definicja jest asymptotyczna, ponieważ jest możliwe, że przy małych wartościach n przeciwnik może odnieść sukces z dużym prawdopodobieństwem. Aby zobaczyć to bardziej szczegółowo, w powyższym stwierdzeniu rozszerzamy termin „nieistotny”:

Schemat jest bezpieczny, jeśli dla każdego ppt przeciwnika A przeprowadzającego atak jakiegoś formalnie określonego typu i dla każdego dodatniego wielomianu p istnieje liczba całkowita N taka, że gdy $n > N$ prawdopodobieństwo, że A powiedzie się w ataku wynosi 1 mniej niż $p(n)$. —

Należy pamiętać, że dla wartości $n < N$ nie ma żadnej gwarancji.

O wyborach dokonanych przy definiowaniu bezpieczeństwa asymptotycznego

Definiując ogólne pojęcie bezpieczeństwa asymptotycznego, dokonaliśmy dwóch wyborów: zidentyfikowaliśmy skuteczne strategie kontradyktoryjne z klasą probabilistycznych algorytmów czasu wielomianowego i zrównaliśmy małe szanse powodzenia z znikomym prawdopodobieństwem. Obydwa te wybory są w pewnym stopniu arbitralne i można zbudować całkowicie rozsądną teorię, definiując, powiedzmy, skuteczne strategie, takie jak te działające w czasie kwadratowym, lub małe prawdopodobieństwa sukcesu, takie jak te ograniczone przez 2^{-n} . Niemniej jednak krótko uzasadniamy dokonane przez nas wybory (które są standardowymi).

Osoby zaznajomione z teorią lub algorytmami złożoności rozpoznają, że koncepcja utożsamiania wydajnych obliczeń z (probabilistycznymi) algorytmami czasu wielomianowego nie jest wyłącznie związana z kryptografią. Jedną z zalet stosowania (probabilistycznego) czasu wielomianowego jako miary efektywności jest to, że uwalnia nas to od konieczności precyzyjnego określania naszego modelu obliczeń, ponieważ rozszerzona teza Churcha-Turinga stwierdza, że wszystkie „rozsądne” modele obliczeń są równoważne wielomianowo. Zatem nie musimy określać, czy używamy maszyn Turinga, obwodów logicznych czy maszyn o dostępie swobodnym; możemy przedstawić algorytmy w pseudokodzie wysokiego poziomu i mieć pewność, że jeśli nasza analiza wykaże, że algorytmy te działają w czasie wielomianowym, to każda rozsądna implementacja również to zrobi.

Inną zaletą (probabilistycznych) algorytmów czasu wielomianowego jest to, że spełniają one pożądane właściwości domknięcia: w szczególności algorytm, który wykonuje wielomianowo wiele wywołań podprogramu czasu wielomianowego (i dodatkowo wykonuje tylko obliczenia wielomianowe), sam będzie działał w czasie wielomianowym.

Najważniejszą cechą prawdopodobieństw pomijałnych jest własność domknięcia, którą widzieliśmy już w Twierdzeniu 3.6(2): każdy wielomian razy pomijalna funkcja jest nadal pomijalna. Oznacza to w szczególności, że jeśli algorytm

wykonuje wielomianowo wiele wywołań pewnego podprogramu, który „nie powiedzie się” z pomijalną wartością prawdopodobieństwo przy każdym wywołaniu, a następnie nie powiedzie się, że którekolwiek z wywołań awaria podprogramu jest nadal nieistotna.

Konieczność relaksu

Tajemnica obliczeniowa wprowadza dwa złagodzenia tajemnicy doskonałej: po pierwsze, bezpieczeństwo jest gwarantowane tylko wobec skutecznych przeciwników; po drugie, dopuszczalne jest małe prawdopodobieństwo sukcesu. Obydwa te relaksacje są niezbędne do osiągnięcia praktycznych schematów szyfrowania, a w szczególności ominięcia negatywnych wyników w celu uzyskania całkowicie tajnego szyfrowania. Nieformalnie dyskutujemy, dlaczego tak jest sprawą. Założymy, że mamy schemat szyfrowania, w którym wielkość przestrzeni klucza K jest znacznie mniejsza niż rozmiar przestrzeni wiadomości M . (Jak pokazano w poprzednim rozdziale, oznacza to, że schemat nie może być całkowicie tajny.) Dwa ataki mają zastosowanie niezależnie od konstrukcji schematu szyfrowania:

- Mając zaszyfrowany tekst c , przeciwnik może odszyfrować c przy użyciu wszystkich kluczy $k \in K$. Daje to listę wszystkich wiadomości, którym c może ewentualnie odpowiadać. Ponieważ ta lista nie może zawierać wszystkich M (ponieważ $|K| < |M|$), ten atak ujawnia pewne informacje o zaszyfrowanej wiadomości.

Co więc cej, założymy, że przeciwnik przeprowadza atak ze znanym tekstem jawnym i dowiaduje się, że szyfrogramy c_1, \dots, c_n odpowiadają komunikatom m_1, \dots, m_n , odpowiednio. Przeciwnik może ponownie spróbować odszyfrować każdy z tych szyfrogramów wszystkimi możliwymi kluczami, aż znajdzie klucz k , dla którego $\text{Deck}(c_i) = m_i$ dla wszystkich. Później, mając zaszyfrowany tekst c będący szyfrowaniem nieznanej wiadomości m , prawie na pewno jest tak, że $\text{Deck}(c) = m$.

Ataki wyczerpujące, takie jak powyższe, pozwalają przeciwnikowi odnieść sukces z prawdopodobieństwem zasadniczo 1 w czasie liniowym w $|K|$.

- Rozważmy ponownie przypadek, w którym przeciwnik dowiaduje się, że szyfrogramy c_1, \dots, c_n odpowiadają komunikatom m_1, \dots, m_n . Przeciwnik może odgadnąć: a uniform key $k \in K$ i sprawdź, czy $\text{Deck}(c_i) = m_i$ dla wszystkich i . W takim razie, następnie, jak powyżej, atakujący może użyć k do późniejszego odszyfrowania czegokolwiek szyfrowane przez uczciwe strony.

Tutaj przeciwnik działa w zasadniczo stałym czasie i odnosi sukces niezerowy (choć bardzo małe) prawdopodobieństwo $1/|K|$.

Wynika z tego, że jeśli chcemy zaszyfrować wiele wiadomości za pomocą jednego krótkiego klucza, bezpieczeństwo można osiągnąć tylko wtedy, gdy ograniczymy czas działania przeciwnika (tj. przeciwnik nie ma wystarczająco dużo czasu, aby przeprowadzić przeszukanie metodą brute-force) i są gotowi przyznać bardzo małe prawdopodobieństwo sukcesu (a więc drugie „atak” jest wykluczony).

3.2 Definiowanie bezpiecznego szyfrowania obliczeniowego

Biorąc pod uwagę tło poprzedniej sekcji, jesteśmy gotowi przedstawić definicję bezpieczeństwa obliczeniowego dla szyfrowania klucza prywatnego. Po pierwsze, ponownie definiujemy składnię szyfrowania kluczem prywatnym; bę dzie to zasadniczo takie samo jak składnia wprowadzona w rozdziale 2, z tą różnicą, że teraz wyraźnie uwzglę dnimy parametr bezpieczeństwa n . Umożliwiamy również algorytmowi deszyfrowania wyświetlenie komunikatu o błędzie w przypadku, gdy zostanie wyświetlony nieprawidłowy tekst zaszyfrowany.

Na koniec domyślnie pozwalamy, aby przestrzeń komunikatów była zbiorem $\{0, 1\}^n$ wszystkich ciągów binarnych (o skończonej długości).

DEFINICJA 3.7 Schemat szyfrowania klucza prywatnego to krotka probabilistycznych algorytmów czasu wielomianowego (Gen , Enc , Dec), takich że:

Algorytm generowania klucza Gen przyjmuje jako dane wejściowe n (tj. zabezpieczenie 1).

1 parametr zapisany w postaci jednoargumentowej) i wyprowadza klucz k ; piszemy $k = \text{Gen}(1^n)$ (podkreślając, że Gen jest algorymem losowym). Zakładamy bez utraty ogólności, że dowolny wynik klucza $\text{Gen}(1^n)$ spełnia $|k| = n$.

2. Algorytm szyfrowania Enc przyjmuje jako dane wejściowe klucz k i wiadomość w postaci zwykłego tekstu $m \in \{0, 1\}^n$ i wyprowadza tekst zaszyfrowany c . Ponieważ Enc może być losowy, zapisujemy to jako $c = \text{Enc}_k(m)$.

3. Algorytm deszyfrujący Dec przyjmuje jako dane wejściowe klucz k i tekst zaszyfrowany c i generuje komunikat m lub błąd. Zakładamy, że Dec jest deterministyczny i dlatego zapisujemy $m := \text{Deck}(c)$ (zakładając, że Dec nie zwraca błędu). Błąd ogólny oznaczamy symbolem \perp .

Wymagane jest, aby dla każdego n , każdego klucza k wyprowadzonego przez $\text{Gen}(1^n)$ i każdego $m \in \{0, 1\}^n$ utrzymywało się, że $\text{Deck}(\text{Enc}_k(m)) = m$.

Jeżeli $(\text{Gen}, \text{Enc}, \text{Dec})$ jest takie, że dla k wyjście przez $\text{Gen}(1^n)$, algorytm Enc_k jest zdefiniowany tylko dla komunikatów $m \in \{0, 1\}^n$ to mówimy, że $(\text{Gen}, \text{Enc}, \text{Dec})$ jest schematem szyfrowania kluczem prywatnym o stałej długości dla wiadomości o długości n .

Prawie zawsze $\text{Gen}(1^n)$ po prostu wyprowadza jednolity n -bitowy串 znaków jako klucz. W takim przypadku pominiemy Gen i po prostu zdefiniujemy schemat szyfrowania klucza prywatnego za pomocą pary algorytmów (Enc, Dec) .

Powysza definicja uwzglę dnia schematy bezstanowe, w których każde wywołanie Enc (i Dec) jest niezależne od wszystkich wcześniejszych wywołań. W dalszej części rozdziału od czasu do czasu bę dziemy omawiać schematy stanowe, w których nadawca (i ewentualnie odbiorca) jest zobowiązany do utrzymywania stanu podczas wywołań. O ile wyraźnie nie zaznaczono inaczej, wszystkie nasze wyniki zakładają bezstanowe szyfrowanie/deszyfrowanie.

3.2.1 Podstawowa definicja bezpieczeństwa

Zaczniemy od przedstawienia najbardziej podstawowego pojęcia bezpieczeństwa klucza prywatnego szyfrowanie: zabezpieczenie przed atakiem wykorzystującym wyłącznie szyfrrogram, w którym przeciwnik przestrzega tylko jednego szyfrrogramu lub, równoważnie, zabezpieczenia przy danym kluczu służy do szyfrowania tylko pojedynczej wiadomości. Rozważamy mocniejsze definicje bezpieczeństwa w dalszej części rozdziału.

Motywowanie – definicja. Jak już omówiliśmy, dowolna definicja bezpieczeństwa składa się z dwóch odrębnych elementów: modelu zagrożenia (tj. określenia zakładanej siły przeciwnika) oraz celu bezpieczeństwa (zwyczajnie określone poprzez opisanie, co stanowi „przerwę” w schemacie). Zaczynamy nasze podejście definicyjne poprzez rozważenie najprostszego modelu zagrożenia, gdzie mamy podsłuchującego przeciwnika, który obserwuje szyfrowanie pojedynczego wiadomości. To jest dokładnie ten model zagrożenia, który rozważaliśmy w poprzednim artykule rozdziału, z tym wyjątkiem, że – jak wyjaśniono w poprzedniej sekcji – tak jest teraz interesują się tylko przeciwnikami ograniczonymi obliczeniowo i tak dalej ograniczone do działania w czasie wielomianowym.

Chociaż przyjęliśmy dwa założenia dotyczące możliwości przeciwnika (mianowicie, że tylko podsłuchuje i że działa w czasie wielomianowym), to próbując tego dokonać, nie należy przyjmować żadnych założeń na temat strategii przeciwnika rozszyfrować zaszyfrowany tekst, który zauważa. Jest to kluczowe dla uzyskania znaczenia pojęcia bezpieczeństwa; definicja zapewnia ochronę przed każdym ograniczonym obliczeniowo przeciwnikiem, niezależnie od używanego algorytmu.

Prawidłowe zdefiniowanie celu bezpieczeństwa dla szyfrowania nie jest trywialne, ale my omówiliśmy już tę kwestię obszernie w sekcji 1.4.1 oraz w poprzednim rozdziale. Dlatego przypominamy tylko, że idea definicji jest o których przeciwnik nie powinien być w stanie uzyskać żadnych częściowych informacji tekstu zaszyfrowanego. Definicja bezpieczeństwa semantycznego (por. Sekcja 3.2.2) dokładnie formalizuje to pojęcie i była pierwszą definicją zaproponowanie bezpiecznego szyfrowania obliczeniowego. Bezpieczeństwo semantyczne jest złożone i trudne w obsłudze. Na szczęście istnieje równoważna definicja nazywająca się nierozróżnialnością, która jest znacznie prostsza.

Definicja nierozróżnialności wzorowana jest na alternatywnej definicji doskonalej tajemnicy podanej jako Definicja 2.5. (To stanowi dodatkową motywację dla tego, że definicja nierozróżnialności jest dobra.) Przypomnijmy sobie to

Definicja 2.5 dotyczy eksperymentu PrivKeav dwie w którym przeciwnik A out- A, Π wysyła wiadomości m0 i m1, a następnie otrzymuje szyfrowanie jednej z nich wiadomości przy użyciu jednolitego klucza. Z definicji wynika, że schemat Π jest bezpieczny jeśli żaden przeciwnik A nie może określić, która z wiadomości m0, m1 została zaszyfrowana z prawdopodobieństwem różnym od 1/2, czyli prawdopodobieństwem, że A jest poprawne, jeśli po prostu zgaduje.

Tutaj zachowujemy eksperyment PrivKeav A, Π prawie dokładnie tak samo (z wyjątkiem pewne różnice techniczne omówione poniżej), ale wprowadzają dwie kluczowe modyfikacje w samej definicji:

1. Rozważamy teraz tylko przeciwników działających w czasie wielomianowym, podczas gdy Definicja 2.5 uwzględnia nawet przeciwników o nieograniczonym czasie działania.

2. Przyznajemy teraz, że przeciwnik może określić zaszyfrowaną wiadomość z prawdopodobieństwem znakomitości niż 1/2.

Jak szczegółowo omówiono w poprzedniej sekcji, powyższe relaksacje stanowią podstawowe elementy bezpieczeństwa obliczeniowego.

Jeśli chodzi o inne różnice, najbardziej widoczna jest taka, że teraz parametryzujemy eksperyment za pomocą parametru bezpieczeństwa n . Następnie mierzymy zarówno czas działania przeciwnika A, jak i prawdopodobieństwo jego sukcesu jako funkcje n . Zapisujemy $\text{PrivKeav } A, \Pi(n)$, aby oznaczyć eksperyment przeprowadzany z parametrem bezpieczeństwa n i piszemy

$$\Pr[\text{PrivKeav } A, \Pi(n) = 1] \quad (3.1)$$

aby oznaczyć prawdopodobieństwo, że wynik eksperymentu $\text{PrivKeav } A, \Pi(n)$ wynosi 1.

Zauważ, że przy stałym A, Π równanie (3.1) jest funkcją n .

Druga różnica w doświadczeniu $\text{PrivKeav } A, \Pi$ polega na tym, że teraz wyraźnie wymagamy, aby przeciwnik wysłał dwa komunikaty m_0, m_1 o równej długości. (W definicji 2.5 to wymaganie jest ukryte, jeśli przestrzeń wiadomości M zawiera tylko wiadomości o określonej długości, jak ma to miejsce w przypadku schematu szyfrowania jednorazowego.) Oznacza to, że domyślnie nie wymagamy bezpiecznego schematu szyfrowania ukrywającego długość tekstu jawnego. Powrócimy do tego punktu na końcu tej sekcji, patrz także ćwiczenia 3.2 i 3.3.

Nierozróżnialność w obecności podsłuchującego. Podamy teraz formalną definicję, zaczynając od eksperymentu opisanego powyżej. Eksperyment jest zdefiniowany dla dowolnego schematu szyfrowania klucza prywatnego $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, dowolnego przeciwnika A i dowolnej wartości n parametru bezpieczeństwa:

Eksperyment nieodróżnialności kontradyktoryjnej $\text{PrivKeav } A, \Pi(n)$:

1. Przeciwnik A otrzymuje dane wejściowe $1 n$ i wyprowadza parę wiadomości m_0, m_1 z $|m_0| = |m_1|$.
2. Klucz k jest generowany poprzez uruchomienie $\text{Gen}(1n)$ i wybierany jest jednolity bit $b \in \{0, 1\}$. Zaszyfrowany tekst $c = \text{Enc}_k(mb)$ jest obliczany i przekazywany A . c nazywamy zaszyfrowanym tekstem wyzwania.
3. A wyprowadza trochę b .
4. Wynik eksperymentu definiuje się jako 1, jeśli $b = b$, i 0 w przeciwnym razie. Jeśli $\text{PrivKeav } A, \Pi(n) = 1$, mówimy, że A się udało.

Nie ma ograniczeń co do długości m_0 i m_1 , o ile są one takie same. (Oczywiście, jeśli A przebiega w czasie wielomianowym, wówczas m_0 i m_1 mają wielomianową długość w n .) Jeśli Π jest schematem o stałej długości dla komunikatów o długości (n) , powyższy eksperyment jest modyfikowany poprzez wymaganie $m_0, m_1 \in \{0, 1\}^n$

Fakt, że przeciwnik może jedynie podsłuchiwać, wynika z faktu, że wprowadzane przez niego dane ograniczają się do (pojedynczego) tekstu zaszyfrowanego, a przeciwnik nie wchodzi w żadną dalszą interakcję z nadawcą ani odbiorcą. (Jak zobaczymy później, umożliwienie dodatkowej interakcji znacznie wzmacni przeciwnika.)

Definicja nieroróżnialności stwierdza, że schemat szyfrowania jest bezpieczny, jeśli żadnemu przeciwnikowi ppt A nie uda się odgadnąć, która wiadomość została zaszyfrowana w powyższym eksperymentie z prawdopodobieństwem znacznie większym niż zgadywanie losowe (co jest poprawne z prawdopodobieństwem 1/2):

DEFINICJA 3.8 Schemat szyfrowania kluczem prywatnym $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ ma nieroróżnialne szyfrowanie w obecności podsłuchującego lub jest bezpieczny EAV, jeśli dla wszystkich probabilistycznych przeciwników w czasie wielomianowym A istnieje pomijalna funkcja negl taki, że dla wszystkich n ,

$$\Pr[\text{PrivKeav } A, \Pi(n) = 1] \xrightarrow{\text{negl}(n)} \Pr[\text{zaniebywanie}(n), 2]$$

gdzie prawdopodobieństwo przejmuje losowość zastosowaną przez A i losowość zastosowaną w eksperymencie (do wyboru klucza i bitu b, a także losowość zastosowaną przez Enc).

Uwaga: jeśli nie określono inaczej, pisząc „ $f(n)$ ” – „ $g(n)$ ” mamy na myśli że nierówność zachodzi dla wszystkich n .

Powinno być jasne, że Definicja 3.8 jest słabsza niż Definicja 2.5, co jest równoznaczne z doskonałą tajemnicą. Zatem każdy doskonale tajny schemat szyfrowania ma szyfrowanie nie do odróżnienia w obecności podsłuchującego. Naszym celem będzie zatem pokazanie, że istnieją schematy szyfrowania spełniające powyższe, w których klucz jest krótszy od wiadomości. Oznacza to, że pokażemy schematy, które spełniają Definicję 3.8, ale nie spełniają Definicji 2.5.

Równoważne sformułowanie. Definicja 3.8 wymaga, aby żaden przeciwnik ppt nie mógł określić, która z dwóch wiadomości została zaszyfrowana, z prawdopodobieństwem znacznie większym niż 1/2.

Równoważne sformułowanie jest takie, że każdy przeciwnik ppt zachowuje się tak samo, niezależnie od tego, czy widzi szyfrowanie m_0 , czy m_1 . Ponieważ A wyprowadza pojedynczy bit, „zachowując się tak samo” oznacza, że wyprowadza 1 z prawie takim samym prawdopodobieństwem w każdym przypadku. Aby to sformalizować, zdefiniuj $\text{PrivKeav } A, \Pi(n, b)$ jak powyżej, z tą różnicą, że używany jest stały bit b (a nie wybierany losowo).

Niech $\text{outA}(\text{PrivKeav } A, \Pi(n, b))$ oznacza bit wyjściowy b elementu A w eksperymencie. Poniżej zasadniczo stwierdza się, że żaden A nie może określić, czy działa w eksperymencie $\text{PrivKeav } A, \Pi(n, 0)$ czy w eksperymencie $\text{PrivKeav } A, \Pi(n, 1)$.

DEFINICJA 3.9 Schemat szyfrowania klucza prywatnego $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ ma nieroróżnialne szyfrowanie w obecności podsłuchującego, jeśli dla wszystkich przeciwników ppt A istnieje pomijalna funkcja negl taka, że

$$\Pr[\text{outA}(\text{PrivKeav } A, \Pi(n, 0)) = 1] = \Pr[\text{outA}(\text{PrivKeav } A, \Pi(n, 1)) = 1] = \text{negl}(n).$$

Fakt, że jest to równoważne definicji 3.8, pozostawiamy jako ćwiczenie.

Szyfrowanie i długość tekstu jawnego

Domyślna koncepcja bezpiecznego szyfrowania nie wymaga, aby schemat szyfrowania ukrywał długość tekstu jawnego i w rzeczywistości wszystkie powszechnie stosowane schematy szyfrowania ujawniają długość tekstu jawnego (lub jej przybliżoną długość). Głównym powodem jest to, że nie jest możliwa obsługa komunikatów o dowolnej długości przy jednoczesnym ukrywaniu wszystkich informacji o długości tekstu jawnego (por. Ćwiczenie 3.2). W wielu przypadkach nie ma to znaczenia, ponieważ długość tekstu jawnego jest już publiczna lub nie jest poufna. Nie zawsze tak jest jednak i czasami wyciekanie długości tekstu jawnego jest problematyczne. Jako przykłady:

- Proste dane numeryczne/tekstowe: Założmy, że używany schemat szyfrowania dokładnie określa długość tekstu jawnego. Następnie zaszyfrowana informacja o wynagrodzeniu ujawniłaby, czy ktoś zarabia pięć cyocyfrową, czy sześciocyfrową pensję. Podobnie szyfrowanie odpowiedzi „tak”/„nie” spowodowałoby wyciek dokładnej odpowiedzi.
- Automatyczne sugestie: Strony internetowe często zawierają funkcję „automatycznego uzupełniania” lub „automatycznej sugestii”, dzięki której serwer WWW sugeruje listę potencjalnych słów lub wyrażeń w oparciu o częstotliwe informacje, które użytkownik już wpisał. Rozmiar tej listy może ujawnić informacje o literach, które użytkownik wpisał do tej pory. (Na przykład liczba autouzupełnień zwróconych dla „th” jest znacznie większa niż liczba dla „zo”).
- Wyszukiwanie w bazie danych: Rozważmy sytuację, w której użytkownik przeszukuje bazę danych w poszukiwaniu wszystkich rekordów pasujących do wyszukiwanego hasła. Liczba zwróconych rekordów może ujawnić wiele informacji o tym, czego szukał użytkownik. Może to być szczególnie szkodliwe, jeśli użytkownik szuka informacji medycznych, a zapytanie ujawnia informacje o chorobie, na którą cierpi.
- Dane skompresowane: Jeśli tekst jawnego zostanie skompresowany przed zaszyfrowaniem, informacje o tekście jawnym mogą zostać ujawnione, nawet jeśli szyfrowane będą tylko dane o stałej długości. (Taki schemat szyfrowania nie spełnia zatem definicji 3.8.) Na przykład krótki skompresowany tekst jawnego wskazywałby, że oryginalny (nieskompresowany) zwykły tekst ma dużą redundancję. Jeśli przeciwnik może kontrolować częstotliwość zaszyfrowanej treści, luka ta może umożliwić mu zdobycie dodatkowych informacji na temat zwykłego tekstu; wykazano, że możliwe jest zastosowanie dokładnie tego rodzaju ataku (atak CRIME) na zaszyfrowany ruch HTTP w celu ujawnienia tajnych plików cookie sesji.

Korzystając z szyfrowania, należy określić, czy wyciek tekstu jawnego stanowi problem, a jeśli tak, podjąć kroki w celu ograniczenia takiego wycieku lub zapobiegania mu poprzez dopełnienie wszystkich wiadomości do określonej z góry długości przed ich zaszyfrowaniem.

3.2.2 *Bezpieczeństwo semantyczne

Motywując definicję bezpiecznego szyfrowania stwierdzeniem, że przeciwnik nie powinien mieć możliwości poznania jakichkolwiek częstotliwych informacji o tekście jawnym

z szyfrogramu. Definicja nieodróżnialności wygląda jednak zupełnie inaczej. Jak wspomnieliśmy, Definicja 3.8 jest równoważna definicji zwanej bezpieczeństwem semantycznym, która formalizuje pogląd, że częściowych informacji nie można się nauczyć. Dochodzimy do tej definicji, omawiając dwa słabsze pojęcia i pokazując, że implikuje się je poprzez nierozróżnialność.

Zaczniemy od pokazania, że nierozróżnialność oznacza, że zaszyfrowane teksty nie ujawniają żadnych informacji o poszczególnych fragmentach tekstu jawnego. Formalnie, powiedzmy, że schemat szyfrowania (Enc , Dec) jest zabezpieczony EAV (pamiętaj, że gdy pominię to Gen , klucz jest jednolitym ciągiem n -bitowym), a $m \in \{0, 1\}^n$ jest jednolite. Następnie pokazujemy, że dla dowolnego indeksu i nie można odgadnąć mi na podstawie $\text{Enck}(m)$ (gdzie w tej sekcji mi oznacza i-ty bit m) z prawdopodobieństwem znacznie większym niż $1/2$.

TWIERDZENIE 3.10 Niech $\Pi = (\text{Enc}, \text{Dec})$ będzie schematem szyfrowania kluczem prywatnym o stałej długości dla wiadomości o długości, których szyfrowanie jest nierozróżnialne w obecności podsłuchującego. Następnie dla wszystkich przeciwników ppt A i dowolnego $i \in \{1, \dots, n\}$, istnieje pomijalna funkcja negl taka, że

$$\Pr[\text{A}(\text{1}^n, \text{Enck}(m)) = m_i] = \frac{1}{2} + \text{negl}(n),$$

gdzie przyjmuje się prawdopodobieństwo jednorodne $m \in \{0, 1\}^n$ i $k \in \{0, 1\}^n$, losowość A i losowość Enc .

DOWÓD Ideą dowodu tego twierdzenia jest to, że gdyby można było odgadnąć i-ty bit m z $\text{Enck}(m)$, to byłoby również możliwe rozróżnienie pomiędzy szyfrowaniem wiadomości m_0 i m_1 , których i-te bity różnią się.

Formalizujemy to poprzez dowód redukcyjny, w którym pokazujemy, jak wykorzystać dowolnego efektywnego przeciwnika A do skonstruowania efektywnego przeciwnika A w taki sposób, że jeśli A narusza pojęcie bezpieczeństwa twierdzenia dla Π , wówczas A narusza definicję nierozróżnialności dla Π . (Patrz sekcja 3.3.2.) Ponieważ Π ma nierozróżnialne szyfrowanie, musi być również bezpieczne w sensie twierdzenia.

Napraw dowolnego przeciwnika ppt A i $i \in \{1, \dots, n\}$. Niech $I_0 \subseteq \{0, 1\}^n$ będą zbiorem wszystkich ciągów, których i-ty bit ma wartość 0 i niech $I_1 \subseteq \{0, 1\}^n$ będą zbiorem wszystkich ciągów, których i-ty bit wynosi 1. Mamy

$$\begin{aligned} \Pr[\text{A}(\text{1}^n, \text{Enck}(m)) = m_i] &= \Pr[m \in I_0 \mid \text{A}(\text{1}^n, \text{Enck}(m_0)) = 0] + \Pr[m \in I_1 \mid \text{A}(\text{1}^n, \text{Enck}(m_1)) = 1]. \end{aligned}$$

Skonstruuj następującego przeciwnika podsłuchującego A:

Przeciwnik A: 1.

Wybierz jednolite $m_0 \in I_0$ i $m_1 \in I_1$. Wyjście m_0, m_1 .

2. Po zaobserwowaniu zaszyfrowanego tekstu c wywołaj $\text{A}(\text{1}^n, c)$. Jeśli A wprowadza 0, wyjście $b = 0$; w przeciwnym razie wyjście $b = 1$.

A przebiega w czasie wielomianowym, ponieważ A to robi.

Z definicji eksperymentu $\text{PrivKeav}_{A,\Pi(n)}$ wiemy A powiedzie się wtedy i tylko wtedy, gdy A wyprowadzi b po otrzymaniu $\text{Enck}(mb)$. Więc

$$\begin{aligned} \Pr[\text{PrivKeav}_{A,\Pi(n)} = 1] &= \Pr[A(1^n, \text{Enck}(mb)) = b] \\ &= \frac{1}{m_0} \cdot \Pr_{I_0}[A(1^n, \text{Enck}(m_0)) = 0] + \frac{1}{m_1} \cdot \Pr_{I_1}[A(1^n, \text{Enck}(m_1)) = 1] \\ &= \Pr[A(1^n, \text{Enck}(m)) = m]. \end{aligned}$$

Zakładając, że (Enc, Dec) ma nieroróżnialne szyfrowanie w obecności podsłuchującego, istnieje pomijalna funkcja negl taka, że

$\Pr[\text{PrivKeav}_{A,\Pi(n)} = 1] = \frac{1}{2} + \text{zaniebywanie}(n)$. Dochodzimy do wniosku, że

$$\Pr[A(1^n, \text{Enck}(m)) = m] = \frac{1}{2} + \text{zaniebywanie}(n).$$

uzupełnienie dowodu. ■

Następnie twierdzimy z grubsza, że nieroróżnialność oznacza, że żaden przeciwnik ppt nie może nauczyć się żadnej funkcji tekstu jawnego, biorąc pod uwagę tekst zaszyfrowany, niezależnie od dystrybucji wysyłanej wiadomości. Ma to na celu uchwycenie idei, że wynikowy tekst zaszyfrowany nie wycieka żadnej informacji o tekście jawnym. Wymóg ten nie jest jednak łatwy do formalnego zdefiniowania. Aby zrozumieć dlaczego, zauważ, że nawet w przypadku rozważanym powyżej łatwo jest obliczyć i-ty bit m, jeśli m zostanie wybrane, powiedzmy, równomiernie ze zbioru wszystkich ciągów, których i-ty bit wynosi 0 (a nie równomiernie z $\{0, 1\}$). Zatem tak naprawdę chcemy powiedzieć, że jeśli istnieje przeciwnik, który poprawnie oblicza $f(m)$ z pewnym prawdopodobieństwem, mając dane $\text{Enck}(m)$, to istnieje przeciwnik, który może poprawnie obliczyć $f(m)$ z takim samym prawdopodobieństwem bez w ogóle otrzymywać szyfrogram (i znać tylko rozkład m). W dalszej części skupimy się na przypadku, gdy m jest wybrane równomiernie z jakiegoś zbioru $S \subseteq \{0, 1\}$.

TWIERDZENIE 3.11 Niech (Enc, Dec) będzie schematem szyfrowania kluczem prywatnym o stałej długości dla wiadomości o długości, których szyfrowanie jest nieroróżnialne w obecności podsłuchującego. Wtedy dla dowolnego algorytmu ppt A istnieje algorytm ppt A taki, że dla dowolnego $S \subseteq \{0, 1\}$ i dowolnej funkcji $f : \{0, 1\} \rightarrow \{0, 1\}$ istnieje funkcja pomijalna taka, że :

$$\Pr[A(1^n, \text{Enck}(m)) = f(m)] - \Pr[A(1^n) = f(m)] = \text{negl}(n),$$

gdzie pierwsze prawdopodobieństwo przyjmuje się za jednolity wybór $k \in \{0, 1\}^n$ i $m \in S$, losowość A i losowość Enc, a drugie prawdopodobieństwo za równomierny wybór m S i losowość z A.

DOWÓD (szkic) Fakt, że (Enc, Dec) jest bezpieczny pod względem EAV, oznacza, że dla dowolnego $S \subseteq \{0, 1\}^n$ żaden przeciwnik ppt nie może rozróżnić pomiędzy Enck(m) (dla jednorodnej formy $m \in S$) i Enck(1). Rozważmy teraz prawdopodobieństwo, że A pomyślnie obliczy f(m), biorąc pod uwagę Enck(m). Twierdzimy, że A powinno pomyślnie obliczyć f(m) biorąc pod uwagę Enck(1) z prawie takim samym prawdopodobieństwem; w przeciwnym razie A można zastosować do rozróżnienia Enck(m) i Enck(1). Rozróżniacz można łatwo skonstruować: wybierz jednorodny $m \in S$ i wyprowadź $m_0 = m$, $m_1 = 1$.

Po otrzymaniu tekstu zaszyfrowanego c, który jest zaszyfrowanym m_0 lub m_1 , wywołaj A(1^n , c) i wyprowadź 0 wtedy i tylko wtedy, gdy A wyprowadzi f(m). Jeżeli A wyprowadza f(m) przy zaszyfrowaniu m z prawdopodobieństwem znacznie różniącym się od prawdopodobieństwa, że wyprowadza f(m) przy zaszyfrowaniu 1, to opisany wyróżnik narusza definicję 3.9.

Powyższe sugeruje następujący algorytm A, który nie otrzymuje $c = \text{Enck}(m)$, ale oblicza $f(m)$ prawie tak dobrze jak A: A(1^n) wybiera jednorodny klucz $k \in \{0, 1\}^n$, wywołuje A na $\text{Enck}(1)$ i wyprowadza wszystko, co robi A. Z powyższego wynika, że A wyprowadza f(m), gdy jest uruchamiany jako podprogram przez A z prawie takim samym prawdopodobieństwem, jak wtedy, gdy otrzymuje Enck(m). Zatem A spełnia właściwość wymaganą w zastrzeżeniu. ■

Bezpieczeństwo semantyczne. Pełna definicja bezpieczeństwa semantycznego gwarantuje znacznie więcej niż własność rozważana w Twierdzeniu 3.11. Definicja pozwala na uzależnienie długości tekstu jawnego od parametru bezpieczeństwa i pozwala na zasadniczo dowolne rozkłady w tekstach jawnych. (Właściwie dopuszczałyśmy tylko rozkłady, które można skutecznie próbować. Oznacza to, że istnieje pewien probabilistyczny algorytm wielomianowy Samp taki, że Samp(1^n) generuje komunikaty zgodnie z rozkładem.) Definicja uwzględnia również dowolną „zewnętrzną” informację $h(m)$ o tekście jawnym, który może przedostać się do przeciwnika w inny sposób (np. dlatego, że ta sama wiadomość m jest wykorzystywana również w innym celu).

DEFINICJA 3.12 Schemat szyfrowania kluczem prywatnym (Enc, Dec) jest semantycznie bezpieczny w obecności podsłuchiwacza, jeśli dla każdego algorytmu ppt A istnieje algorytm ppt A taki, że dla dowolnego algorytmu ppt obliczalne funkcje f oraz h, co następuje jest nieistotne:

$$\Pr[A(1^n, \text{Enck}(m), h(m)) = f(m)] = \Pr[A(1^n, |m|, h(m)) = f(m)],$$

gdzie pierwsze prawdopodobieństwo jest przejmowane przez jednorodne $k \in \{0, 1\}^n$, m wyjście przez Samp(1^n), losowość A i losowość Enc, a drugie prawdopodobieństwo jest przejmowane przez m wyjście przez Samp(1^n) i losowość A.

Przeciwnik A otrzymuje szyfrogram Enck(m) oraz informację zewnętrzną h(m) i próbuje odgadnąć wartość f(m). Algorytm A również próbuje odgadnąć wartość f(m), ale podaje tylko h(m) i

długość m. Wymóg bezpieczeństwa stwierdza, że prawdopodobieństwo A jest prawidłowe zgadywanie $f(m)$ jest mniej więcej taki samo jak A. Intuicyjnie zatem szyfrogram $Enck(m)$ nie ujawnia żadnych dodatkowych informacji na temat wartości $f(m)$.

Definicja 3.12 stanowi bardzo mocne i przekonujące sformułowanie gwarancje bezpieczeństwa, które powinien zapewniać schemat szyfrowania. Jednakże łatwiej jest posługiwać się definicją nieroróżnialności (Definicja 3.8). Na szczęście definicje są równoważne:

TWIERDZENIE 3.13 Schemat szyfrowania klucza prywatnego jest nieroróżnialny jeśli szyfrowanie w obecności osoby podsłuchującej wtedy i tylko wtedy, gdy jest to semantyczne bezpieczne w obecności osoby podsłuchującej.

Patrząc w przyszłość, podobna równoważność między bezpieczeństwem semantycznym a nieroróżnialnością jest znana dla wszystkich definicji prezentowanych w tym rozdziale jak również te z rozdziału 11. Możemy zatem użyć nieroróżnialności jako zgodnie z naszą roboczą definicją, mając jednocześnie pewność, że osiągnięte gwarancje są takie te związane z bezpieczeństwem semantycznym.

3.3 Konstruowanie schematów bezpiecznego szyfrowania

Po zdefiniowaniu, co oznacza, że schemat szyfrowania jest bezpieczny, plik czytelnik może się spodziewać, że od razu przejdziemy do konstrukcji schematów bezpiecznego szyfrowania. Zanim jednak to zrobimy, musimy przedstawić pojęcie generatorów pseudolosowych (PRG) i szyfrów strumieniowych, ważny budynek bloki do szyfrowania klucza prywatnego. To kolejno doprowadzi do dyskusji pseudolosowości, która odgrywa w ogóle fundamentalną rolę w kryptografii oraz w szczególności szyfrowanie klucza prywatnego.

3.3.1 Generatory pseudolosowe i szyfry strumieniowe

Generator pseudolosowy G jest wydajnym, deterministycznym algorytmem przekształcanie krótkiego, jednolitego ciągu zwanego ziarnem w dłuższy „jednolicie wyglądający” (lub „pseudolosowy”) ciąg wyjściowy. Inaczej mówiąc, generator pseudolosowy wykorzystuje niewielką ilość prawdziwej losowości do generowania dużej ilości pseudolosowości. Jest to przydatne, gdy potrzebna jest duża liczba losowych (wyglądających) bitów, ponieważ generowane są prawdziwie losowe bity jest trudne i powolne. (Zobacz dyskusję na początku rozdziału 2.) W rzeczywistości generatory pseudolosowe badano co najmniej od lat czterdziestych XX wieku kiedy zaproponowano je do prowadzenia symulacji statystycznych. W tym kontekście badacze zaproponowali różne testy statystyczne, które posłużyły za generator pseudolosowy powinien przejść, aby można go było uznać za „dobry”. Jako prosty przykład pierwszy

bit wyjścia generatora pseudolosowego powinien być równy 1 z prawdopodobieństwem bardzo bliskim 1/2 (w przypadku gdy za prawdopodobieństwo przyjmuje się równomierny wybór materiału siewnego), ponieważ pierwszy bit jednolitego ciągu jest równy 1 z prawdopodobieństwo dokładnie 1/2. W rzeczywistości parzystość dowolnego ustalonego podzbioru bitów wyjściowych powinna również wynosić 1 z prawdopodobieństwem bardzo bliskim 1/2. Można również rozważyć bardziej złożone testy statystyczne.

To historyczne podejście do określania jakości jakiegoś potencjalnego generatora pseudolosowego ma charakter ad hoc i nie jest jasne, czy przejście jakiegoś zestawu testów statystycznych wystarczy, aby zagwarantować zasadność użycia potencjalnego generatora pseudolosowego w jakimś zastosowaniu. (W szczególności może istnieć inny test statystyczny, który skutecznie odróżnia dane wyjściowe generatora od prawdziwych bitów losowych). Podejście historyczne jest jeszcze bardziej problematyczne w przypadku używania generatorów pseudolosowych do zastosowań kryptograficznych; w tym ustawieniu bezpieczeństwo może zostać zagrożone, jeśli atakujący bę dzie w stanie odróżnić sygnał wyjściowy generatora od sygnału wyjściowego, a nie wiemy z góry, jaką strategię może zastosować atakujący.

Powysze rozważania motywowały kryptograficzne podejście do definiowania generatorów pseudolosowych w latach 80-tych. Podstawowym założeniem było to, że dobry generator pseudolosowy powinien przejść wszystkie (wydajne) testy statystyczne. Oznacza to, że dla dowolnego wydajnego testu statystycznego (lub wyróżnika) D prawdopodobieństwo, że D zwróci 1, gdy otrzymamy wynik generatora pseudolosowego, powinno być bliskie prawdopodobieństwu, że D zwróci 1, gdy otrzymamy jednolity ciąg o tej samej długości. Zatem nieformalnie wynik generatora pseudolosowego powinien „wyglądać” jak jednolity ciąg znaków dla każdego wydajnego obserwatora.

(Podkreślamy, że formalnie rzecz biorąc, nie ma sensu twierdzenie, że dowolny stały ciąg znaków jest „pseudolosowy” w taki sam sposób, w jaki nie ma sensu nazywanie dowolnego stałego ciągu ciągiem „losowym”. Pseudolosowość jest raczej właściwością Niemniej jednak czasami nieformalnie nazywamy ciąg próbkowany według rozkładu równomiernego „ciągiem jednolitym”, a ciąg wyprowadzany przez generator pseudolosowy „łańcuchem pseudolosowym”).

Inną perspektywę uzyskuje się poprzez zdefiniowanie, co oznacza, że rozkład jest pseudolosowy. Niech Dist bę dzie rozkładem na ciągach -bitowych. (Oznacza to, że Dist przypisuje pewne prawdopodobieństwo każdemu ciągowi w $\{0, 1\}$; próbkowanie z Dist oznacza, że wybieramy ciąg -bitowy zgodnie z tym rozkładem prawdopodobieństwa.) Nieformalnie Dist jest pseudolosowy, jeśli eksperyment, w którym próbkowany jest ciąg znaków z Dist jest nie do odróżnienia od eksperymentu, w którym próbkowany jest ciąg o jednakowej długości. (Ścisłe mówiąc, ponieważ jesteśmy w układzie asymptotycznym, musimy mówić o pseudolosowości sekwencji rozkładów $\text{Dist} = \{\text{Dist}_n\}$, gdzie rozkład Dist_n jest używany jako parametr bezpieczeństwa n . Pomijamy ten punkt w naszej obecnej dyskusji.) Więcej dokładnie, nie powinno być możliwe, aby jakikolwiek algorytm czasu wielomianowego stwierdził (lepiej niż zgadywanie), czy otrzymał ciąg próbkowany zgodnie z Dist , czy też otrzymał jednolity ciąg -bitowy. Oznacza to, że ciąg pseudolosowy jest tak samo dobry jak ciąg jednolity, pod warunkiem, że uwzględnimy tylko obserwatorów czasu wielomianowego. Podobnie jak nierozróżnialność jest relaksacją obliczeniową

doskonała tajemnica, pseudolosowość jest obliczeniową relaksacją prawdziwej losowości. (Uogólnimy tę perspektywę, omawiając pojęcie nierozróżnialności w Rozdziale 7.)

Niech teraz $G : \{0, 1\}^n \rightarrow \{0, 1\}$ będzie funkcją i zdefiniuje Dist jako rozkład na ciągach n -bitowych uzyskany poprzez wybranie uniforma $s \in \{0, 1\}^n$ i wypisanie $G(s)$. Wtedy G jest generatorem pseudolosowym wtedy i tylko wtedy, gdy rozkład Dist jest pseudolosowy.

Definicja formalna. Jak omówiono powyżej, G jest generatorem pseudolosowym, jeśli żaden skuteczny wyróżnik nie może wykryć, czy otrzymuje串 znaków wyjściowy przez G , czy串 wybrany jednolicie i losowo. Podobnie jak w definicji 3.9, sformalizowano to poprzez wymaganie, aby każdy wydajny algorytm zwracał 1 z prawie takim samym prawdopodobieństwem, gdy podano $G(s)$ (dla jednorodnych nasion) lub jednolity串 znaków. (Definicja równoważna do definicji 3.8 znajduje się w ćwiczeniu 3.5.) Definicję uzyskujemy w układzie asymptotycznym, pozwalając, aby parametr bezpieczeństwa n określił długość ziarna. Następnie nalegamy, aby G było obliczalne za pomocą wydajnego algorytmu. Ze względu dów technicznych wymagamy również, aby wyjście G było dłuższe niż wejście; w przeciwnym razie G nie jest zbyt przydatne ani interesujące.

DEFINICJA 3.14 Niech G będzie wielomianem i niech G będzie deterministycznym algorytmem wielomianowo-czasowym takim, że dla dowolnego n i dowolnego wejścia $s \in \{0, 1\}^n$, wynikiem $G(s)$ jest串 znaków o długości (n). Mówimy, że G jest generatorem pseudolosowym, jeśli spełnione są następujące warunki:

1. (Rozwinęcie cie:) Dla każdego n zachodzi to, że $(n) > n$.
2. (Pseudolosowość:) Dla dowolnego algorytmu D jest on nieistotny funkcja negl taka, że

$$\Pr[D(G(s)) = 1] = \Pr[D(r) = 1] = \text{negl}(n),$$

gdzie za pierwsze prawdopodobieństwo przyjmuje się równomierny wybór $s \in \{0, 1\}^n$ i losowość D , a za drugie prawdopodobieństwo równomierny wybór $r \in \{0, 1\}^n$ i losowość D .

Nazywamy współczynnikiem ekspansji G .

Podajemy przykład niepewnego generatora pseudolosowego, który pozwala poznać jego właściwość z definicją.

Przykład 3.15

Zdefiniuj $G(s)$ na wyjście s , po którym następuje串 $i=1s_i$, więc i jest współczynnikiem rozszerzenia G wynosi $(n) = n + 1$. Wynik G można łatwo odróżnić od jednorodnego.

Rozważmy następujący efektywny wyróżnik D : na wejściu串 w , na wyjściu jest 1 wtedy i tylko wtedy, gdy końcowy bit w jest równy XOR wszystkich poprzedzających bitów w .

Ponieważ ta właściwość obowiązuje dla wszystkich串ów wprowadzanych przez G , mamy

$\Pr[D(G(s)) = 1] = 1$. Z drugiej strony, jeśli w jest jednorodne, ostatni bit w jest jednolity, więc c $\Pr[D(w) = 1] = 1$ | jest stała i nie jest pomijalna, co to G nie jest generatorem pseudolosowym. (Zauważ, że D nie zawsze jest „poprawne”, ponieważ czasami zwraca 1, nawet jeśli otrzyma jednolity ciąg znaków. Nie zmienia to faktu, że D jest dobrym wyróżnikiem.)

Dyskusja. Rozkład na wyjściu generatora pseudolosowego G jest daleki od równomiernego. Aby to zobaczyć, rozważmy przypadek, że $(n) = 2n$, a więc c G podwaja długość swojego wejścia. W ramach rozkładu równomiernego na $\{0, 1\}^{2n}$ każdy z 2^{2n} możliwych ciągów jest wybierany z prawdopodobieństwem dokładnie 2^{-2n} . Dla kontrastu, rozważ rozkład wyników G (kiedy G jest uruchamiane na jednolitym ziarnie). Kiedy G otrzymuje dane wejściowe o długości n, liczba różnych ciągów w zakresie G wynosi co najwyżej 2^n . Zatem częśc strun o długości $2n$ mieszcząca się w zakresie G wynosi co najwyżej $2n/2^n$ i widzimy, że zdecydowana większość strun o długości $2n$ nie występuje jako wyniki G^{2n} .

Oznacza to w szczególności, że rozróżnienie mieć przy ciągiem losowym a ciągiem pseudolosowym przy nieograniczonym czasie jest trywialne. Niech G będzie jak powyżej i rozważmy wyróżnik D w czasie wykładowczym, który działa w następujący sposób: D(w) daje 1 wtedy i tylko wtedy, gdy istnieje s $\in \{0, 1\}^n$ taki, że $G(s) = w$. (Obliczenia te przeprowadza się w czasie wykładowczym poprzez wyczerpujące obliczenie $G(s)$ dla każdego s $\in \{0, 1\}^n$). Przypomnijmy, że zgodnie z zasadą Kerckhoffa specyfikacja G jest znana D. Teraz, gdyby w zostało wypisane przez G, wówczas D wyprowadza 1 z prawdopodobieństwem 1. W przeciwnieństwie do tego, jeśli w jest równomiernie rozłożone w $\{0, 1\}^{2n}$, wówczas prawdopodobieństwo, że istnieje s z $G(s) = w$ wynosi co najwyżej 2^{-n} , oraz więc c D wyprowadza w tym przypadku 1 z prawdopodobieństwem co najwyżej 2^{-n} . Więc

$$\Pr[D(r) = 1] = \Pr[D(G(s)) = 1] = 1 - 2^{-n},$$

który jest duży. To tylko kolejny przykład ataku brute-force i nie zaprzecza pseudolosowości G, ponieważ atak nie jest skuteczny.

Nasienie i jego długość. Zarodek generatora pseudolosowego jest analogiczny do klucza kryptograficznego używanego w schemacie szyfrowania, a zarodek musi być wybrany w sposób jednolity i utrzymywany w tajemnicy przed jakimkolwiek przeciwnikiem. Inną ważną kwestią, wynikającą z powyższego omówienia ataków siłowych, jest to, że s muszą być wystarczająco długie, aby nie było możliwe wyliczenie wszystkich możliwych nasion. W sensie asymptotycznym rozwiązuje się to poprzez ustawienie długości ziarna równej parametrowi bezpieczeństwa, tak że wyczerpujące przeszukanie wszystkich możliwych nasion wymaga wykładowczego czasu. W praktyce nasiono musi być na tyle długie, aby nie było możliwości wypróbowania wszystkich możliwych nasion w określonym terminie.

O istnieniu generatorów pseudolosowych. Czy istnieją generatory pseudolosowe? Z pewnością wydają się one trudne do skonstruowania i słusznie można zadać pytanie, czy istnieje algorytm spełniający Definicję 3.14. Chociaż nie wiemy, jak bezwarunkowo udowodnić istnienie generatorów pseudolosowych, mamy mocne powody, aby wierzyć, że istnieją. Po pierwsze, mogą być

skonstruowane przy raczej słabym założeniu, że istnieją funkcje jednokierunkowe (co jest prawda, jeśli pewne problemy, takie jak rozkład na czynniki dużych liczb, są trudne); zostanie to omówione szczegółowo w rozdziale 7. Mamy także kilka praktycznych konstrukcji potencjalnych generatorów pseudolosowych zwanych szyframi strumieniowymi, dla których nie są znane żadne skuteczne wyróżniki. (Później wprowadzimy jeszcze silniejsze prymitywy zwane szyframi blokowymi.) Następnie przedstawiamy ogólny przegląd szyfrów strumieniowych i omawiamy konkretne szyfry strumieniowe.

Szyfry strumieniowe

Nasza definicja generatora pseudolosowego jest ograniczona na dwa sposoby: współczynnik rozszerzalności jest stały, a generator wytworzy całą moc wyjściową „za jednym razem”. Szyfry strumieniowe, stosowane w praktyce do tworzenia generatorów pseudolosowych, działają nieco inaczej. Pseudolosowe bity wyjściowe szyfru strumieniowego są tworzone stopniowo i na żądanie, dzięki czemu aplikacja może zażądać dokładnie tylu bitów pseudolosowych, ile potrzeba. Poprawia to wydajność (ponieważ aplikacja może zażądać mniejszej liczby bitów, jeśli jest to wystarczające) i elastyczność (ponieważ nie ma górnej granicy liczby bitów, których można zażądać).

Formalnie postrzegamy szyfr strumieniowy² jako parę deterministycznych algorytmów (`Init`, `GetBits`) gdzie:

- `Init` przyjmuje jako wejście ziarno s , a opcjonalny wektor inicjujący IV , i wyprowadza stan początkowy $st0$.
- `GetBits` przyjmuje jako dane wejściowe informację o stanie sti , i wyprowadza bit y oraz zaktualizowany stan $sti+1$. (W praktyce y jest blokiem kilku bitów; dla ogólności i uproszczenia traktujemy y tutaj jako pojedynczy bit.)

Mając szyfr strumieniowy i dowolny pożądany współczynnik rozszerzenia, możemy zdefiniować algorytm G mapujący wejścia o długości n na wyjścia o długości (n) . Algorytm po prostu uruchamia `Init`, a następnie kilkakrotnie uruchamia `GetBits` .

ALGORYTM 3.16

Konstruowanie G z (`Init`, `GetBits`)

Wejście: Ziarno s i opcjonalny wektor inicjujący IV

Wyjście: y_1, \dots, y

$st0 := Init(s, IV)$
dla $i = 1$ do :

$(y_i, sti) := GetBits(sti-1)$
return y_1, \dots, y

²Zastosowana tutaj terminologia nie jest całkowicie standardowa i należy pamiętać, że „szyfr strumieniowy” jest używany przez różnych ludzi na różne (ale powiązane) sposoby. Na przykład niektórzy używają go w odniesieniu do G (patrz poniżej), podczas gdy inni używają go w odniesieniu do Konstrukcji 3.17, gdy jest on tworzony za pomocą G .

Szyfr strumieniowy jest bezpieczny w podstawowym sensie, jeśli nie przyjmuje IV i dla dowolnego wielomianu $z(n) > n$, skonstruowana powyżej funkcja G jest generatorem pseudolosowym ze współczynnikiem rozszerzalności. Pokrótkie omawiamy jedną z możliwych koncepcji bezpieczeństwa dla szyfrów strumieniowych, które używają IV w sekcji 3.6.1.

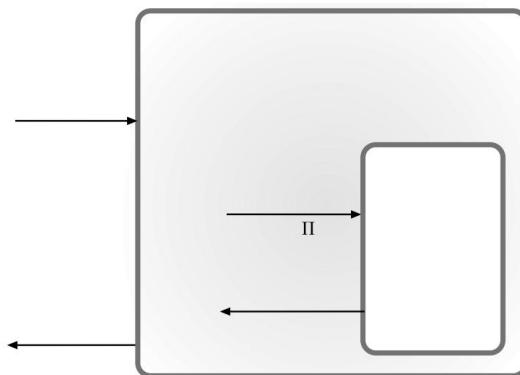
3.3.2 Dowody redukcyjne

Jeżeli chcemy udowodnić, że dana konstrukcja jest bezpieczna obliczeniowo, to musimy oprzeć się na nieudowodnionych założeniach³ (chyba że schemat jest bezpieczny informacyjnie). Naszą strategią będzie założenie, że jakiś problem matematyczny jest trudny lub że jakiś prymityw kryptograficzny niskiego poziomu jest bezpieczny, a nastąpi udowodnienie, że dana konstrukcja oparta na tym problemie/prymitywie jest bezpieczna przy tym założeniu. W sekcji 1.4.2 wyjaśniliśmy już szczegółowo, dlaczego takie podejście jest preferowane, więc nie będę tutaj powtarzać tych argumentów.

Dowód na to, że konstrukcja kryptograficzna jest bezpieczna, o ile jakiś leżący u jej podstaw problem jest trudny, zazwyczaj przebiega poprzez przedstawienie wyraźnej redukcji pokazującej, jak przekształcić dowolnego wydajnego przeciwnika A, któremu uda się „złamać” konstrukcję w wydajny algorytm A, który rozwiązuje problem uznawany za trudny. Ponieważ jest to tak ważne, szczegółowo omówimy ogólny zarys etapów takiego dowodu. (W książce zobaczymy wiele konkretnych przykładów, zaczynając od dowodu Twierdzenia 3.18.) Zaczynamy od założenia, że pewnego problemu X nie można rozwiązać (w pewnym ścisłe określonym sensie) żadnym algorymem czasu wielomianowego, z wyjątkiem znikome prawdopodobieństwa. Chcemy udowodnić, że pewna konstrukcja kryptograficzna Π jest bezpieczna (znowu w pewnym sensie ścisłe określona). Dowód przebiega w następujących krokach (patrz także rysunek 3.1):

1. Ustal jakiś skuteczny (tzn. probabilistyczny wielomian w czasie) atak przeciwnika A Π . Oznacz prawdopodobieństwo sukcesu tego przeciwnika przez $\epsilon(n)$.
2. Skonstruj wydajny algorytm A zwany „redukcją”, który próbuje rozwiązać problem X, wykorzystując przeciwnika A jako podprogram. Ważne jest tutaj to, że A nie wie nic o tym, jak A działa; jedynie, co A wie, to to, że A spodziewa się ataku Π . Zatem, mając pewną instancję wejściową x problemu X, nasz algorytm A będzie symulował dla A instancję Π w taki sposób, że:
 - (a) O ile A może stwierdzić, oddziałuje z Π . Oznacza to, że widok A, gdy jest uruchamiany jako podprogram przez A, powinien być rozłożony identycznie (lub przynajmniej blisko) widoku A, gdy oddziałuje on z samym Π . (b) Jeśli A uda się „złamać” instancję Π symulowaną przez A, powinno to umożliwić A rozwiązywanie instancji x , która została podana, przynajmniej z prawdopodobieństwem odwrotnym wielomianu $1/p(n)$.

³W szczególności więcej wiadomości kryptografii wymaga niepotwierdzonego założenia, że $P = N P$.



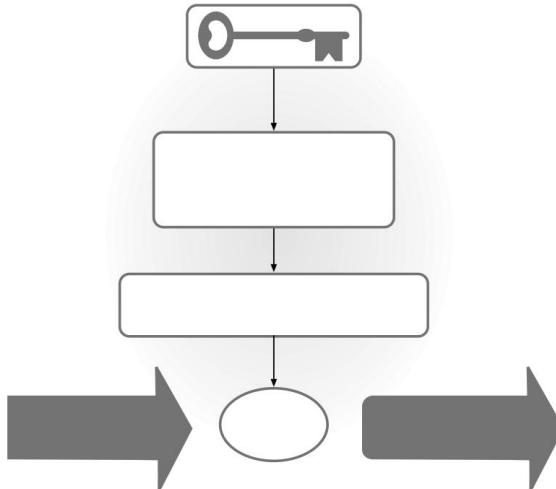
RYSUNEK 3.1: Ogólny przegląd dowodu bezpieczeństwa poprzez redukcję .

3. Podsumowując, z 2(a) i 2(b) wynika, że A rozwiązuje X z prawdopodobieństwem $\epsilon(n)/p(n)$. Jeśli $\epsilon(n)$ nie jest nieistotne, to $\epsilon(n)/p(n)$ też nie jest. Co więcej, jeśli A jest efektywne, to otrzymujemy efektywny algorytm A rozwiązujący X z nieznaniedbywalnym prawdopodobieństwem, co jest sprzeczne z początkowym założeniem.
4. Biorąc pod uwagę nasze założenia dotyczące X, dochodzimy do wniosku, że żadnemu efektywnemu przeciwnikowi A nie uda się przełamać Π z nieznaniedbywalnym prawdopodobieństwem. Inaczej mówiąc, Π jest bezpieczne obliczeniowo.

W poniższej sekcji zilustrujemy dokładnie powyższy pomysł: pokażemy, jak wykorzystać dowolny generator pseudolosowy G do skonstruowania schematu szyfrowania; udowadniamy, że schemat szyfrowania jest bezpieczny, pokazując, że każdy atakujący, który może „zlamać” schemat szyfrowania, może zostać użyty do odróżnienia wyniku G od jednolitego ciągu. Przy założeniu, że G jest generatorem pseudolosowym, schemat szyfrowania jest bezpieczny.

3.3.3 Bezpieczny schemat szyfrowania o stałej długości

Generator pseudolosowy zapewnia naturalny sposób skonstruowania bezpiecznego schematu szyfrowania o stałej długości z kluczem krótszym niż wiadomość. Przypomnijmy, że w jednorazowym bloku (patrz sekcja 2.2) szyfrowanie odbywa się poprzez XOR losowego bloku z wiadomością. Wniosek jest taki, że zamiast tego możemy użyć podkładki pseudolosowej. Zamiast jednak dzielić się tą długą, pseudolosową podkładką, nadawca i odbiorca mogą zamiast tego dzielić ziarno, które w razie potrzeby służy do generowania tej podkładki (patrz rysunek 3.2). to ziarno będzie krótsze niż podkładka, a zatem krótsze niż wiadomość. Jeśli chodzi o bezpieczeństwo, intuicja jest taka, że pseudolosowy ciąg „wygląda losowo” dla dowolnego przeciwnika działającego w czasie wielomianowym, w związku z czym ograniczony obliczeniowo podsłuchujący nie jest w stanie rozróżnić wiadomości zaszyfrowanej przy użyciu klucza jednorazowego od wiadomości zaszyfrowanej przy użyciu tego „pseudo-” -schemat szyfrowania podkładki czasowej.



RYSUNEK 3.2: Szyfrowanie za pomocą generatora pseudolosowego.

Schemat szyfrowania. Popraw pewną długość wiadomości i niech G będzie generatorem pseudolosowym ze współczynnikiem rozwinięcia (to znaczy $|G(s)| = |s|$). Przypomnijmy, że schemat szyfrowania definiują trzy algorytmy: algorytm generowania klucza Gen , algorytm szyfrowania Enc i algorytm deszyfrowania Dec .

Algorytm generowania klucza jest trywialny: $\text{Gen}(1^n)$ po prostu wyprowadza jednolity klucz o długości n . Szyfrowanie polega na zastosowaniu G do klucza (który służy jako zarodek) w celu uzyskania podkładki, która jest następnie dodawana XOR ze zwykłym tekstem. Podczas deszyfrowania stosuje się G do klucza i XOR wynikowy element zaszyfrowany w celu odzyskania wiadomości. Schemat jest opisany formalnie w Konstrukcji 3.17. W sekcji 3.6.1 opisujemy, jak szyfry strumieniowe są wykorzystywane do praktycznej implementacji wariantu tego schematu.

KONSTRUKCJA 3.17

Niech G będzie generatorem pseudolosowym ze współczynnikiem rozszerzalności. Zdefiniuj schemat szyfrowania kluczem prywatnym dla wiadomości o następującej długości:

- Gen : na wejściu 1^n , wybierz uniform $k \in \{0, 1\}^n$ i wypisz jako klawisz.
 - Enc : na wejściu klucz $k \in \{0, 1\}^n$ i wiadomość $m \in \{0, 1\}^n$ wyprowadzają tekst $c^{(n)}$, zaszyfrowany

$$\text{do } := G(k) \oplus m.$$
- wejściu klucz $k \in \{0, 1\}^n$ wyprowadź wiadomość $m := G(k) \oplus c$.

Schemat szyfrowania klucza prywatnego oparty na dowolnym generatorze pseudolosowym.

TWIERDZENIE 3.18 Jeżeli G jest generatorem pseudolosowym, to Konstrukcja 3.17 jest schematem szyfrowania kluczem prywatnym o stałej długości, którego szyfrowanie jest nieroróżnialne w obecności podsłuchującego.

DOWÓD Niech Π oznacza konstrukcję 3.17. Pokazujemy, że Π spełnia definicję 3.8.

Mianowicie pokazujemy, że dla dowolnego probabilistycznego przeciwnika A w czasie wielomianowym istnieje pomijalna funkcja negl taka, że

$$\Pr_{\Pi}[\text{PrivKeav } A, \Pi(n)] = 1 - \frac{1}{2} + \text{zaniebywanie}(n). \quad (3.2)$$

Intuicja jest taka, że jeśli Π użyłby jednolitej podkładki zamiast pseudolosowej podkładki $G(k)$, wówczas powstały schemat byłby identyczny ze schematem jednorazowego szyfrowania podkładki, a A nie byłby w stanie poprawnie zgadnąć, która wiadomość została zaszyfrowana z prawdopodobieństwem lepiej niż $1/2$. Zatem, jeśli równanie (3.2) nie jest spełnione, wówczas A musi w sposób dorozumiany odróżniać wynik G od losowego ciągu znaków. Wyrażamy to wyraźnie, pokazując redukcję ; mianowicie poprzez pokazanie, jak używać A do skonstruowania skutecznego wyróżnika D , którego właściwość polega na tym, że zdolność D do odróżnienia wyniku G od jednolitego ciągu jest bezpośrednio powiązana ze zdolnością A do określenia, która wiadomość została zaszyfrowana przez Π . Bezpieczeństwo G implikuje zatem bezpieczeństwo Π .

Niech A będzie dowolnym przeciwnikiem ppt . Konstruujemy wyróżnik D , który jako dane wejściowe przyjmuje ciąg w i którego celem jest określenie, czy w zostało wybrane jednolito (tj. w jest „ciągiem losowym”), czy też w zostało wygenerowane przez wybór jednolitego k i obliczenie $w := G(k)$ (tzn. w jest „łańcuchem pseudolosowym”). Konstruujemy D w taki sposób, aby emulował eksperyment podsłuchiwanego dla A , jak opisano poniżej, i obserwujemy, czy A powiedzie się , czy nie. Jeśli A się powiedzie, D zgaduje, że w musi być ciągiem pseudolosowym, natomiast jeśli A się nie powiedzie, D zgaduje, że w jest ciągiem losowym. Szczegółowo:

Wyróżnik D : D jest

podawany jako sygnał wejściowy $w \in \{0, 1\}^n$. (Zakładamy, że n można wyznaczyć na podstawie (n) .)

1. Uruchom $A(1n)$, aby otrzymać parę wiadomości $m_0, m_1 \in \{0, 1\}^n$.

2. Wybierz bit jednolity $b \in \{0, 1\}$. Zestaw $c := w \oplus mb$.

3. Podaj c A i uzyskaj wynik b . Wyjście 1 jeśli $b = b$ i

w przeciwnym razie wyprowadź 0.

D wyraźnie przebiega w czasie wielomianowym (zakładając, że A tak).

Przed analizą zachowania D definiujemy zmodyfikowany schemat szyfrowania $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, to jest dokładnie jednorazowy schemat szyfrowania padu, np. włączamy teraz parametr bezpieczeństwa, który określa długość wiadomości do zaszyfrowania. Oznacza to, że $\text{Gen}(1n)$ wyprowadza jednolity klucz k długości (n) oraz szyfrowanie wiadomości $m_2(n)$ przy użyciu klucza $k \in \{0, 1\}^n$

jest szyfrogramem $c := k \oplus m$. (Odszyfrowanie można przeprowadzić w zwykły sposób, ale jest to nieistotne dla dalszych działań.) Oznacza to, że całkowita tajemnica jednorazowego bloku

$$\Pr_{A, \Pi}[\text{PrivKeav}_{A, \Pi}(n) = 1] = 2^{-\frac{1}{n}}. \quad (3.3)$$

Aby przeanalizować zachowanie D, główne obserwacje są następujące:

- Jeśli w zostanie wybrane równomiernie z $\{0, 1\}^n$, nastąpi pnie widok A, gdy jest uruchamiany jako $A(n)$, podprogram D ma rozkład identyczny z widokiem A w eksperymentie $\text{PrivKeav}(n)$. Dzieje się tym tak, ponieważ gdy A jest uruchamiane jako podprogram przez $D(w)$ w A, Π w przypadku, A otrzymuje tekst zaszyfrowany $c = w \oplus m$, gdzie $w \in \{0, 1\}^n$ jest jednolite. Ponieważ D wyprowadza 1 dokładnie wtedy, gdy A zakończy się sukcesem w eksperymencie podsłuchiwania, zatem mamy (por. Równanie (3.3))

$$\Pr_w[\{0, 1\}^n | D(w) = 1] = \Pr_{A, \Pi}[\text{PrivKeav}_{A, \Pi}(n) = 1] = 2^{-\frac{1}{n}}. \quad (3.4)$$

(Indeks dolny pierwszego prawdopodobieństwa po prostu wyraźnie wskazuje, że w jest tam wybierane równomiernie spośród $\{0, 1\}^n$.)

- Jeśli zamiast tego w jest generowane przez wybranie uniformu $k \in \{0, 1\}^n$, a nastąpi pnie ustawienie $w := G(k)$, widok A, gdy jest wykonywany jako podprogram przez D, jest rozkładany identycznie jak widok A w eksperymencie $\text{PrivKeav}_{A, \Pi}(n)$. Dzieje się tak, ponieważ A, gdy jest uruchamiany jako podprogram przez D, otrzymuje teraz zaszyfrowany tekst $c = w \oplus m$, gdzie $w = G(k)$ dla jednolitego $k \in \{0, 1\}^n$. Zatem,

$$\Pr_k[\{0, 1\}^n | D(G(k)) = 1] = \Pr_{A, \Pi}[\text{PrivKeav}_{A, \Pi}(n) = 1]. \quad (3.5)$$

Ponieważ G jest generatorem pseudolosowym (i ponieważ D działa w czasie wielomianowym), wiemy, że istnieje pomijalna funkcja negl taka, że

$$\Pr_w[\{0, 1\}^n | D(w) = 1] = \Pr_k[\{0, 1\}^n | D(G(k)) = 1] = \text{negl}(n).$$

Korzystając z równań (3.4) i (3.5) widzimy to

$$\frac{1}{2} = \Pr_{A, \Pi}[\text{PrivKeav}_{A, \Pi}(n) = 1] = \text{negl}(n),$$

co implikuje $\Pr_{A, \Pi}[\text{PrivKeav}_{A, \Pi}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$. Ponieważ A było dowolne przeciwnika, co kończy dowód, że Π ma nierozróżnialne szyfry w obecności podsłuchującego. ■

Łatwo jest pogubić się w szczegółach dowodu i zastanawiać się, czy udało się coś zyskać w porównaniu z jednorazowym podkładem; w końcu podkładka jednorazowa szyfruje także wiadomość -bit, XORując ją ciągiem -bit! Celem tej konstrukcji jest oczywiście to, że ciąg -bitowy G(k) może być dużo

dłuższy niż klucz współdzielony k . W szczególności, stosując powyższy schemat, możliwe jest bezpieczne zaszyfrowanie pliku o wielkości 1 Mb przy użyciu jedynie klucza 128-bitowego.

Opierając się na tajemnicy obliczeniowej, ominęliśmy w ten sposób niemożność wynikającą z Twierdzenia 2.10, które stwierdza, że każdy doskonale tajny schemat szyfrowania musi używać klucza co najmniej tak długiego jak wiadomość.

Redukcje – dyskusja. Nie udowadniamy bezwarunkowo, że Konstrukcja 3.17 jest bezpieczna. Raczej udowadniamy, że jest to bezpieczne przy założeniu, że G jest generatorem pseudolosowym. Takie podejście polegające na zredukowaniu bezpieczeństwa konstrukcji wyższego poziomu do prymitywu niższego poziomu ma wiele zalet (omówionych w sekcji 1.4.2). Jedną z tych zalet jest to, że ogólnie łatwiej jest zaproektować prymityw niższego poziomu niż wyższy; ogólnie łatwiej jest też bezpośrednio analizować algorytm G w odniesieniu do definicji niższego poziomu niż analizować bardziej złożony schemat Π w odniesieniu do definicji wyższego poziomu. Nie oznacza to, że skonstruowanie generatora pseudolosowego jest „łatwe”, a jedynie, że jest łatwiejsze niż zbudowanie schematu szyfrowania od zera. (W tym przypadku schemat szyfrowania nie robi nic poza XOR wyjściem generatora pseudolosowego z wiadomością, więc nie jest to do końca prawdę. Jednakże zobaczymy bardziej złożone konstrukcje i w tych przypadkach możliwość zmniejszenia zadania do prostszego ma ogromne znaczenie.) Kolejną zaletą jest to, że po skonstruowaniu odpowiedniego G można go wykorzystać jako składnik różnych innych schematów.

Betonowe bezpieczeństwo. Chociaż Twierdzenie 3.18 i jego dowód mają układ asymptotyczny, możemy łatwo dostosować dowód, aby ograniczyć konkretne bezpieczeństwo schematu szyfrowania w kategoriach konkretnego bezpieczeństwa G . Ustal jakąś wartość n na pozostałą część tej dyskusji, i niech Π oznacza teraz Konstrukcję 3.17, używając tej wartości n . Założymy, że G jest (t, ε) -pseudolosowym (dla danej wartości n) w tym sensie, że dla wszystkich wyróżników D działających w czasie co najwyżej t mamy

$$\Pr[D(r) = 1] = \Pr[D(G(s)) = 1] = \varepsilon. \quad (3.6)$$

(Pomyśl o $t = 2$ i $\varepsilon = 80^2 / 60$, chociaż dokładne wartości nie mają znaczenia dla naszej dyskusji.) Twierdzimy, że Π jest $(t - c, \varepsilon)$ -pewne dla pewnej (małej) stałej c w tym sensie, że dla wszystkie A działające w czasie co najwyżej $t - c$ mamy

$$\Pr_{A, \Pi}[\text{PrivKeav} = 1] = 1 + \varepsilon. \quad (3.7)$$

(Zauważ, że powyższe są teraz stałymi liczbami, a nie funkcjami n , ponieważ nie jesteśmy tutaj w układzie asymptotycznym.) Aby to zobaczyć, niech A będzie dowolnym przeciwnikiem działającym w czasie co najwyżej $t - c$. Wyróżnik D , skonstruowany w dowodzie Twierdzenia 3.18, ma bardzo niewielkie obciążenie poza uruchomieniem A ; ustalenie c odpowiednio zapewnia, że D działa w czasie co najwyżej t . Z naszego założenia o konkretnym bezpieczeństwie G wynika równanie (3.6); postępując dokładnie tak jak w dowodzie Twierdzenia 3.18, otrzymujemy Równanie (3.7).

3.4 Silniejsze koncepcje bezpieczeństwa

Do tej pory rozważaliśmy stosunkowo słabą definicję bezpieczeństwa, w której przeciwnik jedynie biernie podsłuchuje pojedynczy szyfrrogram przesyłany pomiędzy uczciwymi stronami. W tej sekcji rozważymy dwa silniejsze pojęcia dotyczące bezpieczeństwa. Przypomnijmy, że definicja bezpieczeństwa określa cel bezpieczeństwa i model ataku. Definiując pierwsze nowe pojęcie bezpieczeństwa, modyfikujemy cel bezpieczeństwa; po drugie wzmacniamy model ataku.

3.4.1 Bezpieczeństwo przy wielokrotnym szyfrowaniu

Definicja 3.8 dotyczy przypadku, gdy komunikujące się strony przesyłają pojedynczy tekst zaszyfrowany, który jest obserwowany przez osobę podsłuchującą. Byłoby jednak wygodnie, gdyby komunikujące się strony mogły wysyłać sobie wiele szyfrrogramów – wszystkie wygenerowane przy użyciu tego samego klucza – nawet jeśli osoba podsłuchująca mogłaby je wszystkie obserwować. Do takich zastosowań potrzebny jest bezpieczny schemat szyfrowania umożliwiający szyfrowanie wielu wiadomości.

Zaczynamy od odpowiedniej definicji bezpieczeństwa dla tego ustalenia. Podobnie jak w przypadku definicji 3.8, najpierw wprowadzamy odpowiedni eksperyment zdefiniowany dla dowolnego schematu szyfrowania Π , przeciwnika A i parametru bezpieczeństwa n: Eksperyment podsłuchiwanie wielu

wiadomości $\text{PrivKmult } A, \Pi(n)$:

- Dany jest przeciwnik A wejście 1 n i wyprowadzenie pary = (m_0, \dots, m_{t-1}) i M lista wiadomości M o równej długości M (m_0, \dots, m_{t-1}), gdzie $|m_0, \dots, m_{t-1}| = |m_1, \dots, m_{t-1}|$ dla wszystkich $i = 1, \dots, t-1$

2. Klucz k jest generowany poprzez uruchomienie $\text{Gen}(1n)$ i wybierany jest jednolity bit b $\in \{0, 1\}$. Dla wszystkich i obliczany jest szyfrrogram ci $\text{Enck}(mb, i)$ i lista C = (c_1, \dots, c_t) jest przekazywana A.

3. A wyprowadza trochę b.

4. Wynik eksperymentu definiuje się jako 1, jeśli $b = b$ i 0 w przeciwnym razie.

Definicja bezpieczeństwa jest taka sama jak poprzednio, z tą różnicą, że teraz nawiązuje do powyższego eksperymentu.

DEFINICJA 3.19 Schemat szyfrowania kluczem prywatnym $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ charakteryzuje się nieroróżnialnymi wielokrotnymi szyfrowaniami w obecności podsłuchującego, jeśli dla wszystkich probabilistycznych przeciwników A w czasie wielomianowym istnieje pomijalna funkcja negl taka, że

$$\Pr[\text{PrivKmult } A, \Pi(n) = 1] \xrightarrow[+ \text{ zaniedbywanie}(n)]{} 2^{-n}$$

gdzie prawdopodobieństwo przejmuje losowość zastosowaną przez A i losowość zastosowaną w eksperymencie.

Każdy schemat, który ma nieroróżnialne wielokrotne szyfrowanie w obecności podsłuchującego, wyraźnie spełnia również definicję 3.8, ponieważ eksperyment PrivKeav odpowiada specjalnemu przypadkowi PrivKmult, w którym przeciwnik generuje dwie listy zawierające tylko jedną wiadomość każda. W rzeczywistości nasza nowa definicja jest ściślejsza niż Definicja 3.8, jak pokazano poniżej.

PROPOZYCJA 3.20 Istnieje schemat szyfrowania z kluczem prywatnym, który charakteryzuje się nieroróżnialnymi szyfrowaniami w obecności podsłuchującego, ale nie nieroróżnialnymi wielokrotnymi szyfrowaniami w obecności podsłuchującego.

DOWÓD Nie musimy daleko szukać, aby znaleźć przykład schematu szyfrowania spełniającego tę tezę. Jednorazowa podkładka jest całkowicie tajna, dlatego też posiada szyfrowanie nie do odróżnienia w obecności podsłuchującego. Pokazujemy, że nie jest to bezpieczne w rozumieniu Definicji 3.19. (Omówiliśmy ten atak już w Rozdziale 2; tutaj analizujemy jedynie atak w odniesieniu do Definicji 3.19.)

Konkretnie, rozważmy następującego przeciwnika A atakującego schemat (w sensie zdefiniowanym przez eksperyment PrivKmult): A wyprowadza $M = (0, 0)$ i $\theta(0, 1)$. (Pierwsza M_1 zawiera dwukrotnie ten sam tekst jawnny, podczas gdy druga zawiera dwie różne wiadomości.) Niech $C = (c_1, c_2)$ będą listą szyfrogramów, które otrzymuje A. Jeśli $c_1 = c_2$, wówczas A wyprowadza $b = 0$; w przeciwnym razie A wyprowadza $b = 1$.

Przeanalizujmy teraz prawdopodobieństwo, że $b = b$. Kluczową kwestią jest to, że podkładka jednorazowa jest deterministyczna, więc dwukrotne zaszyfrowanie tej samej wiadomości (przy użyciu tego samego klucza) daje ten sam tekst zaszyfrowany. Zatem jeśli $b = 0$, to w tym przypadku musimy mieć $c_1 = c_2$ i A na wyjściu 0. Z drugiej strony, jeśli $b = 1$, to za każdym razem zaszyfrowana jest inna wiadomość; stąd $c_1 \neq c_2$ i A wyjście 1.

Dochodzimy do wniosku, że A poprawnie wyprowadza $b = b$ z prawdopodobieństwem 1, a zatem schemat szyfrowania nie jest bezpieczny w odniesieniu do definicji 3.19. ■

Konieczność zaszyfrowania probabilistycznego. Powyższe może wydawać się wskazywać, że Definicja 3.19 nie jest możliwa do osiągnięcia przy użyciu jakiegokolwiek schematu szyfrowania. Ale w rzeczywistości jest to prawdą tylko wtedy, gdy schemat szyfrowania jest deterministyczny, a zatem wielokrotne zaszyfrowanie tej samej wiadomości (przy użyciu tego samego klucza) zawsze daje ten sam wynik. Jest to na tyle ważne, że można je przedstawić jako twierdzenie.

Twierdzenie 3.21 Jeżeli Π jest (bezstanowym⁴) schematem szyfrowania, w którym Enc jest deterministyczną funkcją klucza i wiadomości, to Π nie może mieć nieroróżnialnych wielokrotnych zaszyfrowań w obecności podsłuchującego.

Nie należy tego rozumieć jako zbyt mocnej definicji 3.19. Rzeczywiście,

⁴W sekcji 3.6.1 przekonamy się, że jeśli schemat szyfrowania jest stanowy, możliwe jest bezpieczne zaszyfrowanie wielu wiadomości, nawet jeśli zaszyfrowanie jest deterministyczne.

wyciek do osoby podsłuchującej fakt, że dwie zaszyfrowane wiadomości są takie same, może stanowić poważne naruszenie bezpieczeństwa. (Rozważmy np. scenariusz, w którym uczeń szyfruje serię odpowiedzi typu prawda/fałsz!)

Aby skonstruować schemat bezpieczny do szyfrowania wielu wiadomości, musimy zaprojektować schemat, w którym szyfrowanie jest losowe, tak aby w przypadku wielokrotnego szyfrowania tej samej wiadomości można było utworzyć różne teksty zaszyfrowane.

Może się to wydawać niemożliwe, ponieważ odszyfrowanie musi zawsze umożliwiać odzyskanie wiadomości. Wkrótce jednak przekonamy się, jak to osiągnąć.

3.4.2 Ataki za pomocą wybranego tekstu jawnego i bezpieczeństwo CPA

Ataki z wybranym tekstem jawnym wychwytują zdolność przeciwnika do sprawowania (częściowej) kontroli nad tym, co szyfrują uczciwe strony. Wyobraźmy sobie scenariusz, w którym dwie uczciwe strony dzielą klucz k , a atakujący może wpływać na te strony, aby zaszyfrowały wiadomości m_1, m_2, \dots (używając k) i wysyłać powstałe szyfrogramy kanałem widocznym dla atakującego. W pewnym momencie atakujący zauważa zaszyfrowany tekst odpowiadający jakiejś nieznanej wiadomości m zaszyfrowanej przy użyciu tego samego klucza k ; założymy nawet, że atakujący wie, że m jest jedną z dwóch możliwości m_0, m_1 . Zabezpieczenie przed atakami z użyciem wybranego tekstu jawnego oznacza, że nawet w tym przypadku atakujący nie jest w stanie stwierdzić, która z tych dwóch wiadomości została zaszyfrowana, z prawdopodobieństwem znacznie większym niż przypadkowe zgadywanie. (Na razie wracamy do przypadku, w którym podsłuchujący otrzymuje tylko jedno szyfrowanie nieznanej wiadomości.) Wkrótce powrócimy do rozpatrywania przypadku wielu wiadomości.)

Ataki z wybranym tekstem jawnym w świecie rzeczywistym. Czy ataki za pomocą wybranego tekstu jawnego stanowią realny problem? Na początek należy pamiętać, że ataki z wybranym tekstem jawnym obejmują także ataki ze znanym tekstem jawnym —w których atakujący wie, jakie wiadomości są zaszyfrowane, nawet jeśli nie może ich wybrać —co jest szczególnym przypadkiem. Co więcej, istnieje kilka rzeczywistych scenariuszy, w których przeciwnik może mieć znaczący wpływ na to, jakie wiadomości będą zaszyfrowane. Prosty przykład podaje osoba atakująca wpisującą na terminalu, który z kolei szyfruje i wysyła wszystko, co przeciwnik wpisuje, używając klucza współdzielonego ze zdalnym serwerem (i nieznanego atakującemu). W tym przypadku atakujący dokładnie kontroluje, co zostanie zaszyfrowane, ale schemat szyfrowania powinien pozostać bezpieczny, gdy zostanie użyty —przy użyciu tego samego klucza —do szyfrowania danych innego użytkownika.

Co ciekawe, ataki z użyciem wybranego tekstu jawnego były również z powodzeniem stosowane w ramach historycznych wysiłków mających na celu złamanie wojskowych schematów szyfrowania. Na przykład podczas II wojny światowej Brytyjczycy umieścili miny w określonych miejscach, wiedząc, że Niemcy — gdy znajdą te miny — zaszyfrują te lokalizacje i odesiąt je z powrotem do kwatery głównej. Te zaszyfrowane wiadomości zostały wykorzystane przez kryptoanalityków w Bletchley Park do złamania niemieckiego schematu szyfrowania.

Inny przykład podaje słynna historia dotycząca bitwy o Midway. W maju 1942 roku kryptoanalitycy Marynarki Wojennej Stanów Zjednoczonych przechwycili zaszyfrowaną wiadomość od Japończyków, którą udało im się częściowo odszyfrować. Wynik w-

wskazał, że Japończycy planowali atak na AF, gdzie AF był fragmentem zaszyfrowanego tekstu, którego Stany Zjednoczone nie były w stanie odszyfrować. Z innych powodów Stany Zjednoczone uważały, że celem była wyspa Midway. Niestety, ich próby przekonania planistów z Waszyngtonu, że tak właśnie było, okazały się daremne; panowało powszechnie przekonanie, że celem nie może być Midway. Kryptoanalytycy Marynarki Wojennej opracowali następujący plan: poinstruowali siły amerykańskie w Midway, aby wysyłały fałszywą wiadomość, że ich zapasy słodkiej wody są niskie. Japończycy przechwycili tę wiadomość i natychmiast zgłosili swoim przełożonym, że „AF ma mało wody”. Kryptoanalytycy Marynarki Wojennej mieli teraz dowód, że AF odpowiada Midway i Stany Zjednoczone wysyłały w to miejsce trzy lotniskowce. W rezultacie Midway zostało uratowane, a Japończycy ponieśli znaczne straty. Bitwa ta była punktem zwrotnym w wojnie między USA i Japonią na Pacyfiku.

Kryptoanalytycy Marynarki Wojennej przeprowadzili tutaj atak wybranym tekstem jawnym, ponieważ udało im się wpłynąć na Japończyków (aczkolwiek w okrąży sposób), aby zaszyfrowali słowo „Midway”. Gdyby japoński schemat szyfrowania był zabezpieczony przed atakami z użyciem wybranego tekstu jawnego, ta strategia amerykańskich kryptoanalytyków nie zadziałałaby (a historia mogła potoczyć się zupełnie inaczej)!

Bezpieczeństwo CPA. W formalnej definicji modelujemy ataki wybranym tekstem jawnym, dając przeciwnikowi A dostęp p do wyroczni szyfrującej, postrzeganej jako „czarna skrzynka”, która szyfruje wybrane przez A wiadomości przy użyciu klucza k nieznanego A. Oznacza to, że wyobrażamy sobie A ma dostęp p do „wyroczni” $\text{Enck}(\cdot)$; kiedy A pyta tę wyrocznię, podając jej wiadomość m jako dane wejściowe, wyrocznia zwraca w odpowiedzi szyfrogram c $\text{Enck}(m)$. (Kiedy Enc jest losowy, wyrocznia wykorzystuje nową losowość za każdym razem, gdy odpowiada na pytanie.) Przeciwnik może adaptacyjnie wchodzić w interakcję z wyrocznią szyfrującą tyle razy, ile chce.

Rozważmy następujący eksperyment zdefiniowany dla dowolnego schematu szyfrowania $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, przeciwnik A i wartość n dla parametru bezpieczeństwa:

Eksperyment nieroróżnialności CPA PrivKcpa

$A, \Pi(N)$:

1. Klucz k jest generowany poprzez uruchomienie $\text{Gen}(1^n)$.
 2. Przeciwnik A otrzymuje dane wejściowe 1 n i dostęp p do Oracle do $\text{Enck}(\cdot)$ i wysyła parę komunikatów m0, m1 o tej samej długości.
 3. Wybierany jest bit jednolity b {0, 1}, a następnie tekst zaszyfrowany c $\text{Enck}(mb)$ jest obliczane i podawane A.
 4. Przeciwnik A nadal ma dostęp p do $\text{Enck}(\cdot)$, i wyprowadza trochę b.
 5. Wynik eksperymentu definiuje się jako 1, jeśli b = b, i 0 w przeciwnym razie.
- W pierwszym przypadku mówimy, że A odnosi sukces.

DEFINICJA 3.22 Schemat szyfrowania kluczem prywatnym $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ ma nieroróżnicalne szyfrowanie w przypadku ataku wybranym tekstem jawnym lub jest bezpieczny CPA, jeśli dla wszystkich probabilistycznych przeciwników A w czasie wielomianowym istnieje pomijalna funkcja negl takie, że

$$\Pr_{A, \Pi}[\text{PrivKcpa}_1(n) = 1] = \text{negl}(n),$$

gdzie prawdopodobieństwo przejmuje losowość zastosowaną przez A , a także losowość zastosowaną w eksperymencie.

Bezpieczeństwo CPA dla wielokrotnych szyfrowań

Definicję 3.22 można rozszerzyć na przypadek wielokrotnych szyfrowań w taki sam sposób, w jaki Definicję 3.8 można rozszerzyć, aby uzyskać Definicję 3.19, tj. poprzez użycie list tekstów jawnych. W tym przypadku przyjmujemy inne podejście, które jest nieco prostsze i ma tę zaletę, że modeluje atakujących, którzy mogą adaptacyjnie wybierać teksty jawnie do zaszyfrowania, nawet po obserwacji poprzednich tekstów zaszyfrowanych. W niniejszej definicji dajemy atakującemu dostęp p do „lewej lub prawej” wyroczni LRk,b , która po wprowadzeniu pary wiadomości $m0, m1$ o równej długości oblicza zaszyfrowany tekst $c = Enck(mb)$ i zwraca C . Oznacza to, że jeśli $b = 0$, to przeciwnik otrzymuje szyfrowanie „lewego” tekstu jawnego, a jeśli $b = 1$, to otrzymuje zaszyfrowanie „prawego” tekstu jawnego. Tutaj b jest losowo wybranym bitem na początku eksperymentu i podobnie jak w poprzednich definicjach celem atakującego jest odgadnąć b . To uogólnia poprzednią definicję bezpieczeństwa wielu wiadomości (Definicja 3.19), ponieważ zamiast wyświetlać listy $(m0, 1, \dots, m0, t)$ i $(m1, 1, \dots, m1, t)$, których jedna wiadomość zostanie zaszyfrowana, atakujący może teraz sekwencyjnie wysyłać zapytania do $LRk,b(m0, 1, m1, 1, \dots, LRk,b(m0, t, m1, t))$. Obejmuje to również dostęp p atakującego do wyroczni szyfrującej, ponieważ atakujący może po prostu wysłać zapytanie do $LRk,b(m, m)$, aby uzyskać $Enck(m)$.

Teraz formalnie definiujemy ten eksperiment, zwany eksperimentem LR-Oracle.

Niech Π będzie schematem szyfrowania, A przeciwnikiem, a n parametrem bezpieczeństwa:

Eksperiment LR-Oracle $\text{PrivKLR-cpa}(n): A, \Pi$

1. Klucz k jest generowany poprzez uruchomienie $\text{Gen}(1n)$.
2. Wybierany jest bit jednolity $b \in \{0, 1\}$.
3. Przeciwnik A otrzymuje dane wejściowe 1 i dostęp p Oracle do $LRk,b(\cdot, \cdot)$, jak zdefiniowano powyżej.
4. Przeciwnik A wysyła bit b .
5. Wynik eksperimentu definiuje się jako 1 , jeśli $b = b$, i 0 w przeciwnym razie. W pierwszym przypadku mówimy, że A odnosi sukces.

DEFINICJA 3.23 Schemat szyfrowania kluczem prywatnym Π ma nieroźnialne wielokrotne szyfrowania w przypadku ataku wybranym tekstem jawnym lub jest bezpieczny CPA dla wielokrotnych szyfrowań, jeśli dla wszystkich probabilistycznych przeciwników A w czasie wielomianowym istnieje pomijalna funkcja negl taka, że

$$\Pr_{\substack{A, \Pi}}[\text{PrivKLR-cpa}(\Pi) = 1] = \frac{1}{2^{\text{zaniebywanie}(\Pi)}},$$

gdzie prawdopodobieństwo przejmuje losowość zastosowaną przez A i losowość zastosowaną w eksperymencie.

Nasza wcześniejsza dyskusja pokazuje, że bezpieczeństwo CPA dla wielu szyfrowań jest co najmniej tak samo silne, jak wszystkie nasze poprzednie definicje. W szczególności, jeśli schemat szyfrowania z kluczem prywatnym jest zabezpieczony CPA w przypadku wielu szyfrowań, to z pewnością jest on również bezpieczny CPA. Co ważne, obowiązuje również sytuacja odwrotna; oznacza to, że bezpieczeństwo CPA oznacza bezpieczeństwo CPA dla wielu szyfrowań. (To kontrastuje z przypadkiem podsłuchiwania przeciwników; zobacz Twierdzenie 3.20.) Następujące twierdzenie przedstawiamy tutaj bez dowodu; podobny wynik w przypadku ustwienia klucza publicznego udowodniono w sekcji 11.2.2.

TWIERDZENIE 3.24 Każdy schemat szyfrowania klucza prywatnego, który jest bezpieczny CPA, jest również bezpieczny CPA w przypadku wielokrotnych szyfrowań.

Jest to znacząca zaleta techniczna zabezpieczenia CPA: wystarczy udowodnić, że schemat jest bezpieczny CPA (dla pojedynczego szyfrowania), a następnie „za darmo” otrzymujemy, że jest on bezpieczny również przy wielokrotnym szyfrowaniu.

Zabezpieczenie przed atakami z użyciem wybranego tekstu jawnego jest obecnie minimalnym pojęciem bezpieczeństwa, jaki powinien spełniać schemat szyfrowania, chociaż coraz powszechniejsze jest wymaganie nawet silniejszych właściwości zabezpieczeń omówionych w sekcji 4.5.

Wiadomości o stałej i dowolnej długości. Kolejną zaletą pracy z definicją bezpieczeństwa CPA jest to, że pozwala nam ona traktować schematy szyfrowania o stałej długości bez utraty ogólności. W szczególności, biorąc pod uwagę dowolny schemat szyfrowania o stałej długości bezpieczny CPA $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, możliwe jest skonstruowanie schematu szyfrowania zabezpieczonego CPA $\Pi' = (\text{Gen}', \text{Dec}')$ dla wiadomości Enc o dowolnej długości łatwo. Dla uproszczenia powiedzmy, że Π szyfruje wiadomości o długości 1 bita (choćż wszystko, co mówimy, rozciąga się w naturalny sposób niezależnie od długości wiadomości obsługiwanej przez Π). Pozostaw Gen taki sam dla każdej wiadomości m (o dowolnej długości), jak Gen. Define Enc' $kk(m) = \text{Enck}(m_1), \dots, \text{Enck}(m_l)$, gdzie m jest zabezpieczone CPA, co oznacza i-ty bit m . Szyfrowanie De-Enc odbywa się w sposób naturalny. Π' jeśli Π jest; dowód wynika z Twierdzenia 3.24.

Istnieją skuteczniejsze sposoby szyfrowania wiadomości o dowolnej długości niż dostosowywanie schematu szyfrowania o stałej długości w powyższy sposób. Zbadamy to szerzej w sekcji 3.6.

3.5 Konstruowanie schematów szyfrowania z zabezpieczeniem CPA

Przed skonstruowaniem schematów szyfrowania zabezpiecz wybranym tekstem jawnym pod adresem-Tacks, najpierw wprowadzimy ważne pojęcie funkcji pseudolosowych.

3.5.1 Funkcje pseudolosowe i szyfry blokowe

Funkcje pseudolosowe (PRF) uogólniają pojęcie generatorów pseudolosowych. Teraz zamiast rozważać „losowo wyglądające” ciągi znaków, rozważmy funkcje „losowo wyglądające”. Podobnie jak w naszej wcześniejszej dyskusji na temat pseudolosowości, nie ma wiele sensu twierdzenie, że jakakolwiek stała funkcja $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ jest pseudolosowa (w ten sam sposób, w jaki nie ma sensu powiedzieć, że każda ustalona funkcja jest losowa). Dlatego zamiast tego musimy odnieść się do pseudolosowości rozkładu funkcji. Taki rozkład jest indukowany w sposób naturalny poprzez uwzględnienie funkcji kluczowych, zdefiniowanych poniżej.

Funkcja z kluczem $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ jest funkcją z dwoma wejściami, gdzie pierwsze wejście nazywane jest kluczem i oznaczane k . Mówimy, że F jest efektywne, jeśli istnieje algorytm czasu wielomianowego, który oblicza $F(k, x)$ przy danych k i x . (Będzie nam interesować tylko wydajnymi funkcjami z kluczem.) W typowym użyciu wybiera się k i ustala klucz k , a następnie interesuje nas funkcja pojedynczego wejścia $F_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$ określone przez $F_k(x) = F(k, x)$. Parametr bezpieczeństwa n określa długość klucza, długość wejściową i długość wyjściową.

Oznacza to, że kojarzymy z F trzy funkcje i out ; dla dowolnego klucza k , wejścia x i $in(n)$ funkcja zdefiniowana tylko dla wejścia x i $in(n)$ w tym przypadku $F_k(x) = out(n)$.

O ile nie zaznaczono inaczej, dla uproszczenia zakładamy, że F zachowuje długość, co oznacza, że $klucz(n) = in(n) = out(n) = n$.

Oznacza to, że ustalając klucz $k \in \{0, 1\}^n$, otrzymujemy funkcję F_k odwzorowującą n -bitowe ciągi wejściowe na n -bitowe ciągi wyjściowe.

Funkcja kluczowana F indukuje naturalny rozkład funkcji danych poprzez wybranie jednolitego klucza $k \in \{0, 1\}^n$, a następnie uwzględnienie wynikowej funkcji pojedynczego wejścia F_k . F nazywamy pseudolosową, jeśli funkcja F_k (dla klucza jednolitego k) jest nie do odróżnienia od funkcji wybranej jednoznacznie losowo ze zbioru wszystkich funkcji mających tę samą dziedzinę i zakres; to znaczy, jeśli żaden skuteczny przeciwnik nie jest w stanie rozróżnić – w pewnym sensie, który dokładniej zdefiniujemy poniżej – czy oddziałuje z F_k (dla jednolitego k) czy z f (gdzie f jest wybierane równomiernie ze zbioru wszystkich funkcji odwzorowujących n -bitowe wejścia na wyjścia n -bitowe).

Ponieważ losowe wybranie funkcji jest mniej intuicyjne niż losowe wybranie ciągu znaków, warto poświęcić temu pomysłowi nieco więcej czasu. Rozważmy zbiór $Func_n$ wszystkich funkcji odwzorowujących ciągi n -bitowe na ciągi n -bitowe. Zbiór ten jest skończony i wybranie funkcji jednolitej odwzorowującej n -bitowe ciągi na n -bitowe ciągi oznacza jednorodne wybranie elementu z tego zbioru. Jak duży jest $Func_n$? Funkcję f określa się, podając jej wartość w każdym punkcie swojej dziedziny. Możemy zobaczyć dowolną funkcję (w skończonej dziedzinie) jako dużą tabelę przeglądową, która przechowuje

$f(x)$ w wierszu tabeli oznaczonym jako x . Dla $f \in \text{Funcn}$ tablica przeglądowa dla f ma $2n$ wierszy (po jednym dla każdego punktu dziedziny $\{0, 1\}^n$), przy czym każdy wiersz zawiera n -bitowy ciąg znaków (ponieważ zakres f wynosi $\{0, 1\}^n$). Łącząc wszystkie wpisy tabeli, widzimy, że dowolną funkcję w Funcn można przedstawić za pomocą ciągu znaków o długości $2n \cdot n$. Co więcej, ta zgodność jest jednoznaczna, ponieważ każdy ciąg o długości $2n \cdot n$ (tj. każda tabela zawierająca $2n$ wpisów o długości n) definiuje unikalną funkcję w Funcn . Zatem rozmiar Funcn jest dokładnie liczbą ciągów o długości $n \cdot 2^n$, czyli $|\text{Funcn}| = 2^n \cdot 2^n$. Oglądanie funkcji w formie tabeli przeglądowej stanowi kolejny użyteczny sposób myślenia.

o wyborze funkcji jednolitej $f \in \text{Funcn}$: Jest to dokładnie równoważne równomierнемu wybraniu każdego wiersza w tabeli przeglądowej f . Oznacza to w szczególności, że wartości $f(x)$ i $f(y)$ (dla dowolnych dwóch wejść $x = y$) są jednorodne i niezależne. Możemy z wyprzedzeniem zobaczyć, jak ta tabela przeglądowa jest zapełniana losowymi wpisami, zanim f zostanie ocenione na jakimkolwiek wejściu, lub możemy zobaczyć, czy wpisy tabeli są wybierane równomiernie „w locie”, w razie potrzeby, za każdym razem, gdy f jest oceniane na nowym wejściu, na którym f nie zostało wcześniej ocenione.

Wracając do dyskusji na temat funkcji pseudolosowych, przypomnijmy, że funkcja pseudolosowa jest funkcją kluczową F taką, że F_k (dla $k \in \{0, 1\}^n$ wybranych równomiernie losowo) jest nie do odróżnienia od f (dla $f \in \text{Funcn}$ wybranych jednolicie losowo). Ten pierwszy jest wybierany z rozkładu na (co najwyżej) funkcje w 2^n różnych funkcji, przy czym ta ostatnia jest wybrana spośród wszystkich $2^n \cdot 2^n$

Funkcja. Mimo to „zachowanie” tych funkcji musi wyglądać tak samo dla dowolnego wyróżnika wielomianowego w czasie.

Pierwszą próbą sformalizowania pojęcia funkcji pseudolosowej byłoby postępowanie w taki sam sposób, jak w definicji 3.14. Oznacza to, że moglibyśmy wymagać, aby każdy wyróżnik czasu wielomianowego D , który otrzyma opis funkcji pseudolosowej F_k , dał na wyjściu 1 z „prawie” takim samym prawdopodobieństwem, jak wtedy, gdy otrzyma opis funkcji losowej f . Jednakże ta definicja jest niewłaściwa, ponieważ opis funkcji losowej ma długość wykładniczą (podaną w tabeli przeglądowej o długości $n \cdot 2^n$), podczas gdy D ogranicza się do przebiegu w czasie wielomianowym. Zatem D nie miałby nawet wystarczającej ilości czasu na zapoznanie się z całością swoich danych wejściowych.

Definicja daje zatem D dostęp p do wyroczni O , która jest albo równa F_k (dla jednolitej funkcji k), albo f (dla funkcji jednolitej f). Wyróżnik D może zapytać swoją wyrocznię w dowolnym punkcie x , w odpowiedzi na co wyrocznia zwraca $O(x)$.

Traktujemy wyrocznię jak czarną skrzynkę w taki sam sposób, jak wtedy, gdy zapewniliśmy przeciwnikowi dostęp p do algorytmu szyfrowania w definicji ataku wybranym tekstem jawnym. Tutaj jednak wyrocznia oblicza funkcję deterministyczną i dlatego zwraca ten sam wynik, jeśli zostanie odpytana dwukrotnie na tym samym wejściu.

(Z tego powodu możemy założyć bez utraty ogólności, że D nigdy nie odpytuje wyroczni dwa razy na tym samym wejściu.) D może swobodnie wchodzić w interakcję ze swoją wyrocznią, dobierając zapytania adaptacyjnie w oparciu o wszystkie poprzednie wyniki. Ponieważ jednak D działa w czasie wielomianowym, może zadawać tylko wielomianowo wiele zapytań.

Przedstawiamy teraz formalną definicję. (Definicja zakłada, że F jest dugością zachowującą jedynie dla uproszczenia.)

DEFINICJA 3.25 Niech $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ będzie efektywną funkcją z kluczem zachowującą długość. F jest funkcją pseudolosową, jeżeli dla wszystkich probabilistycznych wyróżników wielomianowo-czasowych D istnieje funkcja pomijalna taka, że:

$$\Pr[D F_k(\cdot) = 1] = \Pr[D f(\cdot) = 1] = \text{negl}(n),$$

gdzie pierwsze prawdopodobieństwo przyjmuje się za jednorodny wybór $k \in \{0, 1\}^n$ i losowość D , a drugie prawdopodobieństwo za równomierny wybór $f \in \text{Func}_n$ i losowość D .

Ważnym punktem jest to, że D nie otrzymuje klucza k . Nie ma sensu wymagać, aby F_k było pseudolosowe, jeśli k jest znane, ponieważ przy danym k trywialne jest odróżnienie wyroczni dla F_k od wyroczni dla f . (Wszystko, co musi zrobić wyróżnik, to zaprosić wyrocznię w dowolnym punkcie x , aby uzyskać odpowiedź y i porównać ją z wynikiem $y := F_k(x)$, który sam oblicza, używając znanej wartości k .)

Wyrocznia dla F_k wróci $y = y$, podczas gdy wyrocznia dla funkcji losowej będzie miała $y = y$ z prawdopodobieństwem tylko 2^{-n} . Oznacza to, że jeśli k zostanie ujawnione, wszelkie twierdzenia o pseudolosowości F_k nie będą dalej aktualne. Weźmy konkretny przykład, jeśli F jest funkcją pseudolosową, to mając dostęp do Oracle do F_k (dla jednorodnego k), musi być trudno znaleźć dane wejściowe x , dla których $F_k(x) = 0$ (ponieważ byłoby trudno znaleźć takie wejście dla prawdziwej losowej funkcji f). Ale jeśli znane jest k , znalezienie takiego wejścia może być łatwe.

Przykład 3.26 Jak

zwykle możemy zapoznać się z definicją, przyglądając się niepewnemu przykładowi. Zdefiniuj kluczowaną funkcję zachowującą długość F przez $F(k, x) = k \oplus x$. Dla dowolnego wejścia x wartość $F_k(x)$ ma rozkład jednostajny (gdy k jest jednorodne). Niemniej jednak F nie jest pseudolosowe, ponieważ jego wartości w dowolnych dwóch punktach są skorelowane. Konkretnie, rozważ wyróżnik D , który odpytuje swoją wyrocznię O w dowolnych, odrębnych punktach x_1, x_2 , aby otrzymać wartości $y_1 = O(x_1)$ i $y_2 = O(x_2)$ i wyprowadza 1 wtedy i tylko wtedy, gdy $y_1 \oplus y_2 = x_1 \oplus x_2$. Jeśli $O = F_k$, dla dowolnego k , to D daje w wyniku 1. Z drugiej strony, jeśli $O = f$ dla f wybranego równomiernie z Func_n , to prawdopodobieństwo, że $f(x_1) \oplus f(x_2) = x_1 \oplus x_2$ jest dokładnie równe prawdopodobieństwu, że $f(x_2) = x_1 \oplus x_2$ lub 2^{-n} , a D daje 1 w tym prawdopodobieństwie. Różnica wynosi $|1 - 2^{-n}|$, co nie jest pomijalne.

Permutacje pseudolosowe/szyfry blokowe

Niech Perm_n będzie zbiorem wszystkich permutacji (tzn. bijekcji) na $\{0, 1\}^n$. Oglądając dowolne $f \in \text{Perm}_n$ jako tabelę przeglądową, tak jak poprzednio, mamy teraz dodatkowe ograniczenie polegające na tym, że wpisy w dowolnych dwóch różnych wierszach muszą być różne. Mamy 2^n różnych możliwości wpisu w pierwszym wierszu tabeli; kiedy naprawimy ten wpis, pozostałe nam tylko $2^n - 1$ możliwości wyboru w drugim rzęzie i tak dalej. Widzimy zatem, że rozmiar Perm_n wynosi $(2^n)!$.

Niech F będzie funkcją kluczową. Nazywamy F permutacją kluczową, jeśli ponadto dla wszystkich k klucz(n) funkcja $F_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$ jest jedyne do jednego (tj. F_k jest permutacją). Nazywamy długość bloku F . Tak jak poprzednio, jeśli nie zaznaczono inaczej, zakładamy, że F zachowuje długość, a zatem $\text{klucz}(n) = \text{in}(n) = n$. Permutacja z kluczem jest skuteczna, jeśli istnieje algorytm czasu wielomianowego do obliczania $F_k(x)$ przy danych k i x , a algorytm czasu wielomianowego do obliczania $F_k^{-1}(y)$ przy danych k i y . To jest, powinien być zarówno efektywnie obliczalny, jak i efektywnie odwracalny, biorąc pod uwagę k .

Definicja tego, co oznacza, że wydajna permutacja F z kluczem jest permutacją pseudolosową, jest dokładnie analogiczna do definicji 3.25, z tą tylko różnicą, że teraz wymagamy, aby F_k było nie do odróżnienia od jednolitej permutacji, a nie od jednolitej funkcji. Oznacza to, że wymagamy, aby żaden wydajny algorytm nie rozróżnił dostępu do F_k (dla jednolitego klucza k) od dostępu do f (dla jednolitego $f \in \text{Perm}_n$). Okazuje się, że jest to jedynie wybór estetyczny, ponieważ gdy długość bloku jest wystarczająco duża, losowa permutacja sama w sobie jest nie do odróżnienia od funkcji losowej. Intuicyjnie wynika to z faktu, że funkcja jednostajna f wygląda identycznie jak permutacja jednostajna, chyba że zostaną znalezione różne wartości x i y , dla których $f(x) = f(y)$, gdyż w takim przypadku funkcja nie może być permutacją. Jednak prawdopodobieństwo znalezienia takich wartości x , y przy użyciu wielomianowej liczby zapytań jest znikome. (Wynika to z wyników Załącznika).

TWIERDZENIE 3.27 Jeśli F jest permutacją pseudolosową i dodatkowo $\text{in}(n) = n$, to F jest także funkcją pseudolosową.

Jeśli F jest permutacją z kluczem, wówczas schematy kryptograficzne oparte na F mogą wymagać od uczciwych stron obliczenia odwrotności F oprócz obliczenia samego F_k . Potencjalnie powoduje to nowe problemy związane z bezpieczeństwem. W szczególności może być teraz konieczne nałożenie bardziej rygorystycznego wymogu, aby F_k było nie do odróżnienia od jednolitej permutacji, nawet jeśli wyróżnik ma dodatkowo zapewniony dostęp do odwrotności permutacji. Jeśli F ma tę właściwość, nazywamy ją silną permutacją pseudolosową.

DEFINICJA 3.28 Niech $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ będzie efektywną, zachowującą długość permutacją z kluczem. F jest silną permutacją pseudolosową, jeśli dla wszystkich probabilistycznych wyróżników wielomianowo-czasowych D istnieje funkcja pomialna taka, że:

$$\Pr[D F_k(\cdot), F^{k^{-1}}(\cdot)(1^n) = 1] = \Pr[D f(\cdot), f^{-1}(\cdot)(1^n) = 1] = \text{negl}(n),$$

gdzie za pierwsze prawdopodobieństwo przyjmuje się równomierny wybór $k \in \{0, 1\}^n$ i losowość D , a za drugie prawdopodobieństwo równomierny wybór $f \in \text{Perm}_n$ i losowość D .

Oczywiście każda silna permutacja pseudolosowa jest również permutacją pseudolosową.

Blokuj szyfry. W praktyce szyfry blokowe są zaprojektowane tak, aby stanowić bezpieczne instancje (silnych) permutacji pseudolosowych z pewną stałą długością klucza i długością bloku. Podejścia do szyfrów blokowych i niektóre popularne szyfry blokowe omawiamy w rozdziale 6. Na potrzeby niniejszego rozdziału szczegóły tych konstrukcji są nieistotne i na razie po prostu zakładamy, że istnieją (silne) permutacje pseudolosowe.

Funkcje pseudolosowe i generatory pseudolosowe. Jak można się spodziewać, istnieje ścisły związek pomiędzy funkcjami pseudolosowymi i generatorami pseudolosowymi. Dość łatwo jest skonstruować generator pseudolosowy G z funkcji pseudolosowej F , po prostu oceniąc F na szeregu różnych danych wejściowych; np. możemy zdefiniować $G(s)$

$$\stackrel{\text{def}}{=} F(s(1))F(s(2)) \cdots F(s(l)) \text{ dla dowolnego}$$

żadanego s . Jeżeli F zastąpiono funkcją jednostajną f , wynik G byłby jednolity; dlatego też, jeśli zamiast tego użyje się F , wynik będzie pseudolosowy.

Jesteś poproszony o formalne udowodnienie tego w ćwiczeniu 3.14.

Mówiąc bardziej ogólnie, możemy wykorzystać powyższy pomysł do skonstruowania szyfru strumieniowego (Init, GetBits), który akceptuje wektor inicjujący IV. (Patrz sekcja 3.3.1.) Jedyna różnica polega na tym, że zamiast oceniać F na ustalonej sekwencji wejściowej, oceniamy F na wejściach $IV + 1, IV + 2, \dots, 1, 2, 3, \dots$

KONSTRUKCJA 3.29

Niech F będzie funkcją pseudolosową. Zdefiniuj szyfr strumieniowy (Init, GetBits), gdzie każde wywołanie GetBits generuje n bitów w następujący sposób:

- Init: na wejściu $s \in \{0, 1\}^n$ i $IV \in \{0, 1\}^n$ • GetBits: , ustaw $st0 := (s, IV)$.
na wejściu $sti = (s, IV)$, oblicz $IV := IV + 1$ i ustaw $y := F(sti)$ i $sti+1 := (s, IV)$.
Wyjście $(y, sti+1)$.

Szyfr strumieniowy z dowolnego szyfru funkcyjnego/blokowego pseudolosowego.

Chociaż szyfry strumieniowe można konstruować z szyfrów blokowych, dedykowane szyfry strumieniowe stosowane w praktyce zazwyczaj mają lepszą wydajność, szczególnie w środowiskach o ograniczonych zasobach. Z drugiej strony szyfry strumieniowe wydają się być gorzej rozumiane (w praktyce) niż szyfry blokowe, a pewnością co do ich bezpieczeństwa jest niższa. Dlatego zaleca się stosowanie szyfrów blokowych (ewentualnie poprzez konwersję ich najpierw na szyfry strumieniowe), gdy tylko jest to możliwe.

Biorąc pod uwagę drugi kierunek, generator pseudolosowy G natychmiast daje funkcję pseudolosową F o małej długości bloku. Konkretnie, powiedzmy, że G ma współczynnik rozszerzalności $n \cdot 2 t(n)$. Możemy zdefiniować funkcję kluczowaną $F : \{0, 1\}^n \rightarrow \{0, 1\}^{t(n)}$ następujący sposób: aby obliczyć $F_k(i)$, najpierw oblicz $G(k)$ i zinterpretuj wynik jako tabelę przeglądową zawierającą $2t(n)$ wierszy każdy zawierający n bitów; wyprowadź i wiersz. Działa to w czasie wielomianowym tylko wtedy, gdy $t(n) = O(\log n)$. Możliwe jest, choć bardziej skomplikowane, skonstruowanie funkcji pseudolosowych o dużej długości bloku z generatorów pseudolosowych; to jest

pokazano w Sekcji 7.5. Z kolei generatory pseudolosowe można skonstruować w oparciu o pewne problemy matematyczne, uważane za trudne. Istnienie funkcji pseudolosowych opartych na tych trudnych problemach matematycznych stanowi jeden z niesamowitych osiągnięć współczesnej kryptografii.

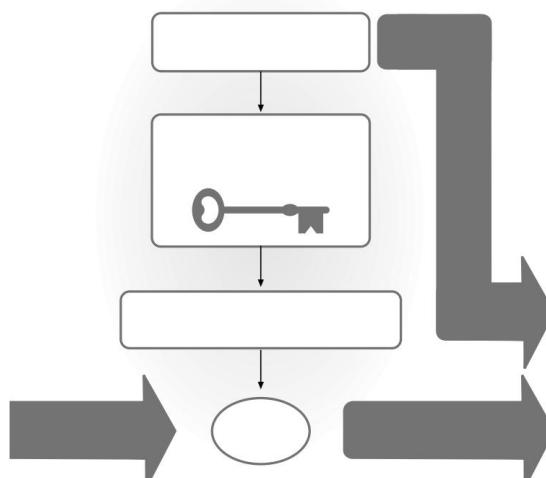
3.5.2 Szyfrowanie bezpieczne CPA z funkcji pseudolosowych

Koncentrujemy się tutaj na skonstruowaniu schematu szyfrowania o stałej długości, bezpiecznego dla CPA. Z tego, co powiedzieliśmy na końcu sekcji 3.4.2, wynika, że istnieje schemat szyfrowania zabezpieczony przez CPA dla wiadomości o dowolnej długości. W sekcji 3.6 omówimy bardziej efektywne sposoby szyfrowania wiadomości o dowolnej długości.

Naiwną próbą skonstruowania bezpiecznego schematu szyfrowania na podstawie permutacji pseudolosowej jest zdefiniowanie $\text{Enck}(m) = F_k(m)$. Chociaż spodziewamy się, że „nie ujawnia to żadnych informacji o m ” (ponieważ jeśli f jest funkcją jednolitą, to $f(m)$ jest po prostu jednolitym n -bitowym ciągiem znaków), ta metoda szyfrowania jest deterministyczna i dlatego nie może być Bezpieczne dla CPA. W szczególności dwukrotne szyfrowanie tego samego tekstu jawnego da ten sam tekst zaszyfrowany.

Nasza bezpieczna konstrukcja jest losowa. W szczególności szyfrujemy, stosując funkcję pseudolosową do losowej wartości r (a nie do wiadomości) i XORując wynik za pomocą zwykłego tekstu. (Patrz rysunek 3.3 i konstrukcja 3.30.)

Można to ponownie postrzegać jako przykład XORowania podkładki pseudolosowej z tekstem jawnym, z główną różnicą polegającą na tym, że za każdym razem używana jest nowa podkładka pseudolosowa. (W rzeczywistości podkładka pseudolosowa jest „świeża” tylko wtedy, gdy funkcja pseudolosowa zostanie zastosowana do „świeżej” wartości, do której nigdy wcześniej nie została zastosowana. Chociaż jest możliwe, że losowe r będą drie równe wybranej wybranej wartości r wcześniej zdarzało się to z niskim prawdopodobieństwem.)



RYSUNEK 3.3: Szyfrowanie funkcją pseudolosową.

Dowody bezpieczeństwa oparte na funkcjach pseudolosowych. Zanim przejdziemy do dowodu, że powyższa konstrukcja jest zabezpieczona CPA, podkreślamy powszechny szablon używany w większości dowodów bezpieczeństwa (nawet poza kontekstem szyfrowania) w przypadku konstrukcji opartych na funkcjach pseudolosowych. Pierwszym krokiem takich dowodów jest rozważenie hipotetycznej wersji konstrukcji, w której funkcja pseudolosowa zostaje zastąpiona funkcją losową. Następnie argumentuje się – posługując się dowodem redukcyjnym – że modyfikacja ta nie wpływa znacząco na prawdopodobieństwo sukcesu atakującego. Pozostaje nam zatem analiza schematu wykorzystującego całkowicie losową funkcję. W tym momencie reszta dowodu zazwyczaj opiera się na analizie probabilistycznej i nie opiera się na żadnych założeniach obliczeniowych. Będziemy korzystać z tego szablonu dowodu kilka razy w tym i następny rozdziale.

BUDOWA 3.30

Niech F będzie funkcją pseudolosową. Zdefiniuj schemat szyfrowania kluczem prywatnym dla wiadomości o długości n w następujący sposób:

- Gen: na wejściu 1^n , wybierz uniform $k \in \{0, 1\}^n$ i wypisz go. • Enc: na wejściu klucz $k \in \{0, 1\}^n$ i wiadomość $m \in \{0, 1\}^n$ uniform $r \in \{0, 1\}^n$ i, wybierając wyprowadzenie tekstu zaszyfrowanego

$$do := r, Fk(r) \quad m.$$
- Dec: po wprowadzeniu klucza $k \in \{0, 1\}^n$ i tekstu zaszyfrowanego $c = r, s$, wyprowadź wiadomość w postaci zwykłego tekstu

$$m := Fk(r) \quad s.$$

Schemat szyfrowania bezpieczny CPA z dowolnej funkcji pseudolosowej.

TWIERDZENIE 3.31 Jeśli F jest funkcją pseudolosową, to Konstrukcja 3.30 jest schematem szyfrowania kluczem prywatnym z zabezpieczeniem CPA dla wiadomości o długości n .

DOWÓD Niech $\Pi = (\text{Gen}, \text{Dec})$ będzie schematem szyfrowania dokładnie takim samym jak $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ z Konstrukcją 3.30, z tą różnicą, że zamiast Fk używana jest prawdziwie losowa funkcja f . Oznacza to, że $\text{Gen}(1^n)$ wybiera funkcję jednolitą $f \in \text{Func}_n$, a Enc szyfruje tak samo jak Enc , z tą różnicą, że zamiast Fk używane jest f . (Ten zmodyfikowany schemat szyfrowania nie jest skuteczny. Ale dla celów dowodu nadal możemy go zdefiniować jako hipotetyczny schemat szyfrowania.)

Napraw dowolnego przeciwnika $\text{ppt } A$ i niech $q(n)$ będzie górną granicą liczby zapytań, które $A(1^n)$ wykonuje do swojej wyroczni szyfrującej. (Zauważ, że q musi być ograniczone w górę przez jakiś wielomian.) W pierwszym kroku dowodu pokażemy, że istnieje pomijalna funkcja negl taka, że

$$\Pr_{A, \Pi}[\text{PrivKcpa}_{A, \Pi}(n) = 1] = \Pr[\text{PrivKcpa}(n) = 1] = \text{negl}(n). \quad (3.8)$$

Udowodnimy to poprzez redukcję . Używamy A do skonstruowania wyróżnika D dla funkcji pseudolosowej F. Wyróżnik D ma dostęp p do jakiejś funkcji O, a jego celem jest ustalenie, czy ta funkcja jest „pseudolosowa” (tzn. równa Fk dla uniforma $k \in \{0, 1\}^n$) lub „losowy” (tj. równy f dla jednorodnego f = Funcn). Aby to zrobić, D emuluje eksperyment PrivKcpa dla A w sposób opisany poniżej i obserwuje, czy A się powiedzie, czy nie.

Jeśli A się powiedzie, D zauważa, że jego wyrocznia musi być funkcją pseudolosową, natomiast jeśli A się nie powiedzie, D zauważa, że jego wyrocznia musi być funkcją losową. Szczegółowo:

Wyróżnik D: D ma dane

wejście 1^n i dostęp p do wyroczni $O: \{0, 1\}^n \rightarrow \{0, 1\}^n$.

1. Uruchom A(1^n). Ilekróć A pyta swoją wyrocznię szyfrującą o wiadomość $m \in \{0, 1\}^n$, odpowiedź na to pytanie w następujący sposób: (a) Wybierz uniform $r \in \{0, 1\}^n$. (b) Zapytaj $O(r)$ i uzyskaj odpowiedź y . (c) Zwróć szyfrogram r, y m do A.
2. Kiedy A wysyła komunikaty $m_0, m_1 \in \{0, 1\}^n$, wybierz bit jednolity $b \in \{0, 1\}$, a następnie: (a) Wybierz jednolity $r \in \{0, 1\}^n$. (b) Zapytaj $O(r)$ i uzyskaj odpowiedź y . (c) Zwróć szyfrogram wyzwania r, y m_b do A.
3. Kontynuuj odpowiadanie na zapytania wyroczni szyfrowania A jak poprzednio, aż A wyśle bit b . Wyjście 1, jeśli $b = b$, i 0 w przeciwnym razie.

D działa w czasie wielomianowym, ponieważ A to robi. Kluczowe punkty są następujące:

1. Jeśli wyrocznia D jest funkcją pseudolosową, to widok A po uruchomieniu jako podprogram przez D rozkłada się identycznie jak widok A w eksperymencie PrivKcpa wybranym równomiernie losowo, a $A, \Pi(N)$. Dzieje się tak, ponieważ w tym przypadku kluczem jest k następnie pnie każde szyfrowanie odbywa się poprzez wybranie jednolitego r , obliczenie $y := F_k(r)$ i ustalenie szyfrogramu na $r, y \in m$, dokładnie tak jak w Construction 3.30. Zatem,

$$\Pr_{k \in \{0,1\}^n} D(F_k(\cdot))(1^n) = 1 = \Pr_{A, \Pi(n)} \text{PrivKcpa}(A, \Pi(n)) = 1, \quad (3.9)$$

gdzie podkreślamy, że k jest wybrane równomiernie po lewej stronie.

2. Jeśli wyrocznia D jest funkcją losową, to widok A, gdy jest wykonywany jako podprogram przez D, rozkłada się identycznie jak widok A w eksperymencie PrivKcpa(n). Można to zobaczyć dokładnie tak jak powyżej, z jedną różnicą A, Π oznacza to, że zamiast F_k używana jest funkcja jednostajna $f = \text{Funcn}$. Zatem,

$$\Pr_{f = \text{Funcn}} D(f(\cdot))(1^n) = 1 = \Pr_{A, \Pi(n)} \text{PrivKcpa}(A, \Pi(n)) = 1, \quad (3.10)$$

gdzie f jest wybierane jednolicie z Funcn po lewej stronie.

Zakładając, że F jest funkcją pseudolosową (i ponieważ D jest efektywne), istnieje pomijalna funkcja negl, dla której

$$\Pr[\text{re } F_k(\cdot) = 1] \approx 1 \quad \Pr[\text{re } f(\cdot) = 1] \approx \text{negl}(n).$$

Połączenie powyższego z równaniami (3.9) i (3.10) daje równanie (3.8).

W drugiej części dowodu wykażemy to

$$\Pr[\text{PrivK}_{A, \Pi}(n) = 1] = 1 - 2 \frac{1}{\pi} \frac{q(n)}{n^2}. \quad (3.11)$$

(Przypomnijmy, że $q(n)$ jest ograniczeniem liczby zapytań szyfrujących wykonanych przez A. Powyższe obowiązuje nawet jeśli nie nałożymy żadnych ograniczeń obliczeniowych na A.) Aby zobaczyć, że równanie (3.11) jest prawdziwe, zauważ, że za każdym razem, gdy wiadomość m jest szyfrowana w $\text{PrivK}_{A, \Pi}(n)$ (albo przez wyrocznię szyfrującą, albo gdy szyfr wezwania- A, Π tekst obliczany), wybiera się jednolity $r \in \{0, 1\}^n$ i zaszyfrowany tekst ustawia się na $r, f(r) \in m$. Niech r oznacza losowy ciąg znaków używany podczas generowania zaszyfrowanego tekstu wyzwania $r, f(r) \in m$. Istnieją dwie możliwości:

szyfrujących A 1. nigdy nie jest używane podczas odpowiadania na którykolwiek z wyroczni Zapytania o wartości r : W tym przypadku A nie dowiaduje się niczego o $f(r) \in m$ na podstawie swojej interakcji z wyrocznią szyfrującą (ponieważ f jest funkcją naprawdę losową). Oznacza to, że w przypadku A wartość $f(r) \in m$ poddana XOR z m jest równomiernie rozłożona i niezależna od reszty eksperymentu, a zatem prawdopodobieństwo, że A wyprowadza $b = b$ w tym przypadku jest dokładnie równe $1/2$ (jak w przypadku podkładki jednorazowej).

A-2. Wartość r jest używany podczas odpowiadania na co najmniej jedno z szyfrowań zapytania Oracle: W tym przypadku A może łatwo określić, czy zaszyfrowano m_0 czy m_1 . Dzieje się tak dlatego, że jeśli wyrocznia szyfrująca kiedykolwiek zwróci tekst zaszyfrowany $r, s \in m$ w odpowiedzi na żądanie zaszyfrowania wiadomości m , przeciwnik dowiaduje się, że $f(r) = s \in m$.

Jednakże, ponieważ A wykonuje co najwyżej $q(n)$ zapytań do swojej wyroczni szyfrującej (a zatem co najwyżej $q(n)$ wartości r jest używanych podczas odpowiadania na zapytania A szyfrującego-wyroczni) i ponieważ r jest wybierane jednolicie z $\{0, 1\}^n$, prawdopodobieństwo tego zdarzenia wynosi co najwyżej $q(n)/2^n$.

Niech Powtórz oznacza zdarzenie, w którym r jest używane przez wyrocznię szyfrującą podczas odpowiadania na co najmniej jedno z zapytań A. Jak właśnie omówiliśmy, prawdopodobieństwo Powtórzenia wynosi co najwyżej $q(n)/2^n$, a prawdopodobieństwo, że A powiedzie się w $\text{PrivK}_{A, \Pi}(n)$ jeśli Powtórzenie nie nastąpi, wynosi dokładnie $1/2$. Dlatego:

$$\begin{aligned} \Pr[\text{PrivK}_{A, \Pi}(n) = 1] &= \Pr[\text{PrivK}_{A, \Pi}(n) = 1 \mid \text{Powtórzenie}] + \Pr[\text{PrivK}_{A, \Pi}(n) = 1 \mid \overline{\text{Powtórzenie}}] \\ &= \Pr[\text{PrivK}_{A, \Pi}(n) = 1 \mid \text{Powtórzenie}] + \Pr[\text{PrivK}_{A, \Pi}(n) = 1 \mid \overline{\text{Powtórzenie}}] \\ &= \Pr[\text{Powtórzenie}] + \Pr[\text{PrivK}_{A, \Pi}(n) = 1 \mid \overline{\text{Powtórzenie}}] = \frac{q(n)}{n^2} + \frac{1}{2} = \frac{q(n) + n^2}{2n^2}. \end{aligned}$$

Łącząc powyższe z równaniem (3.8), widzimy, że istnieje pomijalna funkcja $q(n)$ negl taka, $\Pr[\text{PrivKcpa}(n) = 1] + \text{negl}(n)$. Ponieważ $A, \Pi \vdash n q(n) \xrightarrow{T_2} \underline{\text{że}}$

q jest wielomianem, jest $\frac{1}{2^n}$ pomijalne. Dodatkowo suma dwóch funkcji pomijalnych jest pomijalna, zatem istnieje funkcja pomijalna negl taka, że $\Pr[\text{PrivKcpa} + \text{negl}(n)]$, co kończy dowód. $A, \Pi \vdash (n) = 1 \xrightarrow{T_2}$ ■

Betonowe bezpieczeństwo. Powyższy dowód to pokazuje

$$\Pr[\text{PrivKcpa } A, \Pi(n) = 1] \xrightarrow{n=2} \frac{1}{2} \xrightarrow{q(n)} \text{negl}(n) + 2$$

dla jakiejś pomijalnej funkcji negl. Ostateczny termin zależy od bezpieczeństwa F jako funkcji pseudolosowej; jest ograniczony prawdopodobieństwem rozróżnienia algorytmu D (który ma mniej więcej taki sam czas działania jak przeciwnik A).

Termin ten reprezentuje granicę prawdopodobieństwa, że wartość r użyta do zaszyfrowania zaszyfrowanego tekstu wezwania została użyta do zaszyfrowania innej wiadomości.

3.6 Tryby pracy

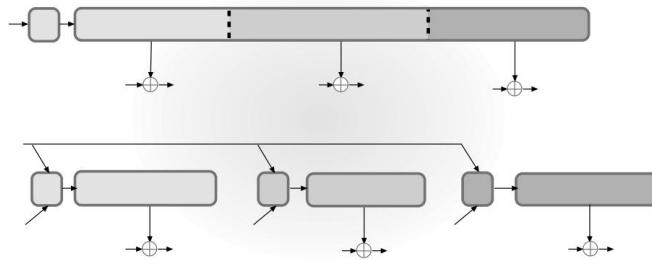
Tryby działania umożliwiają bezpieczne (i wydajne) szyfrowanie długich wiadomości przy użyciu szyfrów strumieniowych lub blokowych.

3.6.1 Tryby działania szyfru strumieniowego

Konstrukcja 3.17 umożliwia skonstruowanie schematu szyfrowania przy użyciu generatora pseudolosowego. Schemat ten ma dwie główne wady. Po pierwsze, jak przedstawiono, długość wiadomości, która ma zostać zaszyfrowana, musi być ustalona i znana z góry. Po drugie, system jest bezpieczny tylko pod kątem EAV, a nie CPA.

Szyfry strumieniowe, które można postrzegać jako elastyczne generatory pseudolosowe, można wykorzystać do rozwiązania tych wad. W praktyce szyfry strumieniowe służą do szyfrowania na dwa sposoby: w trybie zsynchronizowanym i w trybie niezsynchronizowanym.

Tryb zsynchronizowany. W tym stanowym schemacie szyfrowania nadawca i odbiorca muszą być zsynchronizowani w tym sensie, że wiedzą, ile zwykłego tekstu zostało dotychczas zaszyfrowanych (lub odszyfrowanych). Tryb zsynchronizowany jest zwykle używany w pojedynczej sesji komunikacyjnej pomiędzy stronami (patrz sekcja 4.5.3), gdzie stanowość jest akceptowalna, a wiadomości są odbierane w kolejności i bez utraty. Intuicja jest taka, że generowany jest długi strumień pseudolosowy, którego inna część jest wykorzystywana do szyfrowania każdej wiadomości. Synchronizacja jest konieczna, aby zapewnić prawidłowe odszyfrowanie (tzn. aby odbiorca wiedział, która część strumienia została użyta do zaszyfrowania następnej wiadomości) oraz aby mieć pewność, że żadna część strumienia nie zostanie ponownie wykorzystana. W dalszej części szczegółowo opiszymy ten tryb.



RYSUNEK 3.4: Tryb zsynchronizowany i tryb niezsynchronizowany.

W Algorytmie 3.16 widzieliśmy, że szyfr strumieniowy może zostać użyty do skonstruowania generatora pseudolosowego G z dowolnym pożdanym współczynnikiem rozszerzania. Możemy łatwo zmodyfikować ten algorytm, aby otrzymać generator pseudolosowy G o zmiennej długości wyjściowej. G przyjmuje dwa dane wejściowe: ziarno s i żądaną długość wyjściową l (określamy to w jednostkach jednostkowych, ponieważ G będzie działać w wielomianie czasu w). Podobnie jak w algorytmie 3.16, $G(s, l)$ uruchamia $\text{Init}(s)$, a następnie wielokrotnie uruchamia GetBits w sumie l razy.

Możemy użyć G w konstrukcji 3.17 do obsługi szyfrowania wiadomości o dowolnej długości: szyfrowanie wiadomości m przy użyciu klucza k odbywa się poprzez obliczenie tekstu zaszyfrowanego $c := G(k, 1 |m|) \oplus m$; deszyfrowanie tekstu zaszyfrowanego c za pomocą klucza k odbywa się poprzez obliczenie komunikatu $m := G(k, 1 |c|) \oplus c$. Niewielka modyfikacja dowodu Twierdzenia 3.18 pokazuje, że jeśli szyfr strumieniowy jest bezpieczny, to ten schemat szyfrowania ma szyfrowanie nie do odróżnienia w obecności osoby podsłuchującej.

Krótkie przemyślenie pokazuje, że jeśli komunikujące się strony chcą utrzymać stan, mogą użyć tego samego klucza do szyfrowania wielu wiadomości.

(Patrz rysunek 3.4.) Konceptualny pogląd jest taki, że strony mogą traktować wiele komunikatów m_1, m_2, \dots jako pojedynczą, długą wiadomość; ponadto Konstrukcja 3.17 (jak również wersja zmodyfikowana w poprzednim akapicie) ma tę właściwość, że początkowe części wiadomości mogą być szyfrowane i przesyłane, nawet jeśli reszta wiadomości nie jest jeszcze znana. Konkretnie strony mają wspólny klucz k i obie zaczynają od obliczenia $st0 := \text{Init}(k)$. Aby zaszyfrować pierwszą wiadomość m_1 o długości 1, nadawca wielokrotnie uruchamia GetBits w sumie 1 raz, zaczynając od $st0$, aby otrzymać strumień bitów $pad1 = y_1, \dots, y_1$ wraz ze zaktualizowanym stanem $st1$; nastepnie wysyła $c_1 := pad1 \oplus m_1$. Po otrzymaniu c_1 druga strona wielokrotnie uruchamia GetBits , aby uzyskać te same wartości $pad1$ i $st1$; używa $pad1$ do odzyskania $m_1 := pad1 \oplus c_1$. Później, aby zaszyfrować drugą wiadomość m_2 o długości 2, nadawca wielokrotnie uruchamia funkcję GetBit_s , zaczynając od $st1$, aby uzyskać $pad2 = y_{1+1}, \dots, y_{1+2}$ i stan zaktualizowany, a następnie oblicza tekst zaszyfrowany $c_2 := pad2 \oplus m_2$ i tak dalej. To $st1+2$ może trwać w nieskończoność, umożliwiając stronom wysyłanie nieograniczonej liczby wiadomości o dowolnej długości. Zauważmy, że w tym trybie szyfr strumieniowy nie musi używać IV.

Ta metoda szyfrowania wielu wiadomości wymaga od komunikujących się stron utrzymywania stanu synchronizacji, co wyjaśnia terminologię „tryb synchronizacji”. Z tego powodu metoda ta jest odpowiednia, gdy dwie strony komunikują się w ramach jednej „sesji”, ale nie sprawdza się w przypadku sporadycznej komunikacji lub gdy strona może z czasem komunikować się z różnych urządzeń. (Stosunkowo łatwo jest przechowywać kopie stałego klucza w różnych lokalizacjach; trudniej jest utrzymać zsynchronizowany stan w wielu lokalizacjach.) Ponadto, jeśli strony kiedykolwiek stracą synchronizację (np. z powodu jednej z transmisji pomyłki strony zostaną pominięte), odszyfrowanie zwróci nieprawidłowy wynik. Ponowna synchronizacja jest możliwa, ale powoduje dodatkowe obciążenie.

Tryb niezsynchronizowany. W przypadku szyfrów strumieniowych, których funkcja Init przyjmuje jako dane wejściowe wektor inicjujący, możemy osiągnąć bezstanowe szyfrowanie bezpieczne CPA dla wiadomości o dowolnej długości. Tutaj modyfikujemy G , aby akceptować trzy dane wejściowe: ziarno s , wektor inicjujący IV i żądaną długość wyjściową l . Teraz ten algorytm najpierw oblicza $\text{st}0 := \text{Init}(s, IV)$, a następnie wielokrotnie uruchamia GetBits . Szyfrowanie można wówczas przeprowadzić stosując wariant konstrukcji 3.30: szyfrowanie wiadomości m przy użyciu klucza k odbywa się poprzez wybranie jednolitego wektora inicjującego IV $\{0, 1\}^n$ i obliczenie tekstu zaszyfrowanego $IV, G(s, IV, 1 |m|) \oplus m$; deszyfrowanie odbywa się w sposób naturalny. (Patrz rysunek 3.4.) Ten schemat jest bezpieczny CPA, jeśli szyfr strumieniowy ma teraz silniejszą właściwość niż dla dowolnego wielomianu funkcja F zdefiniowana przez $\text{Fk}(IV) \stackrel{\text{def}}{=} G(k, IV, 1)$ jest funkcją pseudolosową. (W rzeczywistości F musi być pseudolosowe tylko wtedy, gdy jest oceniane na jednolitych danych wejściowych. Funkcje z kluczem o tej słabszej właściwości nazywane są słabymi funkcjami pseudolosowymi.)

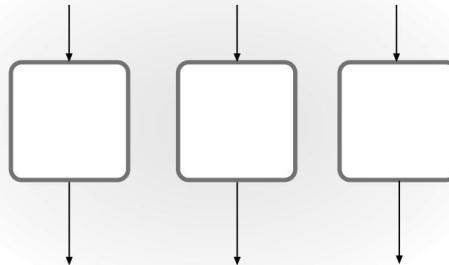
3.6.2 Tryby działania szyfru blokowego

Widzieliśmy już konstrukcję schematu szyfrowania bezpiecznego CPA w oparciu o funkcje pseudolosowe/szyfry blokowe. Jednak Konstrukcja 3.30 (i jej rozszerzenie na komunikaty o dowolnej długości, jak omówiono na końcu sekcji 3.4.2) ma tę wadę, że długość tekstu zaszyfrowanego jest dwukrotnie większa od długości tekstu jawnego. Tryby działania szyfru blokowego umożliwiają szyfrowanie wiadomości o dowolnej długości przy użyciu krótszych tekstów zaszyfrowanych.

W tej sekcji niech F będzie szyfrem blokowym o długości bloku n . Zakładamy tutaj, że wszystkie wiadomości m , które są szyfrowane, mają długość będącą wielokrotnością n , zapisuje się $m = m_1, m_2, \dots, m_n$, reprezentującą kolejne fragmenty wiadomości, które nie są wielokrotnością n , można zawsze jednoznacznie dopełnić do długości będącej wielokrotnością n , dodając jedynkę, po której następuje wystarczająca liczba zer, więcej o tym założeniu nie powoduje większej utraty ogólności.

Znanych jest kilka trybów działania szyfru blokowego; prezentujemy cztery z najczęściej stosowanej i omów ich bezpieczeństwo.

Tryb Elektronicznej Książki Kodowej (ECB). Jest to naiwny sposób działania, w którym zaszyfrowany tekst uzyskuje się poprzez bezpośrednie zastosowanie szyfru blokowego do każdego bloku tekstu jawnego. Oznacza to, że $c := Fk(m_1), Fk(m_2), \dots, Fk(m)$; Widzieć



RYSUNEK 3.5: Tryb Elektronicznej Książki Kodowej (ECB).

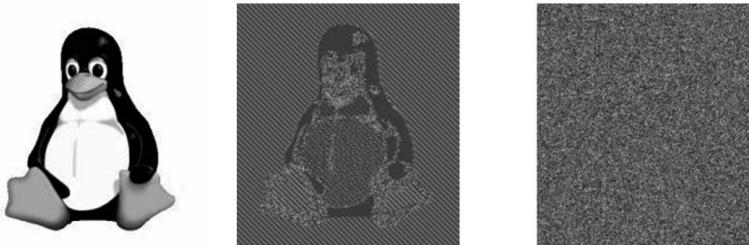
Rysunek 3.5. Deszyfrowanie odbywa się w oczywisty sposób, wykorzystując fakt, że F jest efektywnie obliczalne.

k^{-1} jest

Tryb EBC jest deterministyczny i dlatego nie może być chroniony przez CPA. Co gorsza, szyfrowanie w trybie EBC nie zapewnia szyfrowania nie do odróżnienia nawet w obecności osoby podsłuchującej. Dzieje się tak dlatego, że jeśli blok zostanie powtórzony w tekście jawnym, spowoduje to powtórzenie bloku w tekście zaszyfrowanym. Dlatego łatwo jest odróżnić szyfrowanie tekstu jawnego składającego się z dwóch identycznych bloków od szyfrowania tekstu jawnego składającego się z dwóch różnych bloków. To nie jest tylko problem teoretyczny. Rozważ zaszyfrowanie obrazu, w którym małe grupy pikseli odpowiadają blokowi zwykłego tekstu. Szyfrowanie w trybie EBC może ujawnić znaczną ilość informacji o wzorach na obrazie, co nie powinno mieć miejsca w przypadku korzystania z bezpiecznego schematu szyfrowania.

Pokazuje to rysunek 3.6.

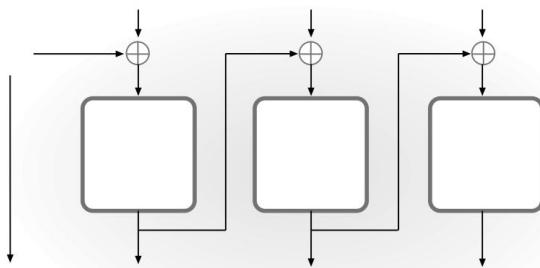
Z tych powodów nie należy nigdy używać trybu EBC. (Uwzglę dniamy tylko to ze względem na swoje historyczne znaczenie.)



RYSUNEK 3.6: Ilustracja niebezpieczeństw związanych z używaniem trybu EBC. Środkowa figura przedstawia szyfrowanie obrazu po lewej stronie przy użyciu trybu EBC; rysunek po prawej stronie przedstawia szyfrowanie tego samego obrazu przy użyciu trybu bezpiecznego.

(Zaczerpnię to z <http://en.wikipedia.org> i zaczerpnię to z obrazów stworzonych przez Larry'ego

Ewinga (lewing@isc.tamu.edu) przy użyciu programu GIMP.)



RYSUNEK 3.7: Tryb łączenia bloków szyfru (CBC).

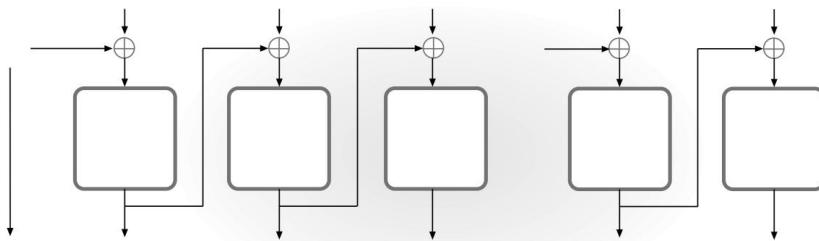
Tryb łączenia bloków szyfrów (CBC). Aby zaszyfrować w tym trybie, najpierw wybierany jest jednolity wektor inicjujący (IV) o długości n . Następnie generowane są bloki tekstu zaszyfrowanego poprzez zastosowanie szyfru blokowego do XOR bieżącego bloku tekstu jawnego i poprzedniego bloku tekstu zaszyfrowanego. Oznacza to, że ustaw $c_0 := \text{IV}$, a następnie dla $i = 1$ ustawić $c_i := F_k(c_{i-1} \oplus m_i)$. Ostateczny tekst zaszyfrowany to c . (Patrz c_0, c_1, \dots , rysunek 3.7.) Odszyfrowanie tekstu zaszyfrowanego c_0, \dots, c_n jest gotowe obliczając $m_i := F_k^{-1}(c_i \oplus c_{i-1})$ dla $i = 1, \dots, n$. Podkreślamy, że IV jest w zaszyfrowanym tekście; jest to kluczowe, aby można było przeprowadzić deszyfrowanie.

Co ważne, szyfrowanie w trybie CBC jest probabilistyczne i udowodniono, że jeśli F jest permutacją pseudolosową, to szyfrowanie w trybie CBC jest zabezpieczone CPA. Główną wadą tego trybu jest to, że szyfrowanie musi być przeprowadzane sekwencyjnie, ponieważ do zaszyfrowania bloku tekstu jawnego m_i potrzebny jest blok tekstu zaszyfrowanego c_{i-1} . Zatem, jeśli dostęp pne jest przetwarzanie równoległe, szyfrowanie w trybie CBC może nie być najskuteczniejszym wyborem.

Mogna pokusić się o myślenie, że wystarczy użyć odrębnego IV (a nie losowego IV) dla każdego szyfrowania, np. najpierw użyć $\text{IV} = 1$, a następnie zwiększyć IV o jeden za każdym razem, gdy wiadomość jest szyfrowana. W ćwiczeniu 3.20 prosimy o pokazanie, że ten wariant szyfrowania w trybie CBC nie jest bezpieczny.

Mogna również rozważyć stanowy wariant szyfrowania w trybie CBC – zwany trybem łańcuchowym CBC – w którym ostatni blok poprzedniego tekstu zaszyfrowanego jest używany jako IV podczas szyfrowania następnej wiadomości. Zmniejsza to przepustowość, ponieważ IV nie musi być wysyłany za każdym razem. Zobacz rysunek 3.8, gdzie początkowy komunikat m_1, m_2, m_3 jest szyfrowany przy użyciu losowego IV, a następnie drugi komunikat m_4, m_5 jest szyfrowany przy użyciu c_3 jako IV. (W przeciwieństwie do tego, szyfrowanie przy użyciu bezstanowego trybu CBC wygenerowałoby nowy IV podczas szyfrowania drugiej wiadomości). Tryb łańcuchowego CBC jest używany w SSL 3.0 i TLS 1.0.

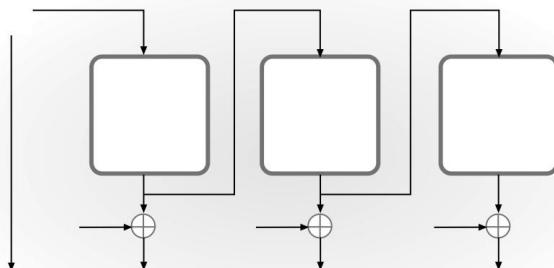
Mogie się wydawać, że łańcuchowy tryb CBC jest tak samo bezpieczny jak tryb CBC, ponieważ szyfrowanie łańcuchowe CBC m_1, m_2, m_3 , po którym następuje szyfrowanie m_4, m_5 , daje te same bloki tekstu zaszyfrowanego, co szyfrowanie (pojedynczej) wiadomości w trybie CBC szalwią m_1, m_2, m_3, m_4, m_5 . Niemniej jednak tryb łańcuchowy CBC jest podatny na atak z wybranym tekstem jawnym. Podstawą ataku jest wiedza przeciwnika



RYSUNEK 3.8: Przykutki CBC.

przesuń „wektor inicjujący”, który będzie używany w drugiej zaszyfrowanej wiadomości. Atak opisujemy nieformalnie, na podstawie rysunku 3.8. Założmy, że atakujący wie, że $m_1 = \{m_0 m_1\}$ i obserwuje pierwszy szyfrogram IV, c_1, c_2, c_3 . Następnie atakujący żąda zaszyfrowania drugiej wiadomości m_4, m_5 za pomocą $m_4 = IV - m_0$ i obserwuje drugi zaszyfrowany tekst c_4, c_5 . Można sprawdzić, że $m_1 = m_0$ wtedy i tylko wtedy, gdy $c_4 = c_1$. Przykład ten powinien służyć jako mocne ostrzeżenie przed dokonywaniem jakichkolwiek modyfikacji schematów kryptograficznych, nawet jeśli modyfikacje te wydają się niegroźne.

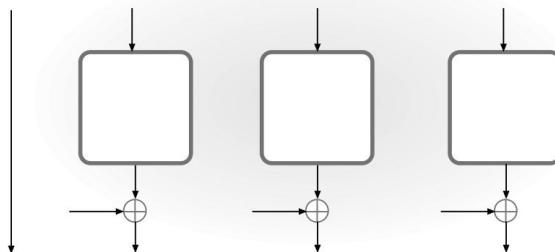
Tryb sprzężenia zwrotnego wyjścia (OFB). Trzeci tryb, który prezentujemy, można postrzegać jako niezsynchronizowany tryb szyfru strumieniowego, w którym szyfr strumieniowy jest konstruowany w określony sposób na podstawie podstawowego szyfru blokowego. Opisujemy ten tryb bezpośrednio. Najpierw wybierany jest jednorodny $IV = \{0, 1\}^n$. Następnie generowany jest strumień pseudolosowy w następujący sposób: Zdefiniuj $y_0 := IV$ i ustaw i -ty blok strumienia na $y_i := F_k(y_{i-1})$. Każdy blok tekstu jawnego jest szyfrowany poprzez XORowanie go z odpowiednim blokiem strumienia; to znaczy $c_i := y_i \oplus m_i$. (Patrz rysunek 3.9.) Podobnie jak w trybie CBC, IV jest częścią tekstu zaszyfrowanego, aby umożliwić deszyfrowanie. Jednak w przeciwieństwie do trybu CBC, tutaj nie jest wymagane, aby F było odwracalne. (Właściwie to potrzebne



RYSUNEK 3.9: Tryb sprzężenia zwrotnego wyjścia (OFB).

nie może być nawet permutacją.) Ponadto, podobnie jak w trybach działania szyfru strumieniowego, tutaj nie jest konieczne, aby długość tekstu jawnego była wielokrotnością długości bloku. Zamiast tego wygenerowany strumień można obciąć dokładnie do długości zwykłego tekstu. Kolejną zaletą trybu OFB jest to, że jego wariant stanowy (w którym ostateczna wartość y użycia do zaszyfrowania jakiejś wiadomości jest używana jako IV do zaszyfrowania następującej wiadomości) jest bezpieczna. Ten wariant stanowy jest równoważny z synchronizowanemu trybowi szyfru strumieniowego, w którym szyfr strumieniowy jest skonstruowany z szyfru blokowego w właśnie opisany sposób.

Można wykazać, że tryb OFB jest bezpieczny pod względem CPA, jeśli F jest funkcją pseudolosową. Chociaż zarówno szyfrowanie, jak i deszyfrowanie muszą być przeprowadzane sekwencyjnie, tryb ten ma tę zaletę w porównaniu z trybem CBC, że większość obliczeń (mianowicie obliczenia strumienia pseudolosowego) można wykonać niezależnie od faktycznej wiadomości, która ma być zaszyfrowana. Możliwe jest zatem wygenerowanie strumienia pseudolosowego z wyprzedzeniem przy użyciu przetwarzania wstępne, po czym szyfrowanie zwykłego tekstu (gdy jest on znany) jest niewiarygodnie szybkie.



RYSUNEK 3.10: Tryb licznika (CTR).

Tryb licznika (CTR). Tryb licznika można również postrzegać jako niezsynchonizowany tryb szyfru strumieniowego, w którym szyfr strumieniowy jest konstruowany z szyfru blokowego, jak w konstrukcji 3.29. Podajemy tutaj samodzielny opis.

Aby zaszyfrować w trybie CTR, najpierw wybierana jest jednolita wartość $ctr \in \{0, 1\}^n$.

Następnie generowany jest strumień pseudolosowy poprzez obliczenie $y_i := F_k(ctr + i)$, gdzie $ctr + i$ są postrzegane jako liczby całkowite, a dodawanie odbywa się modulo 2^n . I-tym blokiem tekstu zaszyfrowanego to $c_i := y_i \oplus m_i$, a IV jest ponownie wysyłany jako część tekstu zaszyfrowanego; patrz rysunek 3.10.

Zauważ jeszcze raz, że deszyfrowanie nie wymaga odwracalności F ani nawet permutacji. Podobnie jak w przypadku trybu OFB, innego trybu „szycfrowania strumieniowego”, wygenerowany strumień można obciąć dokładnie do długości tekstu jawnego, można zastosować przetwarzanie wstępne w celu wygenerowania strumienia pseudolosowego, zanim wiadomość będzie znana, a stanowy wariant trybu CTR jest bezpieczny.

W przeciwieństwie do wszystkich omówionych wcześniej trybów bezpiecznych, tryb CTR ma tę zaletę, że szyfrowanie i deszyfrowanie mogą być w pełni zrównoległe, ponieważ wszystkie bloki strumienia pseudolosowego można obliczać niezależnie od siebie. W przeciwieństwie do OFB możliwe jest również odszyfrowanie i-tego bloku

szystko program wykorzystujący tylko jedną ocenę F. Te cechy sprawiają, że tryb CTR jest atrakcyjnym wyborem.

Tryb CTR jest również dość prosty do analizy:

TWIERDZENIE 3.32 Jeśli F jest funkcją pseudolosową, to tryb CTR jest zabezpieczony CPA.

DOWÓD Postępujemy według tego samego szablonu, co w dowodzie Twierdzenia 3.31: najpierw zastępujemy F funkcją losową, a następnie analizujemy otrzymany schemat.

Niech $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ będzie (bezstanowym) schematem szyfrowania w trybie CTR, i niech $\Pi' = (\text{Gen}', \text{Enc}', \text{Dec}')$ będzie schematem szyfrowania identycznym z Π różnicą, że zamiast F_k zostanie użyta prawdziwie losowa funkcja. Oznacza to, że $\text{Gen}'(1^n)$ wybiera funkcję jednolitą f' Funcn, a Enc' szyfruje tak samo jak Enc , z tą różnicą, że zamiast F_k zostanie użyte f' . (Po raz kolejny ani Gen' , ani Enc' nie są efektywne, ale nie ma to znaczenia dla celów zdefiniowania eksperymentu z udziałem Π).

Napraw dowolnego przeciwnika ppt A i niech $q(n)$ będzie wielomianem górnej granicy liczby zapytań szyfrujących-oracle wykonanych przez $A(1^n)$, a także maksymalną liczbą bloków w każdym takim zapytaniu i maksymalną liczbą bloków w m_0 i m_1 . W pierwszym kroku dowodu założymy, że istnieje funkcja pomijalna taka, że

$$\Pr_{\substack{\text{A}, \Pi}}[\text{PrivKcpa}_{\text{A}, \Pi}(n) = 1] = 1 - \Pr_{\substack{\text{A}, \Pi}}[\text{PrivKcpa}_{\text{A}, \Pi}(n) = 0] = \text{negl}(n). \quad (3.12)$$

Dowodzi się tego poprzez redukcję w sposób podobny do analogicznego kroku w dowodzie Twierdzenia 3.31 i pozostawia się go jako ćwiczenie dla czytelnika.

Następnie to twierdzimy

$$\Pr_{\substack{\text{A}, \Pi}}[\text{PrivKcpa}_{\text{A}, \Pi}(n) = 1] < \frac{1}{n} \cdot \frac{2q(n) + 2^2}{2^2}. \quad (3.13)$$

W połączeniu z równaniem (3.12) oznacza to, że

$$\Pr_{\substack{\text{A}, \Pi}}[\text{PrivKcpa}_{\text{A}, \Pi}(n) = 0] < \frac{1}{n} + \frac{2q(n)}{2^2} + \text{zaniechanie}(n).$$

Ponieważ q jest wielomianem, jest ono pomijalne i to kończy dowód.

Udowodnimy teraz równanie (3.13). Napraw wartość n dla parametru bezpieczeństwa. w $q(n)$ oznacza długość (w blokach) komunikatów m_0, m_1 wprowadzanych przez Let A eksperymentu $\text{PrivKcpa}(n)$ i niech ctr oznacza wartość początkową używaną podczas generowania przez A, Π tekstu zaszyfrowanego wyzwania. Podobnie, niech i $q(n)$ będzie długością (w blokach) i-tego zapytania szyfrującego-oracle wykonanego przez A i niech ctr_i oznacza wartość początkową używaną podczas odpowiadania na to zapytanie. Kiedy zostanie udzielona odpowiedź na i-te zapytanie szyfrujące-oracle, f jest stosowane do wartości $\text{ctr}_i + 1, \dots, \text{ctr}_i + i$. Kiedy zaszyfrowany tekst wyzwania jest szyfrowany, f jest stosowane do $\text{ctr}_i + 1, \dots, \text{ctr}_i + i$ -ty bloku tekstu zaszyfrowanego jest obliczany przez XORing $f(\text{ctr}_i + i)$ z i-tym blokiem wiadomości. Istnieją dwa przypadki:

1. Nie istnieją żadne $i, j, j = 1$ (przy $j \neq i$), dla których $\text{ctr}_i + j = \text{ctr}_j + j$ użyte podczas szyfrowania tekstu ~~względem przypadku wywołania~~ (stąd równomiernie rozłożone i niezależne od reszty eksperymentu, ponieważ f nie zostało zastosowane do żadnego z tych danych wejściowych podczas szyfrowania zapytań wyroczni przeciwnika. Oznacza to, że zaszyfrowany tekst wyzwania jest obliczany poprzez XORowanie strumienia jednolitych bitów z komunikatem mb, a więc prawdopodobieństwo, że A wyprowadza $b = b$ wynosi dokładnie $1/2$ (jak w przypadku jednorazowej podkładki).

2. Istnieje $i, j, j = 1$ (przy $j \neq i$) dla którego $\text{ctr}_i + j = i + j$:
Oznaczamy to zdarzenie przez nakładanie się . W tym przypadku A może potężnie ostatecznie określić, która z jego wiadomości została zaszyfrowana, aby dać zaszyfrowany tekst stanowiący wyzwanie, ponieważ A może wydedukować wartość $f(\text{ctr}_i + j) = f(\text{ctr}_j + j)$ od odpowiedzi na i-te zapytanie Oracle.

Przeanalizujmy prawdopodobieństwo wystąpienia nakładania się . Prawdopodobieństwo $= q(n)$ dla miedzy jeśli i każde i są tak duże, jak to możliwe, więc $c_i = c_j$ wynosi maksimum. Niech Overlap oznacza zdarzenie, że sekwencja $\text{ctr}_i + 1, \dots, \text{ctr}_i + n$ zakończy się na $\text{ctr}_j + 1, \dots, \text{ctr}_j + n$. Nakładanie się jest zdarzeniem, które nakłada się na sekwencję $\text{ctr}_i + 1, \dots, \text{ctr}_i + n$. Nakładanie się jest zdarzeniem, które nakłada się na sekwencję $\text{ctr}_j + 1, \dots, \text{ctr}_j + n$. Ponieważ istnieje co najwyżej $q(n)$ zapytań Oracle, połączenie powiązane (por. Twierdzenie A.7) daje

$$\Pr[\text{Nakładanie się}] = \Pr_{j=1}^n [\text{Nakładanie się}] = \Pr_{j=1}^n [\text{Overlap}] \quad (3.14)$$

Naprawianie ctr_i , zdarzenie Overlap ma miejsce dokładnie wtedy, gdy ctr_i jest spełnione

$$\text{ctr}_i = q(n) + 1 \quad \text{ctr}_i = \text{ctr}_j + q(n) - 1.$$

Ponieważ istnieje $2q(n) - 1$ wartości ctr_i , dla których zachodzi Overlap, a ctr_i jest wybierane równomiernie z $\{0, 1\}^n$, widzimy, że

$$\Pr[\text{Nakładanie się}] = \frac{2q(n) - 1}{2^n} < \frac{2q(n)}{2^n}.$$

W połączeniu z równaniem (3.14) daje to $\Pr[\text{Overlap}] < 2q(n)/2^n$.

Biorąc pod uwagę powyższe, możemy łatwo powiązać prawdopodobieństwo sukcesu A:

$$\begin{aligned} \Pr[\text{PrivKcpa}_{A, \Pi}(n) = 1] &= \Pr[\text{PrivKcpa}_{A, \Pi}(n) = 1 \mid \overline{\text{Nakładanie się}}] \\ &\quad + \Pr[\text{PrivKcpa}_{A, \Pi}(n) = 1 \mid \overline{\text{Nakładanie się}}] \\ &= \Pr[\text{PrivKcpa}_{A, \Pi}(n) = 1 \mid \overline{\text{Nakładanie się}}] + \Pr[\text{Nakładanie się}] \\ &< \frac{1}{2^n} + \frac{2q(n)^2}{2^n}. \end{aligned}$$

To dowodzi równania (3.13) i kończy dowód. ■

Tryby działania i manipulowanie komunikatami. W wielu teksth sposoby działania porównuje się również pod kątem tego, jak dobrze chronią przed wrogą modyfikacją tekstu zaszyfrowanego. Nie zamieszczamy tutaj takiego porównania, gdyż kwestię integralności wiadomości czy uwierzytelniania wiadomości należy rozpatrywać odrę bnie od szyfrowania, co uczynimy w następym rozdziale.

Żaden z powyższych trybów nie zapewnia integralności komunikatu w sensie, który tam zdefiniujemy. Dalszą dyskusję odkładamy do następnego rozdziału.

Jeśli chodzi o zachowanie różnych modów w obecności „łagodnych” (tj. niekontradykcyjnych) błę dów transmisji, patrz Ćwiczenia 3.21 i 3.22. Zauważmy jednak, że ogólnie rzecz biorąc, takie błądy mogą wyeliminować za pomocą standardowych technik (np. korekcji błędów lub retransmisji).

Długość bloku i bezpieczeństwo betonu. Wszystkie tryby CBC, OFB i CTR korzystają z losowego IV. Powoduje to randomizację procesu szyfrowania i zapewnia, że (z dużym prawdopodobieństwem) podstawowy szyfr blokowy jest zawsze oceniany na świeżych (tj. nowych) danych wejściowych. Jest to ważne, ponieważ, jak widzieliśmy w dowodach Twierdzenia 3.31 i Twierdzenia 3.32, jeśli dane wejściowe szyfru blokowego zostaną użyte więcej niż raz, może to spowodować naruszenie bezpieczeństwa.

Długość bloku szyfru blokowego ma zatem znaczący wpływ na konkretne bezpieczeństwo schematów szyfrowania opartych na tym szyfrze. Rozważmy np. tryb CTR wykorzystujący szyfr blokowy F o długości bloku . IV jest wówczas jednolitym ciągiem bitowym i spodziewamy się, że IV powtórzy się po zaszyfrowaniu około 2/2 wiadomości (patrz Dodatek A.4). Jeśli jest za krótki, to nawet jeśli F jest bezpieczne jako permutacja pseudolosowa, wynikająca z tego konkretna granica bezpieczeństwa będzie zbyt słaba do praktycznych zastosowań. Konkretnie, jeśli $= 64$, jak ma to miejsce w przypadku DES (szyfr blokowy, który przeanalizujemy w rozdziale 6), to po $2^{32} = 4\ 300\ 000\ 000$ szyfrowań —czyli około 34 gigabajtach zwykłego tekstu —oczekuje się, że nastąpi powtarzające się IV. Choć może się to wydawać dużą ilością danych, jest ona mniejsza niż pojemność nowoczesnych dysków twardych.

Niewłaściwe użycie IV. W naszym opisie i omówieniu różnych (bezpiecznych) trybów założyliśmy, że za każdym razem, gdy wiadomość jest szyfrowana, wybierany jest losowy IV. Co się stanie, jeśli to założenie się nie powiedzie z powodu np. słabego generowania losowości lub błędu dnej implementacji? Z pewnością nie możemy już zagwarantować bezpieczeństwa w rozumieniu Definicji 3.22. Jednak z praktycznego punktu widzenia „tryby szyfrowania strumieniowego” (OFB i CTR) są znacznie gorsze niż tryb CBC. Jeśli IV powtórzy się podczas korzystania z pierwszego, atakujący może XORować dwa powstałe zaszyfrowane teksty i dowiedzieć się wielu informacji o całej zawartości obu zaszyfrowanych wiadomości (jak widzieliśmy wcześniej w kontekście jednorazowej podkładki, jeśli klucz jest ponownie użyty). Jednak w trybie CBC jest prawdopodobne, że już po kilku blokach dane wejściowe szyfru blokowego „rozędzą się” i osoba atakująca nie będzie w stanie poznać żadnych informacji poza kilkoma pierwszymi blokami wiadomości.

Jednym ze sposobów rozwiązania problemu potencjalnego niewłaściwego użycia IV jest użycie szyfrowania stanowego, jak omówiono w kontekście trybów OFB i CTR. Jeśli szyfrowanie stanowe nie jest możliwe i istnieje obawa, że możliwe będzie niewłaściwe użycie dożywione, zalecany jest tryb CBC z powodów opisanych powyżej.

3.7 Ataki za pomocą wybranego tekstu

zaszyfrowanego 3.7.1 Definiowanie bezpieczeństwa CCA

Do tej pory zdefiniowaliśmy zabezpieczenia przed dwoma rodzajami ataków: pasywnym podsłuchiwaniem i atakami z wybranym tekstem jawnym. Atak wybranym szyfrogramem jest jeszcze potę żniejszy. W ataku z wybranym tekstem zaszyfrowanym przeciwnik ma możliwość nie tylko uzyskania szyfrowania wybranych przez siebie wiadomości (jak w przypadku ataku z wybranym tekstem jawnym), ale także uzyskania odszyfrowania wybranego tekstu zaszyfrowanego (z jednym wyjątkiem omówionym później). Formalnie, oprócz wyroczni szyfrującej, dajemy przeciwnikowi dostęp p do wyroczni deszyfrującej. Przedstawiamy formalną definicję i odraczamy dalszą dyskusję.

Rozważ następujący eksperyment dla dowolnego schematu szyfrowania klucza prywatnego $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, przeciwnik A i wartość n dla parametru bezpieczeństwa.

Eksperyment nieroróżnialności CCA $\text{PrivKcca } A, \Pi(n)$:

1. Klucz k jest generowany poprzez uruchomienie $\text{Gen}(1n)$.
2. Przeciwnik A otrzymuje dane wejściowe 1 n i dostęp p do Oracle do $\text{Enck}(\cdot)$ i $\text{Deck}(\cdot)$.
Wysyła parę komunikatów m_0, m_1 o tej samej długości.
3. Wybierany jest jednolity bit b $\in \{0, 1\}$, a następnie obliczany jest zaszyfrowany tekst c $= \text{Enck}(mb)$ i podawany A. Nazywamy c szyfrogramem prowokacyjnym.
4. Przeciwnik A w dalszym ciągu ma dostęp p do wyroczni do $\text{Enck}(\cdot)$ i $\text{Deck}(\cdot)$, ale nie może odpytywać tego ostatniego o sam zaszyfrowany tekst wyzwania.
Ostatecznie A wyprowadza trochę b.
5. Wynik eksperymentu definiuje się jako 1, jeśli $b = b$, i 0 w przeciwnym razie. Jeśli wynik eksperymentu wynosi 1, mówimy, że A się udało.

DEFINICJA 3.33 Schemat szyfrowania kluczem prywatnym Π ma szyfrowanie nieroróżnialne w przypadku ataku wybranym tekstem zaszyfrowanym lub jest bezpieczny CCA, jeśli dla wszystkich probabilistycznych przeciwników A w czasie wielomianowym istnieje pomijalna funkcja zaniedbania taka, że:

$$\Pr[\text{PrivKcca } A, \Pi(n) = 1] \stackrel{1}{\longrightarrow} \text{zaniedbywanie}(n), 2$$

gdzie prawdopodobieństwo uwzględnia całą losowość zastosowaną w eksperymencie.

Dla kompletności zauważamy, że naturalna analogia Twierdzenia 3.24 dotyczy również bezpieczeństwa CCA. (Mianowicie, jeśli schemat ma nieroróżnialne szyfrowania w przypadku ataku z wybranym tekstem zaszyfrowanym, wówczas ma on nieroróżnialne wielokrotne szyfrowania w przypadku ataku z wybranym tekstem zaszyfrowanym, odpowiednio zdefiniowanym.)

W powyższym eksperymencie dostęp przeciwnika do wyroczni deszyfrującej jest nieograniczony, z wyjątkiem ograniczenia, zgodnie z którym przeciwnik nie może żądać odszyfrowania samego zaszyfrowanego tekstu wyzwania. To ograniczenie jest konieczne, w przeciwnym razie nie ma nadziei, aby jakkolwiek schemat szyfrowania spełniał definicję .

W tym momencie możesz się zastanawiać, czy ataki z użyciem wybranego tekstu zaszyfrowanego realistycznie modelują każdy atak w świecie rzeczywistym. Podobnie jak w przypadku ataków z użyciem wybranego tekstu jawnego, nie oczekujemy, że uczciwe strony odszyfrują dowolne szyfrogramy wybrane przez przeciwnika. Niemniej jednak mogą zaistnieć scenariusze, w których przeciwnik może mieć wpływ na to, co zostanie odszyfrowane i poznąć częściowe informacje na temat wyniku. Na przykład:

1. W przykładzie Midway z sekcji 3.4.2 można sobie wyobrazić, że amerykańscy kryptoanalitycy mogli również próbować wysyłać zaszyfrowane wiadomości do Japończyków, a następnie monitorować ich zachowanie. Takie zachowanie (np. ruch sił i tym podobne) mogło dostarczyć ważnych informacji na temat tekstu jawnego.

2. Wyobraź sobie użytkownika wysyłającego zaszyfrowane wiadomości do swojego banku. Przeciwnik może być w stanie wysłać szyfrogram w imieniu tego użytkownika; bank odszyfruje te zaszyfrowane teksty, a przeciwnik może dowiedzieć się czegoś o wyniku. Na przykład, jeśli szyfrogram odpowiada źle sformułowanemu tekstowi jawnemu (np. niezrozumiałej wiadomości lub po prostu wiadomości, która jest nieprawidłowo sformatowana), przeciwnik może wywnioskować to z reakcji banku (tj. schematu późniejszej komunikacji). Praktyczny atak tego typu przedstawiono w sekcji 3.7.2 poniżej.

3. Szyfrowanie jest częścią stosowane w protokołach wyższego poziomu; np. schemat szyfrowania może być używany jako część protokołu uwierzytelniania, w którym jedna strona wysyła zaszyfrowany tekst do drugiej, która go odszyfrowuje i zwraca wynik. W tym przypadku jedna z uczciwych stron zachowuje się dokładnie jak wyrocznia deszyfrująca.

Niepewność systemów, które badaliśmy. Żaden ze schematów szyfrowania, które widzieliśmy do tej pory, nie jest zabezpieczony CCA. Pokazujemy to dla Construction 3.30, gdzie szyfrowanie jest obliczane jako

$$\text{Enck}(m) = r, \quad Fk(r) = m.$$

Rozważmy przeciwnika A biorącego udział w eksperymencie nieroróżnialności CCA, który wybiera $m_0 = 0n$ i $m_1 = 1n$. Następnie, po otrzymaniu zaszyfrowanego tekstu $c = r, s$, przeciwnik może odwrócić pierwszy bit s i poprosić o odszyfrowanie powstałego zaszyfrowanego tekstu c . Ponieważ $c = c$, to zapytanie jest dozwolone, a wyrocznia deszyfrująca odpowiada albo $10n-1$ (w tym przypadku jest jasne, że $b = 0$), albo $01n-1$ (w tym przypadku $b = 1$). Ten przykład pokazuje, że zabezpieczenia CCA są dość rygorystyczne. Żaden schemat szyfrowania, który pozwala na „manipulację” zaszyfrowanych tekstów w jakikolwiek kontrolowany sposób, nie może być zabezpieczony za pomocą CCA. Zatem bezpieczeństwo CCA implikuje bardzo ważną właściwość zwaną nieplastycznością. Mówiąc najprościej, nieplastyczny schemat szyfrowania ma tę właściwość, że jeśli przeciwnik spróbuje zmodyfikować dany tekst zaszyfrowany, rezultatem będzie albo nieprawidłowy tekst zaszyfrowany, albo jeden

który odszyfrowuje do postaci zwykłego tekstu niemającego żadnego związku z oryginałem. Jest to bardzo przydatna właściwość w przypadku schematów stosowanych w złożonych protokołach kryptograficznych.

Konstruowanie schematu szyfrowania bezpiecznego dla CCA. W sekcji 4.5.4 pokazujemy, jak skonstruować schemat szyfrowania bezpieczny dla CCA. Konstrukcja została tam przedstawiona, ponieważ wykorzystuje narzędzia, które opracowaliśmy w Rozdziale 4.

3.7.2 Ataki dopełniające-Oracle

Atak wybranym szyfrogramem na Construction 3.30 opisany w poprzedniej sekcji jest nieco wymyślony, ponieważ zakłada, że atakujący może uzyskać pełne odszyfrowanie zmodyfikowanego tekstu zaszyfrowanego. Choć tego rodzaju atak jest dozwolony zgodnie z Definicją 3.33, nie jest jasne, czy w praktyce stanowi on poważny problem. Tutaj pokazujemy atak wybranym szyfrogramem na naturalny i szeroko stosowany schemat szyfrowania; co więcej, atak wymaga jedynie zdolności atakującego do ustalenia, czy zmodyfikowany tekst zaszyfrowany został poprawnie odszyfrowany.

Takie informacje są często bardzo łatwe do uzyskania, ponieważ na przykład serwer może zażądać retransmisji lub zakończyć sesję, jeśli kiedykolwiek odbierze tekst zaszyfrowany, którego nie można poprawnie odszyfrować, a każde z tych zdarzeń spowodowałobyauważalną zmianę w obserwowanym ruchu. Wykazano, że atak działa w praktyce na różnych wdrożonych protokołach; podamy jeden konkretny przykład na końcu tej sekcji.

Jak wspomniano wcześniej, podczas korzystania z trybu CBC długość tekstu jawnego musi być wielokrotnością długości bloku; jeśli tekst jawnego nie spełnia tej właściwości, należy go dopełnić przed zaszyfrowaniem. Oryginalny tekst jawnego nazywamy wiadomością, a wynik po dopełnieniu nazywamy zakodowanymi danymi. Stosowany przez nas schemat dopełniania musi umożliwiać odbiorcy jednoznaczne określenie na podstawie zakodowanych danych, gdzie kończy się wiadomość. Jednym z popularnych i ustandaryzowanych podejść jest użycie dopełnienia PKCS #5. Założymy, że oryginalna wiadomość ma całkowitą liczbę bajtów i niech L oznacza długość bloku (w bajtach) używanego szyfru blokowego. Niech b oznacza liczbę bajtów, które należy dołączyć do wiadomości, aby całkowita długość wynikowych zakodowanych danych była wielokrotnością długości bloku. Tutaj b jest liczbą całkowitą od 1 do L włącznie. (Nie możemy mieć b = 0, ponieważ prowadziłoby to do niejednoznacznego dopełnienia. Zatem, jeśli długość wiadomości jest już wielokrotnością długości bloku, dołączanych jest L bajtów dopełnienia.) Następnie dołączamy do wiadomości ciąg znaków zawierający liczbę całkowitą b (przedstawiony w 1 bajcie lub 2 cyfrach szesnastkowych) powtórzony b razy. Oznacza to, że jeśli potrzebny jest 1 bajt dopełnienia, wówczas dołączany jest ciąg 0x01 (zapisany w formacie szesnastkowym); jeśli potrzebne są 4 bajty dopełnienia, dołączany jest ciąg szesnastkowy 0x04040404; itp. Zakodowane dane są następnie szyfrowane przy użyciu zwykłego szyfrowania w trybie CBC.

Podczas deszyfrowania odbiorca najpierw stosuje deszyfrowanie w trybie CBC w zwykły sposób, aby odzyskać zakodowane dane, a następnie sprawdza, czy zakodowane dane są prawidłowo dopełnione. (Można to łatwo zrobić: po prostu odczytaj wartość b ostatniego bajtu, a następnie sprawdź, czy wszystkie ostatnie b bajty wyniku mają wartość b.) Jeśli tak,

następnego dopełnienie jest usuwane i zwracana jest oryginalna wiadomość. W przeciwnym razie standardową procedurą jest zwroćenie błędu „złygo dopełnienia” (np. w Java standardowy wyjątek nazywa się `javax.crypto.BadPaddingException`). Obecność takiego komunikatu o błędzie zapewnia przeciwnikowi czas ściągnięcia odszyfrowanie wyroczni. Oznacza to, że przeciwnik może wysłać do serwera dowolny tekst zaszyfrowany i dowiedzieć się (na podstawie tego, czy zwrocony został błąd „złygo dopełnienia”, czy nie), czy podstawowe zakodowane dane są dopełnione we właściwym formacie. Chociaż może się to wydawać bezsensowną informacją, pokazujemy, że umożliwia ona przeciwnikowi całkowite odzyskanie oryginalnej wiadomości dla dowolnego wybranego tekstu zaszyfrowanego.

Dla uproszczenia opisujemy atak na 3-blokowy tekst zaszyfrowany. Niech IV, c_1, c_2 będą szyfrrogramem zaobserwowanym przez atakującego i niech m_1, m_2 będą bazowymi zakodowanymi danymi (nieznany atakującemu), które odpowiadają uzupełnionej wiadomości, jak omówiono powyżej. (Każdy blok ma długość L bajtów.) Zauważ, że

$$m_2 = F_k^{-1}(c_2) \quad c_1, (3.15) \text{ gdzie } k \text{ jest kluczem (który oczywiście)}$$

nie jest znany atakującemu) używanym przez uczciwe strony. Drugi blok m_2 kończy się na $0xb \cdots 0xb$, gdzie pozwalamy

— — —
b razy

$0xb$ oznaczają 1-bajtową reprezentację liczby całkowitej b . Kluczową właściwością wykorzystywaną w ataku jest to, że pewne zmiany w zaszyfrowanym tekście powodują przewidywalne zmiany w zakodowanych danych po odszyfrowaniu w trybie CBC. W szczególności niech b będzie identyczny z c_1 , z wyjątkiem c_2 modyfikacji w ostatnim bajcie. Rozważ c_2 . Spowoduje to zakodowanie

1 odszyfrowanie zmodyfikowanego tekstu zaszyfrowanego 1,

IV, c dane $m_1, m_2 = F_k^{-1}(c_2)$ do 1. W porównaniu do równania (3.15) my m_2 zobaczymy, że m_2 będzie identyczne z m_2 z wyjątkiem modyfikacji w ostatnim bajcie. (Wartość m jest nieprzewidywalna, ale nie wpłynie to negatywnie na atak.) Podobnie, jeśli c_1 jest takie samo jak c_1 , z wyjątkiem zmiany w jego i -tym bajcie, to deszyfrowanie IV, c jest takie samo jak m_2 z wyjątkiem zmiana jego i -tego bajtu. Mówiąc bardziej ogólnie, dla dowolnego i dla dowolnego m_i , to deszyfrowanie IV, c_1, c_2 daje m_1, m_2 gdzie $m_2 = m_2$. W rezultacie atakujący może sprawować znaczną kontrolę nad ostatnim blokiem zakodowanych danych.

Na rozgrzewkę przyjrzyjmy się, jak przeciwnik może to wykorzystać, aby poznać b , ilość wypełnienia. (To ujawnia długość oryginalnej wiadomości). Przypomnijmy, że po odszyfrowaniu odbiorca sprawdza wartość b ostatniego bajtu drugiego bloku zakodowanych danych, a następnie sprawdza, czy wszystkie ostatnie bajty b mają tę samą wartość. Osoba atakująca zaczyna od modyfikacji pierwszego bajtu c_1 i c_2 w odbiorniku. Jeśli deszyfrowanie nie powiedzie się, wysyłając wynikowy tekst zaszyfrowany IV, c_1 (tj. sprawdza wszystkie L bajtów m_2 , a zatem $b = L$)! W odbiorca zwroci błąd, to musi być tak, że odbiorca przeciwnym razem atakujący dowiaduje się, że $b < L$ i może następnie powtórzyć proces z drugim bajtem i tak dalej. Zmodyfikowany bajt znajdujący się najbardziej na lewo, dla którego odszyfrowanie się nie powiodło, wskazuje dokładnie bajt znajdujący się najbardziej na lewo, który jest sprawdzany przez odbiorcę, a zatem ujawnia dokładnie b .

Znając b , atakujący może przystąpić do poznawania bajtów wiadomości jeden po drugim. Zilustrujemy pomysł na ostatni bajt wiadomości, który mamy

oznaczony przez B. Atakujący wie, że m2 kończy się na 0xB0xb ··· 0xb (z 0xb powtórzone b razy) i chce się nauczyć B. Zdefiniuj

$$\begin{array}{c} \text{def } \\ i = 0x00 + 10x00 \underbrace{0x(b+1)}_{\text{razy}} \\ \hline \end{array} \quad \begin{array}{c} \text{razy} \\ \hline \end{array}$$

$$\begin{array}{c} \text{razy} \\ \hline \end{array} \quad \begin{array}{c} \text{razy} \\ \hline \end{array}$$

$$\begin{array}{c} 0x00 \cdots 0x00 \underbrace{0x00}_{\text{0xb} \cdots \text{0xb}} \\ \hline \end{array}$$

dla $0 \leq i < 2^8$; tj. końcowe $b+1$ bajty i zawierają liczbę całkowitą i (reprezentującą w formacie szesnastkowym), po którym następuje wartość $(b+1) \cdot b$ (w formacie szesnastkowym) powtórzone b razy. Jeżeli atakujący poda szyfrrogram IV, $c_1 \dots c_1, c_2 \dots c_2$ do odbiornika, wówczas, po odszyfrowaniu w trybie CBC, wynikowe zakodowane dane zostaną równe $0x(B-i)0x(b+1) \cdots 0x(b+1)$ (przy $0x(b+1)$ powtórzonym b razy). Odszyfrowanie nie powiedzie się, chyba że $0x(B-i) = 0x(b+1)$. Atakujący po prostu musi spróbować maksymalnie 28 wartości $0, \dots, 2^8 - 1$, dopóki deszyfrowanie nie powiedzie się dla pewnego i . Na którym punkcie może wywnioskować, że $B = 0x(b+1) - i$. Zostawiamy pełny opis jak rozszerzyć ten atak, aby poznać cały tekst jawnego — a nie tylko ostatni blok — jako ćwiczenie.

Atak dopełniający-oracle na CAPTCHA. CAPTCHA jest zniekształcony obraz, powiedzmy, angielskiego słowa, które jest łatwe do odczytania dla ludzi, ale trudne dla niego komputer do przetwarzania. Kody CAPTCHA są używane w celu zapewnienia, że użytkownik — a nie jakieś zautomatyzowane oprogramowanie — wchodzi w interakcję ze stroną internetową. Kody CAPTCHA są używane np. w internetowych usługach e-mail w celu zapewnienia, że ludzie otwarte konta; jest to ważne, aby zapobiec automatycznemu spamowaniu otwieranie tysięcy kont i wykorzystywanie ich do wysyłania spamu.

Jednym ze sposobów konfiguracji CAPTCHA jest uruchomienie oddzielnej usługi niezależny serwer. Aby zobaczyć jak to działa, oznaczamy serwer WWW przez SW, serwer CAPTCHA przez SC i użytkownika przez U. Gdy użytkownik po załadowaniu strony internetowej obsługiwanej przez SW mają miejsce następujące zdarzenia: SW szyfruje losowe angielskie słowo w przy użyciu klucza k, które było początkowo wspólne strony, i wysyła wynikowy tekst zaszyfrowany do użytkownika (wraz z plikiem internetowe SW i SC). U przekazuje zaszyfrowany tekst do SC, który go odszyfrowuje, uzyskuje w i renderuje zniekształcony obraz w do U. Na koniec U wysyła słowo w z powrotem do SW do weryfikacji. Ciekawą obserwacją jest to, że SC odszyfrowuje każdy tekst zaszyfrowany, który otrzyma od U i wyświetli komunikat o błędzie „złego dopełnienia”, jeśli odszyfrowanie kończy się niepowodzeniem, jak opisano wcześniej. Daje Ci to szansę przeprowadzić atak dopełniający-wyrocznię, jak opisano powyżej, i w ten sposób rozwiązać problem CAPTCHA (tj. określenie w) automatycznie, bez udziału człowieka, co sprawia, że CAPTCHA jest nieskuteczne. Chociaż środki ad hoc mogą być użyte (np. zwrócenie przez SC losowego obrazu zamiast błędu deszyfrowania), tak naprawdę potrzebne jest zastosowanie schematu szyfrowania zabezpieczonego przez CCA.

Referencje i dodatkowe lektury

Nowoczesne obliczeniowe podejście do kryptografii zostało zapoczątkowane w przełomowym artykule Goldwassera i Micali [80]. W artykule tym wprowadzono pojęcie bezpieczeństwa semantycznego i pokazano, w jaki sposób cel ten można osiągnąć w przypadku szyfrowania kluczem publicznym (patrz rozdziały 10 i 11). W artykule tym zaproponowano także pojęcie nierozróżnialności (tj. Definicja 3.8) i wykazano, że implikuje ono bezpieczeństwo semantyczne. Odwrotną sytuację pokazano w [125]. Dalszą dyskusję na temat bezpieczeństwa semantycznego odsyłamy do [76].

Blum i Micali [37] wprowadzili pojęcie generatorów pseudolosowych i udowodnili ich istnienie w oparciu o określone założenia z teorii liczb.

W tej samej pracy Blum i Micali wskazali również na związek pomiędzy generatorami pseudolosowymi a szyfrowaniem kluczem prywatnym, jak w Construction 3.17.

Definicja generatorów pseudolosowych podana przez Bluma i Micaliego różni się od definicji, której używamy w tej książce (Definicja 3.14); ta ostatnia definicja wywodzi się z prac Yao [179], który wykazał równoważność obu sformułowań. Yao pokazał także konstrukcje generatorów pseudolosowych oparte na ogólnych założeniach; zbadamy ten wynik w rozdziale 7.

Formalne definicje zabezpieczeń przed atakami z użyciem wybranego tekstu jawnego podali Luby [115] oraz Bellare i in. [15]. Ataki z użyciem wybranego tekstu zaszyfrowanego (w kontekście szyfrowania klucza publicznego) zostały po raz pierwszy formalnie zdefiniowane przez Naora i Yunga [129] oraz Rackoffa i Simona [147], a także zostały omówione w [61] i [15]. Zobacz [101], aby poznać inne koncepcje bezpieczeństwa szyfrowania klucza prywatnego.

Funkcje pseudolosowe zostały zdefiniowane i skonstruowane przez Goldreicha i in. [78], a ich zastosowanie do szyfrowania wykazano w kolejnych pracach tych samych autorów [77]. Permutacje pseudolosowe i silne permutacje pseudolosowe badali Luby i Rackoff [116]. Dowód Twierdzenia 3.27 („lemat o przełączaniu”) można znaleźć w [24]. Szyfry blokowe były stosowane przez wiele lat, zanim zaczęto je badać w sensie teoretycznym zapoczątkowanym przez powyższe prace. Praktyczne szyfry strumieniowe i szyfry blokowe zostały szczegółowo omówione w rozdziale 6.

Tryby działania ECB, CBC i OFB (a także CFB, tryb działania nie opisany tutaj) zostały ujednolicone wraz z szyfrem blokowym DES [130]. Tryb CTR został ustandaryzowany przez NIST w 2001 r. W [15] udowodniono, że tryby CBC i CTR są bezpieczne dla CPA. Nowsze tryby działania można znaleźć na stronie <http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html>.

Atak na łańcuchowe CBC został po raz pierwszy opisany przez Rogawaya (niepublikowany) i został wykorzystany do ataku na SSL/TLS w tak zwany „atak BEAST” przeprowadzonym przez Rizzo i Duonga. Atak wyroczni, który tutaj opisujemy, powstał w pracy Vaudenaya [171].

Ćwiczenia

3.1 Dowód twierdzenia 3.6.

3.2 Udowodnić, że Definicja 3.8 nie może być spełniona, jeśli Π może szyfrować wiadomości o dowolnej długości, a przeciwnik nie jest ograniczony do wysyłania wiadomości o równej długości w eksperymencie $\text{PrivKeav}_{A,\Pi}$.

Wskazówka: Niech $q(n)$ bę dzie górną granicą wielomianu długości zaszyfrowanego tekstu, gdy Π jest używane do szyfrowania pojedynczego bitu. Nastę pnie rozważ przeciwnika, który wyprowadza $m_0 \in \{0, 1\}^n$ i jednorodny $m_1 \in \{0, 1\}^{q(n)+2}$

3.3 Powiedzmy, że $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ jest takie, że dla $k \in \{0, 1\}^n$ algorytm Enck jest zdefiniowany tylko dla komunikatów o długości co najwyżej (n) (dla pewnego wielomianu). Skonstruj schemat spełniający definicję 3.8, nawet jeśli przeciwnik nie jest ograniczony do wysyłania komunikatów o równej długości w $\text{PrivKeav}_{A,\Pi}$.

3.4 Udowodnić równoważność definicji 3.8 i definicji 3.9.

3.5 Niech $|G(s)| = |\$|$ dla niektórych. Rozważmy następujący eksperyment:

Eksperyment nieroróżnialności PRG $\text{PRGA}_G(n)$: (a) Wybierany jest jednolity bit $b \in \{0, 1\}$. Jeśli $b = 0$ to wybierz jednorodny $r \in \{0, 1\}^n$; jeśli $b = 1$ to wybierz uniformny $r \in \{0, 1\}^n$ i ustaw $r := G(s)$. (b) Przeciwnik A otrzymuje r i wyprowadza bit b . (c) Wynik eksperymentu definiuje się jako 1, jeśli $b = b$, i 0 w przeciwnym razie.

Podaj definicję generatora pseudolosowego na podstawie tego eksperymentu i udowodnij, że Twoja definicja jest równoważna definicji 3.14.

(To znaczy pokaż, że G spełnia Twoją definicję wtedy i tylko wtedy, gdy spełnia definicję 3.14.)

3.6 Niech G bę dzie generatorem pseudolosowym ze współczynnikiem rozszerzenia $(n) > 2n$. W każdym z poniższych przypadków powiedz, czy G jest koniecznie generatorem pseudolosowym. Jeśli tak, podaj dowód; jeśli nie, pokaż kontrprzykład.

- (a) Zdefiniuj $G(s) \stackrel{\text{def}}{=} G(s_1 \cdots s_{n/2})$, gdzie $s = s_1 \cdots s_n$.
- (b) Zdefiniuj $G(s) \stackrel{\text{def}}{=} G(0\$|s|)$.
- (c) Zdefiniuj $G(s) \stackrel{\text{def}}{=} G(s)G(s+1)$.

3.7 Udowodnić odwrotność twierdzenia 3.18. Mianowicie pokaż, że jeśli G nie jest generatorem pseudolosowym, to Konstrukcja 3.17 nie ma nieroróżnialnych szyfrowań w obecności podsłuchującego.

3.8 (a) Zdefiniuj pojęcie nieroróżnialności w przypadku szyfrowania wielu różnych wiadomości, w którym schemat nie musi ukrywać, czy ta sama wiadomość jest szyfrowana dwukrotnie.

(b) Pokaż, że Konstrukcja 3.17 nie spełnia Twojej definicji. (c) Podaj konstrukcję deterministycznego (bezstanowego) schematu szyfrowania, który spełnia twoją definicję .

3.9 Udowodnij bezwarunkowo istnienie funkcji pseudolosowej $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ z kluczem $(n) = n$ i $\text{in}(n) = O(\log n)$.

Wskazówka: Zaimplementuj funkcję jednolitą z logarytmiczną długością wejściową.

3.10 Niech F będzie funkcją pseudolosową zachowującą długość. Dla funkcji z kluczem $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{n-1}$ określ, czy F jest funkcją pseudolosową. Jeśli tak, udowodnij to; jeśli nie, pokaż atak.

$$(a) F_k(X) \stackrel{\text{def}}{=} F_k(0x) F_k(1x).$$

$$(b) F_k(X) \stackrel{\text{def}}{=} F_k(0x) F_k(x1).$$

3.11 Zakładając istnienie funkcji pseudolosowych, udowodnij, że istnieje schemat szyfrowania, który w obecności osoby podsłuchującej ma nieroróżnialne wielokrotne szyfrowania (tj. spełnia definicję 3.19), ale nie jest bezpieczny według CPA (tj. nie spełnia definicji 3.22).

Wskazówka: schemat nie musi być „naturalny”. Będziesz musiał wykorzystać ten fakt

że w ataku z wybranym tekstem jawnym przeciwnik może adaptacyjnie wybierać swoje zapytania do wyroczni szyfrującej.

3.12 Niech F będzie funkcją kluczową i rozważmy następujący eksperyment:

Eksperyment nieroróżnialności PRF PRFA, $F(n)$: (a) Wybierany jest jednolity bit $b \in \{0, 1\}$. Jeżeli $b = 1$ to wybierz jednorodne $k \in \{0, 1\}^n$. (b) A ma dane

wejściowe 1^n . Jeżeli $b = 0$, to A ma dostęp do funkcji jednorodnej f_{Funcn} . Jeżeli $b = 1$, wówczas A ma zamiast tego dostęp do $F_k(\cdot)$. (c) A wyprowadza bit b . (d) Wynik

eksperymentu definiuje się jako 1, jeśli $b = b$, i 0 w przeciwnym razie.

Zdefiniuj funkcje pseudolosowe za pomocą tego eksperymentu i udowodnij, że Twoja definicja jest równoważna definicji 3.25.

3.13 Rozważmy następującą funkcję z kluczem F : Dla parametru bezpieczeństwa n kluczem jest $n \times n$ macierz boolowska A i n -bitowy wektor boolowski b . Definiować $F_{A,b} : \{0, 1\}^n \rightarrow \{0, 1\}^n$ przez $F_{A,b}(x) \stackrel{\text{def}}{=} Ax + b$, gdzie wszystkie operacje są wykonywane modulo 2. Pokaż, że F nie jest funkcją pseudolosową.

3.14 Udowodnij, że jeśli F jest funkcją pseudolosową zachowującą długość, to $= F(s(1))F(s(2)) \cdot \dots \cdot F(s(n))$ jest generatorem pseudolosowym z eksponentem r i współczynnikiem renty λ .

3.15 Zdefiniuj pojęcie doskonałej tajemnicy w przypadku ataku wybranym tekstem jawnym, dostosowując Definicję 3.22. Pokaż, że definicja nie może zostać osiągnięta.

3.16 Dowód twierdzenia 3.27.

Wskaźówka: Skorzystaj z wyników Załącznika A.4.

3.17 Założymy, że istnieją permutacje pseudolosowe. Pokaż, że istnieje funkcja F , która jest permutacją pseudolosową, ale nie jest silną permutacją pseudolosową.

Wskaźówka: Skonstruuj F w taki sposób, że $F_k(k) = 0^{\|k\|}$.

3.18 Niech F będzie permutacją pseudolosową i zdefiniuje schemat szyfrowania o stałej długości (Enc, Dec) w następujący sposób: Na wejściu $m \in \{0, 1\}^{n/2}$ i kluczu $k \in \{0, 1\}^n$, algorytm Enc wybiera jednolity串 $r \in \{0, 1\}^{n/2}$ o długości $n/2$ i oblicza $c := F_k(r)m$.

Pokaż, jak odszyfrować i udowodnij, że ten schemat jest bezpieczny CPA dla wiadomości o długości $n/2$. (Jeśli szukasz prawdziwego wyzwania, udowodnij, że ten schemat jest bezpieczny pod względem CCA, jeśli F jest silną permutacją pseudolosową.)

3.19 Niech F będzie funkcją pseudolosową, a G generatorem pseudolosowym ze współczynnikiem rozszerzalności (n) = $n+1$. Dla każdego z poniższych schematów szyfrowania określ, czy schemat ma szyfrowanie nieroróżnicalne w obecności podsłuchującego i czy jest to CPA-bezpieczne. (W każdym przypadku klucz wspólnie dzielony jest jednolitym $k \in \{0, 1\}^n$.) Uzasadnij swoją odpowiedź.

(a) Aby zaszyfrować $m \in \{0, 1\}^{n+1}$, wybierz uniform $r \in \{0, 1\}^n$ i wyrowadź zaszyfrowany串 $r, G(r) \oplus m$.

(b) Aby zaszyfrować $m \in \{0, 1\}^n$, wyrowadź zaszyfrowany串 m

$F_k(0^n)$. (c) Aby zaszyfrować $m \in \{0, 1\}^{2n}$, przeanalizuj m jako m_1m_2 za pomocą $|m_1| = |m_2|$, następnie wybierz uniform $r \in \{0, 1\}^n$ i wyślij $r, m_1 \oplus F_k(r), m_2 \oplus F_k(r+1)$.

3.20 Rozważ stanowy wariant szyfrowania w trybie CBC, w którym nadawca po prostu zwiększa IV o 1 za każdym razem, gdy wiadomość jest szyfrowana (zamiast wybierać losowo IV za każdym razem). Pokaż, że powstały schemat nie jest bezpieczny pod względem CPA.

3.21 Jaki jest skutek jednobitowego błędu w zaszyfrowanym tekście podczas korzystania z trybów działania CBC, OFB i CTR?

3.22 Jaki jest skutek porzuconego bloku tekstu zaszyfrowanego (np. jeśli przesyłany串 jest zaszyfrowany c_1, c_2, c_3, \dots zostanie odebrany jako c_1, c_3, \dots) podczas korzystania z trybów działania CBC, OFB i CTR?

3.23 Założymy, że szyfrowanie w trybie CBC jest używane z szyfrem blokowym posiadającym 256-bitowy klucz i 128-bitową długość bloku do szyfrowania wiadomości 1024-bitowej. Jaka jest długość otrzymanego tekstu zaszyfrowanego?

3.24 Podaj szczegóły dowodu redukcyjnego dla równania (3.12).

3.25 Niech F będzie funkcją pseudolosową taką, że dla $k \in \{0, 1\}^n$ funkcja F_k odwzorowuje wejścia $in(n)$ -bitowe na wyjścia (n) -bitowe.

(a) Rozważ wdrożenie szyfrowania w trybie CTR przy użyciu F . Dla którego wynikowego schematu szyfrowania jest bezpieczny CPA? (b) Rozważ wdrożenie szyfrowania w trybie CTR przy użyciu F , ale tylko dla wiadomości o stałej długości i długości (n) (która jest całkowitą wielokrotnością $out(n)$). W przypadku którego schematu szyfrowanie jest niemożliwe do odróżnienia w obecności osoby podsłuchującej?

3.26 Dla dowolnej funkcji $g : \{0, 1\}^n \rightarrow \{0, 1\}^n$ być wyroczną probabilistyczną, która na wejściu 1^n wybiera jednolitą $r \in \{0, 1\}^n$ i zwraca $r, g(R)$. Funkcja kluczowana F jest słabą funkcją pseudolosową, jeśli dla wszystkich algorytmów $PPT D$ istnieje pomijalna funkcja negl taka, że:

$$\Pr[D \xrightarrow{F_k^{\$}} (\cdot)(1^n) = 1] - \Pr[D \xrightarrow{F^{\$}} (\cdot)(1^n) = 1] = negl(n),$$

gdzie $k \in \{0, 1\}^n$ i $f = Func_n$ dobierane są równomiernie.

(a) Udowodnij, że jeśli F jest pseudolosowe, to jest słabo pseudolosowe. (b) Niech F będzie funkcją pseudolosową i zdefiniuj ją

$$F_k(x) \stackrel{\text{def}}{=} \begin{cases} F_k(x) & \text{jeśli } x \text{ jest parzyste} \\ F_{k+1}(x) & \text{jeśli } x \text{ jest nieparzyste.} \end{cases}$$

Udowodnić, że F jest słabo pseudolosowe, ale nie pseudolosowe.

- (c) Czy szyfrowanie w trybie CTR przy użyciu słabej funkcji pseudolosowej koniecznie zapewnia bezpieczeństwo CPA? Czy koniecznie ma nieroróżnialne szyfrowanie w obecności podsłuchującego? Udowodnij swoje odpowiedzi.
 (d) Udowodnij, że konstrukcja 3.30 jest bezpieczna pod względem CPA, jeśli F jest słabą funkcją pseudolosową.

3.27 Niech F będzie permutacją pseudolosową. Rozważmy tryb działania, w którym wybierana jest jednolita wartość $ctr \in \{0, 1\}^n$, a i -ty blok tekstu zaszyfrowanego c_i jest obliczany jako $c_i := F_k(ctr + i \cdot mi)$. Pokaż, że w tym schemacie nie ma szyfrowania niemożliwego do odróżnienia w obecności osoby podsłuchującej.

3.28 Pokaż, że tryby działania CBC, OFB i CTR nie dają schematów szyfrowania zabezpieczonych CCA (niezależnie od F).

3.29 Niech $\Pi_1 = (\text{Enc}_1, \text{Dec}_1)$ i $\Pi_2 = (\text{Enc}_2, \text{Dec}_2)$ będą dwoma schematami szyfrowania, dla których wiadomo, że co najmniej jeden jest zabezpieczony CPA (ale nie wiadomo który). Pokaż, jak skonstruować schemat szyfrowania Π , który gwarantuje bezpieczeństwo CPA, o ile co najmniej jeden z Π_1 lub Π_2 jest bezpieczny CPA. Podaj pełny dowód swojego rozwiązania.

Wskazówka: Wygeneruj dwie wiadomości w postaci zwykłego tekstu z oryginalnego tekstu jawnego, tak aby znajomość jednego z nich nie ujawniła niczego na temat oryginalnego tekstu jawnego, ale znajomość obu umożliwia obliczenie oryginalnego tekstu jawnego.

3.30 Napisz pseudokod w celu uzyskania całego tekstu jawnego poprzez atak Oracle z dopełnieniem na szyfrowanie w trybie CBC przy użyciu dopełnienia PKCS #5, jak opisano w tekście.

3.31 Opisz atak Oracle z dopełnianiem na szyfrowanie w trybie CTR (zakładając, że dopełnianie PKCS #5 jest używane do dopełniania wiadomości do wielokrotności długości bloku przed szyfrowaniem).

Rozdział 4

Kody uwierzytelniające wiadomości

4.1 Integralność wiadomości

4.1.1 Tajemnica a integralność

Jednym z najbardziej podstawowych celów kriptografii jest umożliwienie stronom bezpiecznej komunikacji za pośrednictwem otwartego kanału komunikacji. Ale co oznacza „bezpieczna komunikacja”? W rozdziale 3 pokazaliśmy, że możliwe jest uzyskanie tajnej komunikacji za pośrednictwem otwartego kanału. Oznacza to, że pokazaliśmy, jak można zastosować szyfrowanie, aby uniemożliwić podsłuchiwaczowi (lub być może bardziej aktywnemu przeciwnikowi) dowiedzenie się czegokolwiek o treści wiadomości wysyłanych niezabezpieczonym kanałem komunikacyjnym. Jednak nie wszystkie obawy związane z bezpieczeństwem są związane z tajemnicą. W wielu przypadkach równie lub więcej znaczenie ma zagwarantowanie integralności wiadomości (lub uwierzytelnienia wiadomości) w tym sensie, że każda ze stron powinna być w stanie określić, kiedy otrzymana przez nią wiadomość została wysłana przez stronę twierdzącą, że ją wysłała i nie została zmodyfikowana w tranzycie. Przyjrzymy się dwóm kanonicznym przykładom.

Rozważmy przypadek użytkownika komunikującego się ze swoim bankiem za pośrednictwem Internetu. Kiedy bank otrzymuje prośbę o przelanie 1000 dolarów z konta użytkownika na konto innego użytkownika X, bank musi wziąć pod uwagę następujące kwestie:

1. Czy prośba jest autentyczna? To znaczy, czy dany użytkownik rzeczywiście wysłał to żądanie, czy też żądanie to zostało wydane przez przeciwnika (być może samego X), który podszywa się pod prawdziwego użytkownika?
2. Zakładając, że żądanie przeniesienia zostało wysłane przez uprawnionego użytkownika, czy otrzymane szczegóły żądania są zgodne z zamierzeniami uprawnionego użytkownika? A może zmieniono np. kwotę przelewu?

Należy zauważyć, że w przypadku drugiego problemu nie wystarczą standardowe techniki korekcji błędów. Kody korygujące błędy mają na celu jedynie wykrywanie i naprawianie „losowych” błędów, które wpływają tylko na niewielką część transmisji, ale nie chronią przed złośliwym przeciwnikiem, który może dokładnie wybrać, gdzie wprowadzić dowolną liczbę błędów.

Drugi scenariusz, w którym w praktyce pojawia się potrzeba integralności wiadomości, dotyczy plików cookie sieciowych. Protokół HTTP używany w ruchu internetowym jest bezstanowy, więc gdy klient i serwer komunikują się w jakiejś sesji (np.

użytkownik [klient] robi zakupy na stronie internetowej sprzedawcy [serwera]), każdy stan wygenerowany w ramach tej sesji (np. zawartość koszyka użytkownika) jest często umieszczany w „pliku cookie”, który jest przechowywany przez klienta i wysyłany od klienta do serwera w ramach każdej wiadomości wysyłanej przez klienta. Założymy, że plik cookie przechowywany przez jakiegoś użytkownika obejmuje pozycje w koszyku użytkownika wraz z ceną za każdą pozycję, co mogłoby mieć miejsce, gdyby sprzedawca oferował różne ceny różnym użytkownikom (odzwierciedlające np. rabaty i promocje lub specyficzne dla użytkownika cennik). W tym przypadku nie powinno być możliwe modyfikowanie przez użytkownika przechowywanego przez siebie pliku cookie w celu zmiany cen artykułów w jego koszyku. Sprzedawca potrzebuje zatem techniki zapewniającej integralność pliku cookie przechowywanego u użytkownika. Należy pamiętać, że zawartość pliku cookie (mianowicie produkty i ich ceny) nie jest tajemnicą i w rzeczywistości musi być znana użytkownikowi. Problem polega zatem wyłącznie na uczciwości.

Generalnie nie można zakładać integralności komunikacji bez podjęcia konkretnych działań zapewniających jej zapewnienie. Zasadniczo nie można ufać, że jakiekolwiek niezabezpieczone zamówienie zakupu online, operacja bankowości internetowej, wiadomość e-mail lub wiadomość SMS pochodzą z zastrzeżonego źródła i nie zostały zmodyfikowane podczas transportu. Niestety, ludzie są na ogół ufni, dlatego informacje takie jak identyfikacja rozmówcy lub adres zwrotny są w wielu przypadkach uznawane za „dowody pochodzenia”, mimo że stosunkowo łatwo je sfałszować. Pozostawia to otwarte drzwi dla potencjalnie szkodliwych ataków.

W tym rozdziale pokażemy, jak osiągnąć integralność wiadomości za pomocą technik kryptograficznych, aby zapobiec niewykrytemu manipulowaniu wiadomościami przesyłanymi otwartym kanałem komunikacyjnym. Należy pamiętać, że nie możemy mieć nadzieję, że uda nam się całkowicie zapobiec wrogiemu manipulowaniu wiadomościami, ponieważ można się przed tym obronić jedynie na poziomie fizycznym. Zamiat tego gwarantujemy, że wszelkie takie manipulacje zostaną wykryte przez uczciwe strony.

4.1.2 Szyfrowanie a uwierzytelnianie wiadomości

Tak jak cele związane z tajemnicą i integralnością przekazu są różne, tak różne są techniki i narzędzia ich osiągnięcia. Niestety, tajemnica i integralność są często mylone i niepotrzebnie ze sobą powiązane, więc postawmy sprawę jasno: szyfrowanie nie zapewnia (ogólnie) jakiekolwiek integralności, a szyfrowania nigdy nie należy używać z zamiarem uzyskania uwierzytelnienia wiadomości, chyba że jest ono specjalnie zaprojektowane z myślą o mając na uwadze ten cel (coś, do czego powrócimy w sekcji 4.5).

Można błędnie sądzić, że szyfrowanie rozwiązuje problem uwierzytelniania wiadomości. (W rzeczywistości jest to częsty błąd). Wynika to z niejasnego i niepoprawnego rozumowania, zgodnie z którym skoro tekst zaszyfrowany całkowicie ukrywa treść wiadomości, przeciwnik nie jest w stanie w żaden znaczący sposób zmodyfikować zaszyfrowanej wiadomości. Pomimo swojej intuicyjnej atrakcyjności, to rozumowanie jest całkowicie fałszywe. Zilustrujemy ten punkt, pokazując, że wszystkie schematy szyfrowania, które widzieliśmy do tej pory, nie zapewniają integralności wiadomości.

Szyfrowanie za pomocą szyfrów strumieniowych. Rozważmy prosty schemat szyfrowania, w którym $Enck(m)$ oblicza zaszyfrowany tekst $c := G(k) \oplus m$, gdzie G jest generatorem pseudolosowym. Zaszyfrowanymi tekstami w tym przypadku można bardzo łatwo manipulować: odwrócenie dowolnego bitu w zaszyfrowanym tekście c powoduje odwrócenie tego samego bitu w wiadomości, która jest odzyskiwana po odszyfrowaniu. Zatem, mając zaszyfrowany tekst c , który szyfruje (prawdopodobnie nieznaną) wiadomość m , możliwe jest wygenerowanie zmodyfikowanego tekstu zaszyfrowanego c w taki sposób, że $m := Deck(c)$ jest taki sam jak m , ale z jednym (lub więcej) ksymą liczbą zamienionych bitów. Ten prosty atak może mieć poważne konsekwencje. Jako przykład rozważmy przypadek użytkownika szyfrującego pewną kwotę w dolarach, który chce przelać ze swojego konta bankowego, gdzie kwota ta jest reprezentowana w formacie binarnym. Odwrócenie najmniej znaczącego bitu powoduje zmianę tej kwoty tylko o 1 dolara, ale odwrócenie 11-tego najmniej znaczącego bitu zmienia kwotę o ponad 1000 dolarów! (Co ciekawe, przeciwnik w tym przykładzie niekoniecznie dowiaduje się, czy zwiększa, czy zmniejsza początkową kwotę, tj. czy zamienia 0 na 1, czy odwrotnie. Ale jeśli przeciwnik ma częściową wiedzę na temat kwoty – powiedzmy, że na początek jest to mniej niż 1000 dolarów – wtedy modyfikacje, które wprowadza, mogą mieć przewidywalny efekt.) Podkreślamy, że ten atak nie zaprzecza tajności schematu szyfrowania (w rozumieniu Definicji 3.8). W rzeczywistości dokładnie ten sam atak dotyczy schematu szyfrowania jednorazowej podkładki, pokazując, że nawet doskonała tajemnica nie jest wystarczająca, aby zapewnić najbardziej podstawowy poziom integralności wiadomości.

Szyfrowanie przy użyciu szyfrów blokowych. Opisany powyżej atak wykorzystuje fakt, że odwrócenie pojedynczego bitu w zaszyfrowanym tekście powoduje, że tekst jawnego pozostaje niezmieniony, z wyjątkiem odpowiedniego bitu (który również jest odwracany). Ten sam atak dotyczy schematów szyfrowania w trybie OFB i CTR, które również szyfrują wiadomości poprzez XORowanie ich strumieniem pseudolosowym (aczkolwiek zmieniającym się za każdym razem, gdy wiadomość jest szyfrowana). Widzimy zatem, że nawet użycie szyfrowania zabezpieczonego CPA nie wystarczy, aby zapobiec manipulowaniu wiadomościami.

Można mieć nadzieję, że zaatakowanie szyfrowania w trybie EBC lub CBC byłoby trudniejsze, ponieważ deszyfrowanie w tych przypadkach obejmuje odwrócenie (silnego) permutacji pseudolosowej F i oczekujemy, że $F^{-1}(x) \oplus F^{-1}(x) = x$.
 bę, gdzie całkowicie nieskorelowane, nawet jeśli x i x' różnią się tylko jednym bitem.
 (Oczywiście tryb EBC nie gwarantuje nawet najbardziej podstawowego pojęcia tajemnicy, ale nie ma to znaczenia dla niniejszej dyskusji.) Niemniej jednak jednobitowe modyfikacje tekstu zaszyfrowanego nadal powodują częściowo przewidywalne zmiany w tekście jawnym. Na przykład, podczas korzystania z trybu EBC, lekkie przesunięcie bitu w i -tym bloku tekstu zaszyfrowanego wpływa tylko na i -ty blok tekstu jawnego - wszystkie pozostałe bloki pozostają niezmienione. Choć wpływ na i -ty blok tekstu jawnego może być niemożliwy do przewidzenia, zmiana tego jednego bloku (przy pozostawieniu wszystkich pozostałych bez zmian) może stanowić szkodliwy atak. Co więcej, kolejność bloków tekstu jawnego można zmienić (bez zniekształcania żadnego bloku), po prostu zmieniając kolejność odpowiednich bloków tekstu zaszyfrowanego, a wiadomość można skrócić, po prostu usuwając bloki tekstu zaszyfrowanego.

Podobnie, gdy używasz trybu CBC, odwrócenie j -tego bitu IV zmienia się tylko

j-ty bit pierwszego bloku komunikatów m_1 (ponieważ $m_1 := F \text{ jest}_k^{-1}(c_1) \oplus IV$, gdzie IV zmodyfikowanym IV); wszystkie bloki tekstu jawnego inne niż pierwszy pozostają niezmienione (ponieważ i-ty blok tekstu jawnego jest obliczany jako $m_i := F \text{ bloki ci}_i \oplus_k^{-1}(c_i) \oplus c_{i-1}$, oraz c_{i-1} nie zostały zmodyfikowane). Dlatego pierwszy blok wiadomości zaszyfrowanej CBC można dowolnie zmieniać. W praktyce stanowi to poważny problem, ponieważ pierwszy blok często zawiera ważne informacje nagłówka.

Na koniec należy zauważyc, że wszystkie schematy szyfrowania, które widzieliśmy do tej pory, mają tę właściwość, że każdy zaszyfrowany tekst (być może spełniający pewne ograniczenie długości) odpowiada jakiejś ważnej wiadomości. Zatem dla przeciwnika „sfałszowanie” wiadomości w imieniu jednej z komunikujących się stron – poprzez wysłanie dowolnego tekstu zaszyfrowanego – jest trywialne, nawet jeśli przeciwnik nie ma pojęcia, jaka będzie treść tej wiadomości. Jak zobaczymy, kiedy formalnie zdefiniujemy szyfrowanie uwierzytelnione w sekcji 4.5, należy wykluczyć nawet atak tego rodzaju.

4.2 Kody uwierzytelniające wiadomości – definicje

Widzieliśmy, że ogólnie szyfrowanie nie rozwiązuje problemu integralności wiadomości. Zamiast tego potrzebny jest dodatkowy mechanizm, który umożliwi komunikującym się stronom dowiedzenie się, czy wiadomość została sfałszowana. Właściwym narzędziem do tego zadania jest kod uwierzytelniający wiadomość (MAC).

Celem kodu uwierzytelniającego wiadomość jest uniemożliwienie przeciwnikowi zmodyfikowania wiadomości wysłanej przez jedną stronę do drugiej lub wstrzygnięcie jej nowej wiadomości tak, aby odbiorca nie wykrył, że wiadomość nie pochodzi od strony zamierzanej. Podobnie jak w przypadku szyfrowania, jest to możliwe tylko wtedy, gdy komunikujące się strony podzielą się jakąś tajemnicą, której przeciwnik nie zna (w przeciwnym razie nic nie stoi na przeszkodzie, aby przeciwnik podszywał się pod nadawcę wiadomości). W tym miejscu będzie dziemy nadal rozważać ustawienie klucza prywatnego, w którym strony mają ten sam tajny klucz.¹

Składnia kodu uwierzytelniającego wiadomość

Zanim formalnie zdefiniujemy bezpieczeństwo kodu uwierzytelniającego wiadomość, najpierw zdefiniujemy, czym jest MAC i jak jest używany. Dwóch użytkowników, którzy chcą komunikować się w sposób uwierzytelniony, zaczyna od wygenerowania i udostępnienia tajnego klucza przed rozpoczęciem komunikacji. Kiedy jedna ze stron chce wysłać wiadomość m do drugiej, oblicza znacznik MAC (lub po prostu znacznik) t w oparciu o wiadomość m i klucz współdzielony, a następnie wysyła wiadomość m i znacznik t do drugiej strony. Znacznik jest obliczany przy użyciu algorytmu generowania znaczników opisanego przez firmę Mac; zatem, formułując to, co już powiedzieliśmy, nadawca wiadomości m oblicza $t = Mac(m)$ i przesyła (m, t) do odbiorcy. Po otrzymaniu (m, t)

¹ W omówionym wcześniej przykładzie pliku cookie, sprzedawca (w rzeczywistości) komunikuje się „ze sobą” z użytkownikiem, który pełni rolę kanału komunikacji. W tym ustawieniu sam serwer musi znać klucz, ponieważ działa zarówno jako nadawca, jak i odbiorca.

druga strona sprawdza, czy t jest prawidłowym znacznikiem w wiadomości m (w odniesieniu do klucza współdzielonego), czy nie. Odbywa się to poprzez uruchomienie algorytmu weryfikacji Vrfy , który jako dane wejściowe przyjmuje klucz współdzielony, a także komunikat m i znacznik t oraz wskazuje, czy dany znacznik jest ważny. Formalnie:

DEFINICJA 4.1 Kod uwierzytelniający wiadomość (lub MAC) składa się z trzech probabilistycznych algorytmów czasu wielomianowego (Gen, Mac, Vrfy), takich jak:

1. Algorytm generowania klucza Gen przyjmuje jako dane wejściowe parametr bezpieczeństwa 1^n i wyprowadza klucz k z $|k| = n$.
2. Algorytm generowania znaczników Mac przyjmuje jako dane wejściowe klucz k i wiadomość $m \in \{0, 1\}^n$ i wyprowadza znacznik t. Ponieważ algorytm ten może być losowy, zapisujemy to jako $t = \text{Mack}(m)$.
3. Deterministyczny algorytm weryfikacji Vrfy przyjmuje jako dane wejściowe klucz k, wiadomość m i znacznik t. Wyprowadza bit b, gdzie $b = 1$ oznacza prawidłowe , a $b = 0$ oznacza nieprawidłowe. Zapisujemy to jako $b := \text{Vrfy}(m, t)$.

Wymagane jest, aby dla każdego n, każdego klucza k wyprowadzonego przez $\text{Gen}(1^n)$ i każdego $m \in \{0, 1\}^n$ utrzymywało się , że $\text{Vrfy}(m, \text{Mack}(m)) = 1$.

Jeżeli istnieje taka funkcja, że dla każdego k wyjścia $\text{Gen}(1^n)$ algorytmu, to nazywamy schemat Mack jest zdefiniowany tylko dla wiadomości $m \in \{0, 1\}^n$, a MAC o stałej długości dla wiadomości o długości (n).

Podobnie jak w przypadku szyfrowania kluczem prywatnym, $\text{Gen}(1^n)$ prawie zawsze po prostu wybiera klucz jednolity $k \in \{0, 1\}^n$ i w tym przypadku pomijamy Gen.

Weryfikacja kanoniczna. W przypadku deterministycznych kodów uwierzytelniania wiadomości (to znaczy, gdy Mac jest algorymem deterministycznym), kanonicznym sposobem przeprowadzenia weryfikacji jest po prostu ponowne obliczenie znacznika i sprawdzenie równości. Innymi słowy, $\text{Vrfy}(m, t)$ najpierw oblicza $t' := \text{Mack}(m)$, a następnie wyprowadza 1 wtedy i tylko wtedy, gdy $t' = t$. Jednak nawet w przypadku deterministycznych adresów MAC przydatne jest zdefiniowanie osobnego algorytmu Vrfy , aby wyraźnie rozróżnić semantykę uwierzytelniania wiadomości od weryfikacji jej autentyczności.

Bezpieczeństwo kodów uwierzytelniających wiadomości

Zdefiniujmy teraz domyślne pojęcie bezpieczeństwa dla kodów uwierzytelniających wiadomości. Intuicyjna idea stojąca za definicją jest taka, że żaden skuteczny przeciwnik nie powinien być w stanie wygenerować prawidłowego tagu dla jakiegokolwiek „nowej” wiadomości, która nie została wcześniej wysłana (i uwierzytelniona) przez jedną z komunikujących się stron.

Jak w przypadku każdej definicji bezpieczeństwa, aby sformalizować to pojęcie, musimy zdefiniować zarówno siłę przeciwnika, jak i to, co należy uznać za „przerwę ”.

Jak zwykle, rozważamy tylko probabilistycznych przeciwników w czasie wielomianowym²

²W sekcji 4.6 omówiono uwierzytelnianie wiadomości w oparciu o teorię informacyjną, gdzie na przeciwnika nie są nakładane żadne ograniczenia obliczeniowe.

zatem prawdziwym pytaniem jest, w jaki sposób modelujemy interakcję przeciwnika ze stronami komunikującymi się. Ustawiając uwierzytelnianie wiadomości, przeciwnik obserwujący komunikację pomiędzy dwiema stronami może zobaczyć wszystkie wiadomości wysłane przez te strony wraz z odpowiadającymi im tagami MAC.

Przeciwnik może także mieć możliwość wpływania na treść tych komunikatów, bezpośrednio lub pośrednio (jeżeli np. zewnątrz działań przeciwnika wpływają na komunikaty wysyłane przez strony). Dotyczy to na przykład wcześniejszego przykładu pliku cookie sieciowego, w którym własne działania użytkownika wpływają na zawartość pliku cookie przechowywanego na jego komputerze.

Aby formalnie zamodelować powyższe, pozwalamy przeciwnikowi żądać tagów MAC dla dowolnych wiadomości. Formalnie dajemy przeciwnikowi dostęp do wyroczni MAC $\text{Mack}(\cdot)$; przeciwnik może wielokrotnie przesyłać do tej wyroczni dowolną wybraną przez siebie wiadomość, a w zamian otrzymuje znacznik $t = \text{Mack}(m)$. (W przypadku adresu MAC o stałej długości można przesyłać tylko wiadomości o prawidłowej długości.)

Uznamy to za „przerwanie” schematu, jeśli przeciwnik jest w stanie wysłać dowolną wiadomość m wraz ze znacznikiem t w taki sposób, że: (1) t jest prawidłowym znacznikiem w wiadomości m (tj. $\text{Vrfy}(m, t) = 1$) oraz (2) przeciwnik nie zażądał wcześniej znacznika MAC w wiadomości m (tj. od swojej wyroczni). Pierwszy warunek oznacza, że gdyby przeciwnik wysłał (m, t) do jednej z uczciwych stron, ta strona ta zostałaby błędem oszukana, myśląc, że m pochodzi od legalnej strony, ponieważ $\text{Vrfy}(m, t) = 1$. drugi warunek jest wymagany, ponieważ przeciwnik zawsze może po prostu skopiować wiadomość i znacznik MAC, które zostały wcześniej wysłane przez jedną z uprawnionych stron (i oczywiście zostaną one zaakceptowane jako ważne). Taki atak polegający na powtórzeniu nie jest uważany za „łamanie” kodu uwierzytelniającego wiadomość. Nie oznacza to, że ataki typu „replay” nie stanowią problemu bezpieczeństwa; są i będą dziennie mieli wiele cej do powiedzenia na ich temat poniżej.

Mówiąc, że adres MAC spełniający określony powyżej poziom bezpieczeństwa jest egzystencjalnie niemożliwy do podrobienia w przypadku ataku adaptacyjnego z wybraną wiadomością. „Egzistencjalna niepodrabialność” odnosi się do faktu, że przeciwnik nie może być w stanie sfalszować prawidłowego znacznika w żadnej wiadomości, a „adaptacyjny atak wybraną wiadomością” odnosi się do faktu, że przeciwnik jest w stanie uzyskać znaczniki MAC w dowolnych wiadomościach wybranych adaptacyjnie podczas jego ataku.

W kierunku formalnej definicji rozważ następujący eksperyment dla kodu uwierzytelniającego wiadomość $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$, przeciwnika A i wartości n dla parametru bezpieczeństwa:

Eksperyment z uwierzytelnianiem wiadomości $\text{Mac-forgeA}, \Pi(n)$: 1. Klucz k jest

generowany poprzez uruchomienie $\text{Gen}(1n)$. n i dostęp p

2. Przeciwnik A otrzymuje dane wejściowe Oracle do $\text{Mack}(\cdot)$.

1. Przeciwnik ostatecznie wyprowadza (m, t) . Niech Q oznacza zbiór wszystkich zapytań, które A zadał swojej wyroczni.

3. A powiedzie się, wtedy i tylko wtedy, gdy (1) $\text{Vrfy}(m, t) = 1$ i (2) $m \in Q$.

W takim przypadku wynik eksperymentu definiuje się jako 1.

MAC jest bezpieczny, jeśli żaden skuteczny przeciwnik nie może odnieść sukcesu w powyższym doświadczeniu z nieistotnym prawdopodobieństwem:

DEFINICJA 4.2 Kod uwierzytelniający wiadomość $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ jest egzystencjalnie niemożliwy do podrobienia w wyniku adaptacyjnego ataku wybranej wiadomości lub po prostu zabezpieczony, jeśli dla wszystkich probabilistycznych przeciwników A w czasie wielomianowym istnieje pomijalna funkcja zaniedbania taka, że :

$$\Pr[\text{Mac-forgeA}, \Pi(n) = 1] \leq \text{negl}(n).$$

Czy definicja jest zbyt mocna? Powyższa definicja jest dość mocna pod dwoma względami. Po pierwsze, przeciwnik może zażądać znaczników MAC dla dowolnych wybranych przez siebie wiadomości. Po drugie, uważa się, że przeciwnik „złamał” schemat, jeśli może wysłać prawidłowy znacznik do dowolnej wcześniej nieuwierzytelnionej wiadomości. Można by zarzucić, że oba te elementy definicji są nierealistyczne i zbyt mocne: w przypadku używania adresu MAC w „prawdziwym świecie” uczciwe strony uwierzytelniają jedynie „znaczące” wiadomości (nad którymi przeciwnik może mieć jedynie ograniczoną kontrolę), a podobnie należy uznać to za naruszenie bezpieczeństwa tylko wtedy, gdy przeciwnik może sfałszować prawidłowy znacznik w „znaczącej” wiadomości.

Dlaczego nie dostosować definicji, aby to uchwycić?

Kluczową kwestią jest to, że to, co stanowi znaczący komunikat, zależy całkowicie od aplikacji. Podczas gdy niektóre aplikacje MAC mogą uwierzytelniać tylko wiadomości tekstowe w języku angielskim, inne aplikacje mogą uwierzytelić pliki arkuszy kalkulacyjnych, inne wpisy w bazach danych i inne surowe dane. Można również projektować protokoły, w których uwierzytelniany będzie dowolny przedmiot — w rzeczywistości niektóre protokoły uwierzytelniania jednostek dokładnie to robią. Dzięki możliwości najsielniejszej definicji zabezpieczeń adresów MAC zapewniamy szerokie zastosowanie bezpiecznych adresów MAC do szerokiego zakresu celów, bez konieczności martwienia się o zgodność adresu MAC z semantyką aplikacji.

Powtórz ataki. Podkreślamy, że powyższa definicja i same kody uwierzytelniające wiadomości nie zapewniają ochrony przed atakami polegającymi na ponownym odtwarzaniu, w ramach których wcześniej wysłana wiadomość (wraz ze znacznikiem MAC) jest odtwarzana uczciwie stronie. Niemniej jednak ataki powtórki stanowią poważny problem! Rozważmy ponownie scenariusz, w którym użytkownik (powiedzmy Alicja) wysyła do swojego banku prośbę o przelanie 1000 dolarów z jej konta innemu użytkownikowi (powiedzmy Bobowi). Robiąc to, Alicja może obliczyć znacznik MAC i dodać go do żądania, aby bank wiedział, że żądanie jest autentyczne. Jeśli adres MAC jest bezpieczny, Bob nie będzie w stanie przechwycić żądania i zmienić kwoty na 10 000 USD, ponieważ wiążałoby się to ze sfałszowaniem prawidłowego tagu w wcześniej nieuwierzytelnionej wiadomości. Nic jednak nie stoi na przeszkodzie, aby Bob przechwycił wiadomość Alicji i odtworzył ją dziesięć ciekotnie w banku. Jeśli bank zaakceptuje każdą z tych wiadomości, efektem końcowym będzie to, że na konto Boba zostanie przelane 10 000 dolarów zamiast żądanego 1000 dolarów.

Pomimo realnego zagrożenia, jakie stanowią ataki typu „replay”, MAC sam w sobie nie jest w stanie chronić przed takimi atakami, ponieważ definicja MAC (Definicja 4.1)

nie uwzględnia żadnego pojęcia stanu w algorytmie weryfikacji (więc z każdym razem, gdy algorytmowi weryfikacji zostanie przedstawiona ważna para (m, t) , zawsze wyświetli się 1). Przeciwnie, ochrona przed atakami typu „replay” – jeśli taka ochrona w ogóle jest konieczna – musi być obsługiwana przez jakąś aplikację wyższego poziomu. Powodem, dla którego definicja MAC jest skonstruowana w ten sposób, jest to, że po raz kolejny nie chcemy zakładać żadnej semantyki dotyczącej aplikacji korzystających z MAC; w szczególności decyzja, czy odtworzona wiadomość powinna być traktowana jako „ważna”, może zależeć od aplikacji.

Dwie popularne techniki zapobiegania atakom polegającym na powtórzeniu polegają na użyciu numerów sekwencyjnych (znanych również jako liczniki) lub znaczników czasu. Pierwsze podejście, opisane (w bardziej ogólnym kontekście) w sekcji 4.5.3, wymaga od komunikujących się użytkowników utrzymywania (zsynchronizowanego) stanu i może być problematyczne, gdy użytkownicy komunikują się poprzez kanał stratny, w którym komunikaty są czasami odrzucane (choć problem ten można złagodzić). W drugim podejściu, korzystając ze znaczników czasu, nadawca dołącza do wiadomości bieżący czas T (powiedzmy z dokładnością do milisekundy) przed uwierzytelnieniem i wysyła T wraz z wiadomością i wynikowym znacznikiem t . Kiedy odbiornik uzyskuje T, m, t , sprawdza, czy t jest ważnym znacznikiem w T i czy T mieści się w pewnym akceptowalnym odchyleniu zegara od bieżącego czasu T w odbiorniku. Metoda ta ma również pewne wady, w tym konieczność utrzymywania przez nadawcę i odbiorcę ścisłe zsynchronizowanych zegarów oraz możliwość, że atak polegający na powtórzeniu może nadal mieć miejsce, jeśli zostanie przeprowadzony wystarczająco szybko (w szczególności w akceptowalnym oknie czasowym).

Mocne MAC. Zgodnie z definicją, bezpieczny adres MAC gwarantuje, że przeciwnik nie będzie w stanie wygenerować prawidłowego znacznika dla nowej wiadomości, która nigdy wcześniej nie została uwierzytelioniona. Nie wyklucza to jednak możliwości, że atakujący będzie w stanie wygenerować nowy tag na wcześniej uwierzytelionowej wiadomości. Oznacza to, że MAC gwarantuje, że jeśli atakujący pozna tagi t_1, \dots na wiadomości m_1, \dots , to nie będzie w stanie sfałszować prawidłowego znacznika t w żadnej wiadomości $m \in \{m_1, \dots\}$. Jednakże może się zdarzyć, że przeciwnik „sfałszuje” inny ważny znacznik $t = t_1$ w wiadomości m_1 . Ogólnie rzecz biorąc, tego typu wrogie zachowanie nie stanowi problemu.

1

Niemniej jednak w niektórych sytuacjach przydatne jest rozważenie mocniejszej definicji bezpieczeństwa komputerów MAC, w których takie zachowanie jest wykluczone.

Formalnie rozważamy zmodyfikowany eksperyment Mac-forge, który jest zdefiniowany dokładnie tak samo jak Mac-forge, z tą różnicą, że teraz zbiór Q zawiera pary zapytań Oracle i powiązane z nimi odpowiedzi. (To znaczy $(m, t) \in Q$, jeśli A zapytał Mack(m) i otrzymał w odpowiedzi znacznik t .) Przeciwnikowi A udaje się (a eksperyment Mac-forge ocenia na 1) wtedy i tylko wtedy, gdy A wyprowadza (m, t) takie, że $\text{Vrfyk}(m, t) = 1$ i $(m, t) \notin Q$.

DEFINICJA 4.3 Kod uwierzytelniający wiadomość $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ jest silnie bezpieczny lub silny MAC, jeśli dla wszystkich probabilistycznych przeciwników A w czasie wielomianowym istnieje pomijalna funkcja negl taka, że:

$$\Pr[\text{Mac-sforge}_A, \Pi(n) = 1] = \text{negl}(n).$$

Nietrudno zauważyc, że jeśli bezpieczny adres MAC korzysta z weryfikacji kanonicznej, to jest również silnie bezpieczny. Jest to ważne, ponieważ wszystkie adresy MAC w świecie rzeczywistym korzystają z weryfikacji kanonicznej. Dowód poniższego dowodu pozostawiamy jako ćwiczenie.

TWIERDZENIE 4.4 Niech $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ będzie bezpiecznym MAC korzystającym z weryfikacji kanonicznej. Wtedy Π jest silnym MAC.

Zapytania weryfikacyjne

Definicje 4.2 i 4.3 dają przeciwnikowi dostęp p do wyroczni MAC, co odpowiada przeciwnikowi w świecie rzeczywistym, który może wpłynąć na uczciwego nadawcę, aby wygenerował znacznik dla jakiejś wiadomości, m.in. Można również rozważyć przeciwnika, który wchodzi w interakcję z uczciwym odbiorcą, wysyłając m do odbiorcy, aby dowiedzieć się, czy $\text{Vrfy}(m, t) = 1$. Takiego przeciwnika można schwytać formalnie w sposób naturalny, dając dostęp p przeciwnikowi z powyższych definicji również do wyroczni weryfikacyjnej.

Definicja obejmująca w ten sposób wyrocznię weryfikacyjną jest być może „właściwym” sposobem zdefiniowania bezpieczeństwa kodów uwierzytelniających wiadomości. Okazuje się jednak, że dla MAC stosujących weryfikację kanoniczną nie ma to znaczenia: każdy taki MAC spełniający Definicję 4.2 spełnia również wariant definicyjny, w którym dozwolone są zapytania weryfikacyjne. Podobnie każdy silny adres MAC automatycznie pozostaje bezpieczny nawet w otoczeniu, w którym możliwe są zapytania weryfikacyjne. (W rzeczywistości stanowi to jedną z motywacji do zdefiniowania silnego bezpieczeństwa adresów MAC). Jednak w przypadku komputerów MAC, które nie korzystają z weryfikacji kanonicznej, zezwolenie na zapytania weryfikacyjne może mieć znaczenie; patrz ćwiczenia 4.2 i 4.3. Ponieważ większość MAC omówionych w tej książce (a także MAC stosowanych w praktyce) wykorzystuje weryfikację kanoniczną, używamy tradycyjnych definicji, które pomijają dostęp p do wyroczni weryfikacyjnej.

Potencjalny atak w czasie. Kwestią nieroziązaną w powyższym tekście jest możliwość przeprowadzenia ataku czasowego na weryfikację adresu MAC. Rozważamy tutaj przeciwnika, który może wysłać do odbiorcy parę komunikat/tag – w ten sposób wykorzystując odbiorcę jako wyrocznię weryfikacyjną – i dowiadujemy się nie tylko, czy odbiorca akceptuje, czy odrzuca, ale także ile czasu zajmuje odbiorca podjęcie tej decyzji. Pokazujemy, że jeśli taki atak jest możliwy, to naturalne wdrożenie weryfikacji MAC prowadzi do łatwej do wykorzystania lukii.

(Zauważ, że w naszych zwykłych kryptograficznych definicjach bezpieczeństwa osoba atakująca poznaje jedynie dane wyjściowe wyroczni, do których ma dostęp p , i nic więcej. Opisywany tutaj atak, będący przykładem ataku bocznego kanału, pokazuje, że pewne rzeczywiste -ataki na świecie nie są ujęte w zwykłych definicjach.)

Konkretnie założymy, że MAC korzysta z weryfikacji kanonicznej. Aby zweryfikować znacznik t w wiadomości m , odbiorca oblicza $t := \text{Mack}(m)$, a następnie porównuje t z t , otrzymując na wyjściu 1 wtedy i tylko wtedy, gdy t i t są równe. Założymy, że to porównanie jest zaimplementowane przy użyciu standardowej procedury (takiej jak `strcmp` w C), która porównuje t i t po jednym bajcie na raz i odrzuca, gdy tylko pierwszy nierówny bajt zostanie

napotkane. Należy zauważać, że przy takiej implementacji czas odrzucenia różni się w zależności od pozycji pierwszego nierównego bajtu.

Możliwe jest wykorzystanie tej pozornie nieistotnej informacji do sfałszowania znacznika dowolnej wiadomości, m.in. Podstawowa idea jest następująca: powiedzmy, że atakujący zna pierwsze i bajty prawidłowego znacznika m. (Na początku $i = 0$) Atakujący poznaje kolejny bajt prawidłowego tagu, wysyłając $(m, t_0), \dots, (m, t_{255})$ do odbiornika, gdzie tj jest ciągiem znaków z pierwszymi i bajtami ustawionymi poprawnie, $(i + 1)$ -szy bajt równy j (w formacie szesnastkowym), a pozostałe bajty ustawione na 0x00. Wszystkie te tagi prawdopodobnie zostaną odrzucone (jeśli nie, atakującemu i tak się to uda); jednakże dla dokładnie jednego z tych znaczników pierwszy $(i + 1)$ bajt będzie pasował do prawidłowego znacznika i odrzucenie zajmie nieco więcej czasu niż w przypadku pozostałych. Jeśli tj jest znacznikiem, który spowodował, że odrzucenie trwało najdłużej, atakujący dowiaduje się, że $(i + 1)$ -szy bajt prawidłowego znacznika to j. W ten sposób atakujący poznaje każdy bajt prawidłowego tagu za pomocą maksymalnie 256 zapытаń do wyczerpania weryfikacyjnej. W przypadku znacznika 16-bajtowego atak ten w najgorszym przypadku wymaga jedynie 4096 zapытаń.

Można się zastanawiać, czy taki atak jest realistyczny, gdyż wymaga dostępu do wyroczni weryfikacyjnej oraz możliwości zmierzenia różnicy w czasie potrzebnym na porównanie i vs $i + 1$ bajt. W rzeczywistości dokładnie takie ataki zostały przeprowadzone na rzeczywiste systemy! Przykładowo, adresy MAC były używane do weryfikowania aktualizacji kodu na konsoli Xbox 360, a zastosowana tam implementacja weryfikacji adresów MAC powodowała różnicę między czasami odrzucenia wynoszącą 2,2 milisekundy. Atakujący mogli to wykorzystać i załadować pirackie gry na sprzęt t.

Na podstawie powyższego dochodzimy do wniosku, że weryfikacja MAC powinna uwzględnić czas niezależne porównanie ciągów, które zawsze porównuje wszystkie bajty.

4.3 Konstruowanie bezpiecznych kodów uwierzytelniających wiadomości

4.3.1 Adres MAC o stałej długości

Funkcje pseudolosowe są naturalnym narzędziem do konstruowania bezpiecznych kodów uwierzytelniających wiadomości. Intuicyjnie, jeśli znacznik MAC zostanie uzyskany poprzez zastosowanie funkcji pseudolosowej do wiadomości m, wówczas sfałszowanie znacznika na wcześniejszej nieuwierzytelnionej wiadomości wymaga od przeciwnika prawidłowego odgadnięcia wartości funkcji pseudolosowej w „nowym” punkcie wejściowym. Prawdopodobieństwo odgadnięcia wartości funkcji losowej w nowym punkcie wynosi 2^{-n} (jeśli długość wyjściowa funkcji wynosi n). Prawdopodobieństwo odgadnięcia tej wartości dla funkcji pseudolosowej może być tylko nieznacznie większe.

Powyższy pomysł, pokazany w Konstrukcji 4.5, sprawdza się przy konstruowaniu bezpiecznego adresu MAC o stałej długości dla wiadomości o długości n (ponieważ nasze funkcje pseudolosowe domyślnie mają n-bitową długość bloku). Jest to przydatne, ale nie spełnia naszego celu. W sekcji 4.3.2 pokazujemy, jak rozszerzyć to, aby obsługiwać komunikaty o dowolnej długości. Bardziej wydajne konstrukcje adresów MAC dla wiadomości o dowolnej długości omówimy w sekcjach 4.4 i 5.3.2.

KONSTRUKCJA 4.5

Niech F będzie funkcją pseudolosową. Zdefiniuj adres MAC o stałej długości dla wiadomości o długości n w następujący sposób:

- Mac: na wejściu klawisz $k \in \{0, 1\}^n$ i wiadomość $m \in \{0, 1\}^n$, wyprowadzają znacznik $t := F_k(m)$. (Jeśli $|m| = |k|$ to nic nie wypisz.)
- Vrfy: na wejściu klawisz $k \in \{0, 1\}^n$ wiadomość $m \in \{0, 1\}^n$, wyjście, i a znacznik $t \in \{0, 1\}^n$ wtedy i tylko wtedy, gdy $t \stackrel{?}{=} F_k(m)$. (Jeśli $|m| = |k|$, to wyprowadź 0 .)

MAC o stałej długości z dowolnej funkcji pseudolosowej.

TWIERDZENIE 4.6 Jeżeli F jest funkcją pseudolosową, to Konstrukcja 4.5 jest bezpiecznym MAC o stałej długości dla komunikatów o długości n .

DOWÓD Podobnie jak w przypadku poprzednich zastosowań funkcji pseudolosowych, dowód ten opiera się na paradygmacie, w którym najpierw analizuje się bezpieczeństwo schematu przy użyciu funkcji prawdziwie losowej, a następnie rozważa się wynik zastąpienia funkcji prawdziwie losowej funkcją pseudolosową.

Niech A będzie probabilistycznym przeciwnikiem wielomianowym w czasie. Rozważ kod uwierzytelnienia wiadomości $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$, który jest taki sam jak $\Pi = (\text{Mac}, \text{Vrfy})$ w Construction 4.5, z tą różnicą, że zamiast funkcji pseudolosowej F_k użyto prawdziwie losowej funkcji f . Oznacza to, że $\text{Gen}(1^n)$ wybiera jednolitą funkcję $f \in \text{Func}_n$, a Mac oblicza znacznik tak samo jak Mac , z tą różnicą, że zamiast F_k używa się f . To jest natychmiastowe

$$\Pr[\text{Mac-forgeA}, \Pi(n) = 1] \quad 2 \quad \text{-rzedz} \quad (4.1)$$

ponieważ dla dowolnej wiadomości $m \in \{0, 1\}^n$ wartość $t = f(m)$ rozkłada się równomiernie w $\{0, 1\}^n$ z punktu widzenia przeciwnika A .

Następnie pokazemy, że istnieje funkcja pomijalna taka, że

$$\Pr[\text{Mac-forgeA}, \Pi(n) = 1] = \Pr[\text{Mac-forgeA}, \Pi(n) = 1] \quad \text{negl}(n); \quad (4.2)$$

w połączeniu z równaniem (4.1) pokazuje to

$$\Pr[\text{Mac-forgeA}, \Pi(n) = 1] = 2^{-\text{rzedz}} + \text{zaniebywanie}(n),$$

udowodnienie twierdzenia.

Aby udowodnić równanie (4.2), konstruujemy wielomianowy wyróżnik D , który ma dostęp p do jakiejś funkcji w Oracle i którego celem jest ustalenie, czy ta funkcja jest pseudolosowa (tj. równa F_k dla uniforma $k \in \{0, 1\}^n$ lub losowy (tj. równy f dla jednorodnego $f \in \text{Func}_n$). Aby to zrobić, D emuluje eksperyment uwierzytelniania wiadomości dla A i obserwuje, czy A pomyślnie wysła prawidłowy znacznik do „nowej” wiadomości. Jeśli tak, D zgaduje, że jego wyrocznia jest funkcją pseudolosową; w przeciwnym razie D zgaduje, że jego wyrocznia jest funkcją losową. Szczegółowo:

Wyróżnik D:

D ma dane wejściowe 1^n i dostęp p do wyroczni $O : \{0, 1\}^n \rightarrow \{0, 1\}^n$ i działa w następujący sposób:

1. Uruchom A(1^n). Ilekroć A pyta swoją wyrocznię MAC w wiadomości m (tzn.

ilekroć A żąda znacznika w wiadomości m), odpowiedz na to zapytanie w następny sposób: Zapytaj O za

pomocą m i uzyskaj odpowiedź t; zwróć t do A.

2. Kiedy na końcu wykonywania A wyprowadza (m, t), wykonaj: (a)

Zapytanie O za pomocą m i uzyskaj odpowiedź t^* . (b)

Jeśli $t^* = t$ i (2) A nigdy nie odpytywał swojej wyroczni MAC na m, wówczas na wyjściu jest 1; w przeciwnym razie wyjście 0.

Jest oczywiste, że D przebiega w czasie wielomianowym.

Zauważ, że jeśli wyrocznia D jest funkcją pseudolosową, to widok A, gdy jest uruchamiany jako podprogram przez D, rozkłada się identycznie jak widok A w eksperymencie Mac-forgeA, $\Pi(n)$. Co więcej, D wyprowadza 1 dokładnie wtedy, gdy $\text{Mac-forgeA}, \Pi(n) = 1$. Zatem

$$\Pr[D \text{ Fk}(\cdot)(1^n) = 1] = \Pr[\text{Mac-forgeA}, \Pi(n) = 1]$$

gdzie $k \in \{0, 1\}^n$ jest wybrane jednolicie w powyższym. Jeśli wyrocznia D jest funkcją losową, to widok A, gdy jest uruchamiany jako podprogram przez D, rozkłada się identycznie jak widok A w eksperymencie Mac-forgeA, $\Pi(n)$ i ponownie D wyprowadza 1 dokładnie wtedy, gdy $\text{Mac-forgeA}, \Pi(n) = 1$. Zatem,

$$\Pr[D \text{ f}(\cdot)(1^n) = 1] = \Pr[\text{Mac-forgeA}, \Pi(n) = 1]$$

gdzie f Func jest wybierane równomiernie.

Ponieważ F jest funkcją pseudolosową, a D przebiega w czasie wielomianowym, istnieje pomijalna funkcja negl taka, że

$$\Pr[DFk(\cdot)(1^n) = 1] = \Pr[Df(\cdot)(1^n) = 1] = \text{negl}(n).$$

Implikuje to Równanie (4.2), kończące dowód twierdzenia. ■

4.3.2 Rozszerzenie domeny dla komputerów MAC

Konstrukcja 4.5 jest ważna, ponieważ pokazuje ogólny paradygmat konstruowania bezpiecznych kodów uwierzytelniających wiadomości na podstawie funkcji pseudolosowych. Niestety, konstrukcja jest w stanie obsłużyć jedynie wiadomości o stałej długości, które w dodatku są raczej krótkie.³ Ograniczenia te są nie do zaakceptowania.

³ Biorąc pod uwagę funkcję pseudolosową pobierającą dane wejściowe o dowolnej długości, Konstrukcja 4.5 dałaby bezpieczny adres MAC dla wiadomości o dowolnej długości. Podobnie funkcja pseudolosowa

w wiele kroki zastosowań. Pokazujemy tutaj, jak ogólny adres MAC obsługujący komunikaty o dowolnej długości może zostać zbudowany z dowolnego adresu MAC o stałej długości dla komunikatów o długości n. Konstrukcja, którą pokazujemy, nie jest zbyt wydajna i raczej nie znajdzie zastosowania w praktyce. Rzeczywiście, znane są znacznie wydajniejsze konstrukcje bezpiecznych adresów MAC, co omawiamy w sekcjach 4.4 i 5.3.2. Uwzglę dniemy niniejszą konstrukcję ze względu na jej prostotę i ogólność.

Niech $\Pi = (\text{Mac}, \text{Vrfy})$ będzie bezpiecznym MAC o stałej długości dla wiadomości o długości n. Przed przedstawieniem konstrukcji MAC dla wiadomości o dowolnej długości opartej na Π wykluczamy kilka prostych pomysłów i opisujemy niektóre kanoniczne ataki, którym należy zapobiegać. Poniżej analizujemy wiadomość m do uwierzytelnienia jako sekwencję bloków m_1, \dots, m_d ; zauważ, że ponieważ naszym celem jest obsługa komunikatów o dowolnej długości, d może się różnić w zależności od komunikatu.

1. Naturalnym pierwszym pomysłem jest po prostu uwierzytelnienie każdego bloku osobno, tj. obliczenie $ti := \text{Mac}_k(m_i)$ dla wszystkich i i wyprowadzenie t_1, \dots, t_d jako znacznik.

Uniemożliwia to przeciwnikowi wysłanie wcześniej nieuwierzytelnionego bloku bez wykrycia. Nie zapobiega to jednak atakowi polegającemu na zmianie kolejności bloków, podczas którego osoba atakująca tasuje kolejność bloków w uwierzytelnionej wiadomości. W szczególności, jeśli t_1, t_2 jest prawidłowym znacznikiem w wiadomości m_1, m_2 (przy $m_1 = m_2$), to t_2, t_1 jest prawidłowym znacznikiem w (innej) wiadomości m_2, m_1 (coś, co nie jest dozwolone przez definicję 4.2).

2. Możemy zapobiec poprzedniemu atakowi, uwierzytelniając indeks bloku (imi) dla td jako wraz z każdym blokiem. Oznacza to, że teraz obliczamy $ti = \text{Mac}_k(imi)$ znacznika. (Będą musiały ulec zmianie.) (Zauważ, że długość bloku $|mi|$ all i oraz wyjście t_1, \dots, t_d nie zapobiega atakowi obcięcia, w wyniku którego osoba atakująca po prostu usuwa bloki z końca wiadomości (i usuwa odpowiadające im bloki znacznika)).

3. Atak obcięcia można udaremnić dodatkowo uwierzytelniając długość wiadomości wraz z każdym blokiem. (Uwierzytelnianie długości wiadomości w oddzielnym bloku nie działa. Czy rozumiesz dlaczego?) Oznacza to, że oblicz $ti = \text{Mac}(imi)$ dla wszystkich i, gdzie oznacza długość wiadomości w bitach. (Po raz kolejny długość bloku $|mi|$ będzie musiała zostać zmniejszona.) Ten schemat jest podatny na atak typu „mix-and-match”, w którym przeciwnik łączy bloki z różnych wiadomości. Na przykład, jeśli przeciwnik uzyska tagi t_1, \dots, t_d i t

$1, \dots, t_d$ odpowiednio w
 m_1, \dots, m_d i $m = mm = \dots, m^m$ wiadomościach d może wyprowadzić znacznik t_1, t_4, \dots, t_{d+1} na wiadomość $m_1, m_2, m_3, \dots, m_d, m$

Możemy zapobiec temu ostatniemu atakowi, dodając losowy „identyfikator wiadomości” do każdego bloku, który zapobiega łączeniu bloków z różnych wiadomości. To prowadzi nas do Konstrukcji 4.7.

z wiele kroków zapewni bezpieczny adres MAC dla dłuższych wiadomości. Jednakże istniejące praktyczne funkcje pseudolosowe (tj. szyfry blokowe) wymagają krótkich danych wejściowych o stałej długości.

KONSTRUKCJA 4.7

Niech $\Pi = (\text{Mac}, \text{Vrfy})$ będzie adresem MAC o stałej długości dla komunikatów o długości n . Zdefiniuj MAC w następujący sposób:

- Mac : na wejściu klucz $k \in \{0, 1\}^n$ i wiadomość $m \in \{0, 1\}^n$ o (niezerowej) długości $< 2 n/4$, rozłoż m na d bloków m_1, \dots, m_d , każdy o długości $n/4$. (W razie potrzeby ostatni blok jest dopełniany zerami.) Wybierz jednolity identyfikator $r \in \{0, 1\}^{n/4}$

Dla $i = 1, \dots, d$, oblicz $t_i = \text{Mac}_k(rimi)$, gdzie i , są kodowane jako ciągi znaków o długości $n/4$.[†] Wypisz znacznik $t := r, t_1, \dots, t_d$. • Vrfy : na wejściu klucz $k \in \{0, 1\}^n$ wiadomość $m \in \{0, 1\}^n$ o długości $< 2 n/4$ i znacznik $t = r, t_1, \dots, t_d$, rozłoż m na d bloki m_1, \dots, m_d , każdy o długości $n/4$. (W razie potrzeby ostatni blok jest dopełniany zerami.) Wynik 1 wtedy i tylko wtedy, gdy $d = d$ i Vrfy

$$k(rimi, t_i) = 1 \text{ dla } 1 \leq i \leq d.$$

[†]Należy pamiętać, że i można zakodować przy użyciu $n/4$ bitów, ponieważ $i < 2 n/4$.

Adres MAC dla wiadomości o dowolnej długości z dowolnego adresu MAC o stałej długości.

(Technicznie rzecz biorąc, schemat obsługuje tylko wiadomości o długości mniejszej niż $2n/4$. Asymptotycznie, ponieważ jest to granica wykładnicza, uczciwe strony nie będą uwierzytelnić wiadomości tak długich, a żaden przeciwnik działający w czasie wielomianowym nie byłby w stanie przesyłać wiadomości tak długich do swojej wyroczni MAC. W praktyce, gdy konkretna wartość n jest ustalona, należy upewnić się, że ta granica jest akceptowalna.)

TWIERDZENIE 4.8 Jeżeli Π jest bezpiecznym MAC o stałej długości dla komunikatów o długości n , to Konstrukcja 4.7 jest bezpiecznym MAC (dla komunikatów o dowolnej długości).

DOWÓD Intuicja jest taka, że dopóki Π jest bezpieczne, przeciwnik nie może wprowadzić nowego bloku z prawidłowym znacznikiem. Co więcej, dodatkowe informacje zawarte w każdym bloku zapobiegają różnym atakom (upuszczanie bloków, zmiana kolejności bloków itp.) opisanych wcześniej. Udoskonalimy bezpieczeństwo, pokazując zasadniczo, że takie ataki są jedynymi możliwymi.

Niech Π będzie MAC podanym w Konstrukcji 4.7 i niech A będzie probabilistycznym przeciwnikiem w czasie wielomianowym. Pokazujemy, że $\Pr[\text{Mac-forgeA}, \Pi(n) = 1]$ jest pomijalnie małe. Najpierw wprowadzimy notację, która będzie używana w dowodzie. Niech Powtórz oznacza zdarzenie, w którym ten sam losowy identyfikator pojawi się w dwóch znacznikach zwróconych przez wyrocznię MAC w eksperymencie $\text{Mac-forgeA}, \Pi(n)$. Let-ting ($m, t = r, t_1, \dots$) oznacza końcowy wynik A, gdzie $m = m_1, \dots, m_d$ ma długość $n/4$, pozwalamy NewBlock na zdarzenie, w którym co najmniej jeden z bloków rimi nigdy wcześniej nie został uwierzytelny przez komputer Mac w trakcie odpowiadania na pytania A na Macu. (Zauważ, że dzięki konstrukcji Π łatwo jest dokładnie określić, które bloki są uwierzytelniane przez komputer Mac podczas obliczania $\text{Mac}(m)$.) Nieformalnie NewBlock to zdarzenie, w którym A próbuje wprowadzić prawidłowy znacznik na bloku, który został

nigdy nie uwierzytelniany przez podstawowy adres MAC o stałej długości Π .

Mamy

$$\begin{aligned} \Pr[\text{Mac-forgeA}, \Pi(n) = 1] &= \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \quad \text{Powtórz}] \\ &\quad + \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \quad \text{Powtórz} \quad \overline{\text{NowyBlok}}] \\ &\quad + \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \quad \text{Powtórz} \quad \overline{\text{NowyBlok}} \quad \overline{\text{NowyBlok}}] \\ &= \Pr[\text{Powtórz}] (4.3) \\ &\quad + \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \quad \text{NewBlock}] \\ &\quad + \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \quad \text{Powtórz} \quad \overline{\text{NowyBlok}} \quad \overline{\text{NowyBlok}}]. \end{aligned}$$

Pokazujemy, że pierwsze dwa wyrazy równania (4.3) są nieistotne, a ostatni wyraz wynosi 0.

Oznacza to, że $\Pr[\text{Mac-forgeA}, \Pi(n) = 1]$ jest nieistotne, zgodnie z oczekiwaniami.

Twierdzenie 4.9 $\Pr[\text{Powtórz}]$ jest nieistotne.

DOWÓD Niech $q(n)$ będzie liczbą zapytań wyroczni MAC zadanych przez A. Aby odpowiedzieć na i-te zapytanie wyroczni A, wyrocznia wybiera r_i równomiernie ze zbioru rozmiaru $2n/4$. Prawdopodobieństwo zdarzenia Powtórzenie jest dokładnie tym samym prawdopodobieństwem, że $r_i = r_j$ dla pewnego $i = j$. Stosując „ograniczenie urodzinowe” (Lemat A.15), otrzymujemy, że $\Pr[\text{Repeat}] = 2n/4$. Ponieważ A wykonuje tylko wielomianowo wiele zapytań, wartość ta jest nieistotna. ■

Następnie rozważmy ostatni wyraz po prawej stronie równania (4.3).

Twierdzimy, że jeśli $\text{Mac-forgeA}, \Pi(n) = 1$, ale Powtórzenie nie wystąpiło, to musi być tak, że wystąpił NewBlock. Oznacza to, że $\text{Mac-forgeA}, \Pi(n) = 1 \quad \text{Powtórzenie} \implies \text{NowyBlok}$ i tak

$$\Pr[\text{Mac-forgeA}, \Pi(n) = 1 \quad \text{Powtórz} \quad \overline{\text{NowyBlok}}] = 0.$$

To jest w pewnym sensie sedno dowodu.

Ponownie niech $q = q(n)$ oznacza liczbę zapytań MAC Oracle wykonanych przez A i niech r_i oznacza losowy identyfikator użyty do odpowiedzi na i-te zapytanie Oracle A. Jeśli Powtórzenie nie nastąpi, wówczas wartości r_1, \dots, r_q są różne. Niech $(m, t = r, t_1, \dots)$ będzie wyjściem A, gdzie $m = m_1, \dots$. Jeśli $r \in \{r_1, \dots, r_q\}$, wtedy wyraźnie pojawia się NewBlock. Jeśli nie, to $r = r_j$ dla jakiegoś unikalnego j i bloki $r_1 m_1, \dots$ w takim przypadku nie byłoby możliwe uwierzytelnienie w trakcie odpowiadania na zapytania Mac inne niż j-te takie zapytanie. Niech $m(j)$ będzie wiadomością użytą przez A w j-tym zapytaniu Oracle i niech będzie jej długością. Należy rozważyć dwa przypadki:

J

Przypadek 1: Wszystkie bloki uwierzytelnione podczas odpowiadania na j-te zapytanie Maca mają na drugiej pozycji. Zatem w szczególności $r_1 m_1$ nigdy nie został uwierzytelniony w trakcie odpowiadania na j-te zapytanie Maca i następnie pojawi NewBlock.

Przypadek 2: = J. Jeśli $\text{Mac-forgeA}, \Pi(n) = 1$, to musimy mieć $m = m(j)$. Pozwalać , . . .
 $m(j) = m$ Ponieważ m i $m(j)$ mają równą długość, musi istnieć at
 co najmniej jeden indeks i , dla którego $m = \overset{(j)}{m} i$. Blok rmi nigdy nie został następ pnie
 uwierzytelniony w trakcie odpowiadania na j -te zapytanie Maca . (Ponieważ i znajduje
 się na trzeciej pozycji bloku, blok rmi (j) mógłby zostać uwierzytelniony tylko wtedy,
 gdyby $rmi = r_{j+1}(j)$.) i
 ale to nie jest prawdą, ponieważ $m = m$

Aby zakończyć dowód twierdzenia, związałmy drugi wyraz z
 prawa strona równania (4.3):

Twierdzenie 4.10 $\Pr[\text{Mac-forgeA}, \Pi(n) = 1 \mid \text{NewBlock}]$ jest nieistotne.

Roszczenie opiera się na zabezpieczeniu Π . Konstruujemy przeciwnika ppt A, który
 atakuje adres MAC Π o stałej długości i z prawdopodobieństwem udaje mu się wyprowadzić
 prawidłowe fałszerstwo na wcześniej nieuwierzytelnionej wiadomości

$$\Pr[\text{Mac-forgeA}, \Pi(n) = 1] = \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \mid \text{NewBlock}]. \quad (4.4)$$

Bezpieczeństwo Π oznacza, że lewa strona jest znikoma, co potwierdza twierdzenie.

Konstrukcja A jest oczywista, dlatego opiszemy ją pokrótko.

A uruchamia A jako podprogram i odpowiada na żądanie A dotyczące znacznika na m ,
 wybierając samo $r \in \{0, 1\}^{n/4}$, odpowiednio analizując m i wykonując niezbę dne zapytania
 do własnej wyroczni MAC $\text{Mac}(\cdot)$. Kiedy A wyprowadza $(m, t = r, t_1, \dots)$, to A sprawdza, czy
 występuje NewBlock (jest to łatwe do zrobienia, ponieważ A może śledzić wszystkie
 zapytania, które kieruje do własnej wyroczni). Jeśli tak, to A znajduje pierwszy blok rmi , który
 nigdy wcześniej nie był uwierzytelniony przez Maca i wyprowadza (rmi, t_i) . (Jeśli nie, A nie
 wyświetla niczego.)

Widok A, gdy jest wykonywany jako podprogram przez A, rozkłada się identycznie jak
 widok A w eksperymencie $\text{Mac-forgeA}, \Pi(n)$, a zatem prawdopodobieństwa zdarzeń Mac-
 $\text{forgeA}, \Pi(n) = 1$ i NewBlock nie zmieniają. Jeśli wystąpi NewBlock, wówczas A wysyła blok rmi ,
 który nigdy wcześniej nie był uwierzytelniony przez jego własną wyrocznię MAC; jeśli Mac-
 $\text{forgeA}, \Pi(n) = 1$, to znacznik w każdym bloku jest ważny (w odniesieniu do Π), a więc w
 szczególności jest to prawdą dla wyjścia bloku przez A. Oznacza to, że ilekroć wystąpią Mac-
 $\text{forgeA}, \Pi(n) = 1$ i NewBlock, mamy $\text{Mac-forgeA}, \Pi(n) = 1$, co potwierdza równanie (4.4).

4.4 CBC-MAC

Twierdzenia 4.6 i 4.8 pokazują, że możliwe jest skonstruowanie bezpiecznego kodu
 uwierzytelniającego wiadomość dla wiadomości o dowolnej długości z pseudolosowego

funkcja pobierająca dane wejściowe o stałej długości n . To pokazuje w zasadzie, że bezpieczne adresy MAC można zbudować z szyfrów blokowych. Niestety, wynikająca z tego konstrukcja jest wyjątkowo nieefektywna: aby obliczyć znacznik w wiadomości o długości dn , szyfr blokowy jest oceniany $4d$ razy; znacznik ma długość wiele krotną niż $4dn$ bitów. Na szczęście dostępne są znacznie wydajniejsze konstrukcje. Badamy tutaj jedną taką konstrukcję, która opiera się wyłącznie na szyfrach blokowych, a drugą w sekcji 5.3.2, która wykorzystuje dodatkowy prymityw kryptograficzny.

4.4.1 Podstawowa konstrukcja

CBC-MAC to ujednolicony kod uwierzytelniający wiadomości, szeroko stosowany w praktyce. Podstawowa wersja CBC-MAC, bezpieczna przy uwierzytelnianiu wiadomości o dowolnej ustalonej długości, jest podana jako Konstrukcja 4.11. (Patrz także Rysunek 4.1.) Ostrzegamy, że ten podstawowy schemat nie jest bezpieczny w ogólnym przypadku, gdy mogą być uwierzytelniane wiadomości o różnej długości; zobacz dalszą dyskusję poniżej.

KONSTRUKCJA 4.11

Niech F będzie funkcją pseudolosową i ustali funkcję $\{0, 1\}^n \rightarrow \{0, 1\}^n$ o długości > 0 . Podstawowa konstrukcja CBC-MAC jest następująca:

- Mac: po wprowadzeniu klucza $k \in \{0, 1\}^n$ i wiadomości m o długości $(n) \cdot n$ wykonaj następujące czynności (ustawiamy $= (n)$ w następstwie poniższego sposobu):
 1. Przeanalizuj m jako $m = m_1, \dots, m_n$ gdzie każde m_i ma długość n .
 2. Ustaw $t_0 := 0^n$. Następnie dla $i = 1$ do n : Ustaw $t_i := F_k(t_{i-1} \oplus m_i)$.
 Wyprowadź t jako znacznik.
- Vrfy: na wejściu klawisz $k \in \{0, 1\}^n$, wiadomość m i znacznik t , wykonaj: Jeśli m nie ma długości $(n) \cdot n$, wówczas wypisz 0. W przeciwnym razie wyprowadź 1 wtedy i tylko wtedy, gdy $t = Mack(m)$.

Podstawowy CBC-MAC (dla wiadomości o stałej długości).

TWIERDZENIE 4.12 Niech F będzie funkcją pseudolosową. Jeżeli F jest funkcją pseudolosową, to Konstrukcja 4.11 jest bezpiecznym MAC dla komunikatów o długości $(n) \cdot n$.

Dowód twierdzenia 4.12 jest nieco skomplikowany. W następstwie sekcji udowodnimy bardziej ogólny wynik, z którego wynika powyższe twierdzenie.

Chociaż Konstrukcję 4.11 można w oczywisty sposób rozszerzyć na obsługę wiadomości, których długość jest dowolną wielokrotnością n , konstrukcja jest bezpieczna tylko wtedy, gdy długość uwierzytelnianych wiadomości jest ustalona i wcześniej uzgodniona przez nadawcę i odbiorcę. (Patrz ćwiczenie 4.13.)

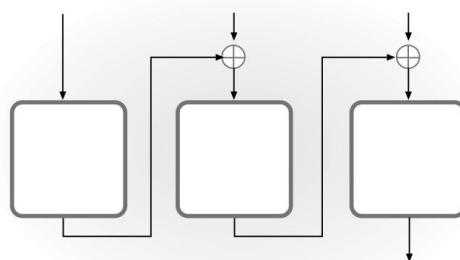
Zaletą tej konstrukcji w porównaniu z Konstrukcją 4.5, która również zapewnia adres MAC o stałej długości, jest to, że obecna konstrukcja może uwierzytelnić dłuższe wiadomości. W porównaniu z Konstrukcją 4.7, CBC-MAC jest znacznie bardziej wydajny, wymagając jedynie d oceny szyfru blokowego dla wiadomości o długości dn i tylko ze znacznikiem o długości n.

Szyfrowanie w trybie CBC-MAC a szyfrowanie w trybie CBC. Tryb CBC-MAC jest podobny do trybu działania CBC. Istnieje jednak kilka istotnych różnic:

1. Szyfrowanie w trybie CBC wykorzystuje losowe IV, co ma kluczowe znaczenie dla bezpieczeństwa. Natomiast CBC-MAC nie wykorzystuje IV (alternatywnie można to postrzegać jako użycie stałej wartości $IV = 0n$), co również jest kluczowe dla bezpieczeństwa. W szczególności CBC-MAC używający losowej kroplówki nie jest bezpieczny.
2. W przypadku szyfrowania w trybie CBC wszystkie wartości pośrednie ti (zwane ci w przypadku szyfrowania w trybie CBC) są wyprowadzane przez algorytm szyfrujący jako część tekstu saszyfrowanego, podczas gdy w CBC-MAC jako znacznik wyprowadzany jest tylko ostatni blok. Jeśli CBC-MAC zostanie zmodyfikowany tak, aby wyświetlał wszystkie {ti} uzyskane w trakcie obliczeń, wówczas nie jest już bezpieczne.

W ćwiczeniu 4.14 zostaniesz poproszony o sprawdzenie, czy omówione powyżej modyfikacje CBC-MAC są niebezpieczne. Przykłady te ilustrują fakt, że nieszkodliwie wyglądające modyfikacje konstrukcji kryptograficznych mogą sprawić, że będą one niebezpieczne. Należy zawsze wdrażać konstrukcję kryptograficzną dokładnie zgodnie ze specyfikacją i nie wprowadzać żadnych odmian (chyba że można udowodnić, że same odmiany są bezpieczne). Ponadto istotne jest zrozumienie zastosowanej konstrukcji. W wielu przypadkach biblioteka kryptograficzna zapewnia programiście „funkcję CBC”, ale nie rozróżnia wykorzystania tej funkcji do szyfrowania lub uwierzytelniania wiadomości.

Bezpieczne CBC-MAC dla wiadomości o dowolnej długości. Pokrótko opisujemy dwa sposoby modyfikacji Construction 4.11, w możliwy do sprawdzenia, bezpieczny sposób, w celu obsługi komunikatów o dowolnej długości. (Tutaj dla uproszczenia zakładamy, że wszystkie uwierzytelniane wiadomości mają długość stanowiącą wielokrotność n i że Vrfy je odrzuca



RYSUNEK 4.1: Podstawowy CBC-MAC (dla wiadomości o stałej długości).

dowolna wiadomość, której długość nie jest wielokrotnością n. W następnej sekcji zajmiemy się bardziej ogólnym przypadkiem, w którym wiadomości mogą mieć dowolną długość.)

1. Do wiadomości m dołącz jej długość $|m|$ (zakodowany jako ciąg n-bitowy), a następnie obliczyć podstawowy CBC-MAC na wyniku; patrz rysunek 4.2. Bezpieczeństwo tego wariantu wynika z wyników udowodnionych w kolejnym podrozdziale.
- Zauważ, że dodanie $|m|$ do końca wiadomości i wtedy obliczenie podstawowego CBC-MAC nie jest bezpieczne.
2. Zmień schemat tak, aby generowanie klucza wybierało dwa niezależne, jednolite klucze $k_1 \in \{0, 1\}^n$ i $k_2 \in \{0, 1\}^n$. Następnie, aby uwierzytelnić wiadomość m, najpierw oblicz podstawowy CBC-MAC m, używając k_1 i niech t będzie wynikiem; wypisz znacznik $t^* := F_{k_2}(t)$.

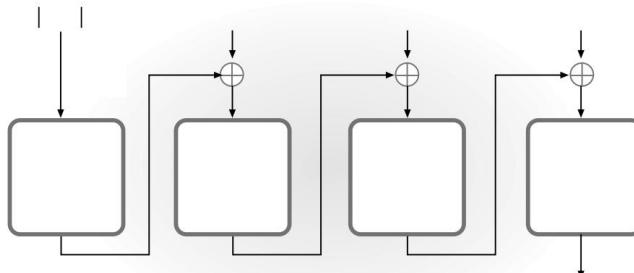
Druga opcja ma tę zaletę, że nie trzeba wcześniej znać długości wiadomości (tj. na początku obliczania znacznika). Ma jednak tę wadę, że używa dwóch kluczy dla F. Należy zauważać, że kosztem dwóch dodatkowych zastosowań funkcji pseudolosowej można zapisać pojedynczy klucz k, a następnie wprowadzić klucze $k_1 := F(k)$ i $k_2 := F(k)$ na początku obliczeń. Mimo to w praktyce operację inicjalizacji klucza dla szyfru blokowego uważa się za stosunkowo kosztowną. Dlatego wymaganie dwóch różnych kluczy —nawet jeśli są one wprowadzane na bieżąco— jest mniej pożądane.

4.4.2 *Dowód bezpieczeństwa

W tej sekcji udowadniamy bezpieczeństwo różnych wariantów CBC-MAC. Zaczniemy od podsumowania wyników, a następnie podamy szczegóły dowodu.

Zanim zaczniemy, zauważmy, że dowód w tej sekcji jest dość skomplikowany i jest przeznaczony dla zaawansowanych czytelników.

W tej sekcji napraw funkcję F z kluczem, która dla parametru bezpieczeństwa n odwzorowuje n-bitowe klucze i n-bitowe wejścia na n-bitowe wyjścia. Definiujemy klucz



RYSUNEK 4.2: Wariant bezpiecznego protokołu CBC-MAC do uwierzytelniania wiadomości o dowolnej długości.

funkcja CBC , która dla parametru bezpieczeństwa n odwzorowuje n-bitowe klucze i dane wejściowe w ($\{0, 1\}^n$) (tj. ciągi znaków, których długość jest wielokrotnością n) na n-bitowe wyjścia. Funkcja ta jest zdefiniowana jako

$$\text{CBCk}(x_1, \dots, x_n) \stackrel{\text{def}}{=} F_k(F_k(\dots F_k(F_k(x_1) \quad x_2) \quad \dots) \quad x_n),$$

gdzie $|x_1| = \dots = |x_n| = |k| = r$. (Pozostawiamy CBCk niezdefiniowane w pustym łańcuchu.) Zauważ, że CBC jest dokładnie podstawowym CBC-MAC, chociaż tutaj rozważamy dane wejściowe o różnej długości.

Zbiór ciągów $P \subseteq \{0, 1\}^n$ jest wolny od prefiksu, jeśli nie zawiera pustego łańcucha i żaden ciąg $X \in P$ nie jest przedrostkiem dowolnego innego ciągu $X' \in P$. Pokazujemy:

TWIERDZENIE 4.13 Jeżeli F jest funkcją pseudolosową, to CBC jest funkcją pseudolosową, o ile zbiór wejść, na którym jest ona odpytywana, jest wolny od prefiksów. Formalnie dla wszystkich probabilistycznych wyróżników wielomianowych D, które wysyłają zapytania do swojej wyciązni na zestawie danych wejściowych bez przedrostków, istnieje pomijalne zaniedbywanie funkcji takie, że

$$\Pr[\text{DCBCk}(\cdot)(1^n) = 1] = \Pr[Df(\cdot)(1^n) = 1] = \text{negl}(n),$$

gdzie k jest wybrane jednolicie spośród $\{0, 1\}^n$ i f jest wybrane równomiernie ze zbioru funkcji odwzorowujących $\{0, 1\}^n$ na $\{0, 1\}^n$ (tj. wartość f na każdym wejściu jest jednolita i niezależnie od wartości f na wszystkich pozostałych wejściach).

W ten sposób możemy przekonwertować funkcję pseudolosową F dla danych wejściowych o stałej długości na funkcję pseudolosową CBC dla danych wejściowych o dowolnej długości (z zastrzeżeniem ograniczenia dotyczącego tego, jakie dane wejściowe mogą być odpytywane)! Aby użyć tego do uwierzytelniania wiadomości, dostosowujemy ideę Konstrukcji 4.5 w następujący sposób: aby uwierzytelnić wiadomość m, najpierw zastosuj funkcję kodowania encode , aby uzyskać (niepusty)串 znaków encode(m) $\subseteq \{0, 1\}^{2n}$; następnie wpisz znacznik CBCk(encode(m)). Aby było to bezpieczne (por. dowód twierdzenia 4.6), kodowanie musi być wolne od przedrostków, a mianowicie posiadać tę właściwość, że dla dowolnych odrębnych (legalnych) wiadomości m1, m2串 znaków encode(m1) nie jest przedrostek encode(m2). Oznacza to, że dla dowolnego zestawu (legalnych) wiadomości $\{m_1, \dots\}$, zbiór zakodowanych wiadomości $\{encode(m_1), \dots\}$ nie zawiera prefiksów.

Przeanalizujmy teraz dwa konkretne zastosowania tego pomysłu:

- Naprawić mechanizm zabezpieczenia komunikatów o dowolnej długości ($0, 1\}^{2n}$). Następnie możemy zastosować tradycyjne kodowanie encode(m) = m , które nie zawiera prefiksów, ponieważ jeden串 nie może być przedrostkiem innego串 o tej samej długości. Jest to dokładnie podstawowy CBC-MAC i z tego, co powiedzieliśmy powyżej wynika, że podstawowy CBC-MAC jest bezpieczny dla wiadomości o dowolnej ustalonej długości (por. Twierdzenie 4.12). • Jednym ze sposobów obsługi komunikatów o dowolnej długości (niepustych) (technicznie rzecz biorąc, komunikatów o długości mniejszej niż $2n$) jest zakodowanie串 $m \in \{0, 1\}^n$ poprzez dodanie jego długości $|m|$ (zakodowany jako串 n-bitowy), a następnie dodając tyle zer, ile potrzeba, aby uzyskać długość wynikowego

ciąg bę dący wielokrotnością n. (Zasadniczo pokazano to na rysunku 4.2.)

To kodowanie nie zawiera prefiksów, dzięki czemu uzyskujemy bezpieczny adres MAC dla wiadomości o dowolnej długości.

Pozostała część tej sekcji poświęcona jest dowodowi twierdzenia 4.13. Dowodząc twierdzenia, analizujemy CBC, gdy jest ono „wpisane” za pomocą funkcji losowej g, a nie losowego klucza k dla jakiejś podstawowej funkcji pseudolosowej F.

Oznacza to, że rozważamy funkcję kluczową CBCg zdefiniowaną jako

$$\text{CBCg}(x_1, \dots, x_n) \stackrel{\text{def}}{=} g(g(\dots g(g(x_1) \oplus x_2) \dots) \oplus x_n)$$

gdzie dla parametru bezpieczeństwa n funkcja g odwzorowuje n-bitowe wejścia na n-bitowe wyjścia, oraz $|x_1| = \dots = |x_n| = rz$. Należy zauważyć, że CBCg zdefiniowane tutaj nie jest efektywne (ponieważ reprezentacja g wymaga przestrzeni wykładniczej w n); niemniej jednak jest to nadal dobrze zdefiniowana funkcja z kluczem.

Pokazujemy, że jeśli g jest wybrane równomiernie z Funcn, wówczas CBCg jest nie do odróżnienia od losowego odwzorowania funkcji $\{(0, 1)^n\}^n$, gdy do ciągów n-bitowych, o ile odpisywany jest zestaw danych wejściowych bez przedrostków. Dokładniej:

Twierdzenie 4.14 Napraw dowolne $n \geq 1$. Dla wszystkich wyróżników D, które odpytują swoją wyrocznię o zestaw danych wejściowych q bez prefiksów, gdzie najdłuższe takie wejście zawiera bloki, obowiązuje to:

$$\Pr[\text{DCBCg}(\cdot)(1^n) = 1] = \Pr[Df(\cdot)(1^n) = 1] = \frac{2^{2^{n-1}}}{2^{rz}},$$

gdzie g jest wybrane jednolicie z Funcn, a f jest wybrane jednolicie ze zbioru funkcji odwzorowujących $\{(0, 1)^n\}^n$ na $\{0, 1\}^n$.

(Twierdzenie jest bezwarunkowe i nie nakłada żadnych ograniczeń na czas działania D. Zatem możemy przyjąć, że D jest deterministyczne.) Z powyższego wynika Twierdzenie 4.13 wykorzystujące standardowe techniki, które już widzieliśmy.

W szczególności dla dowolnego D działającego w czasie wielomianowym musimy mieć $q(n), (n) = \text{poli}(n)$, więc $c q(n)$ jest $\text{poli}(n)^2$: 2^{-rzcz}

DOWÓD (twierdzenia 4.14) Ustal pewne $n \geq 1$. Dowód przebiega w dwóch etapach: Najpierw definiujemy pojęcie gładkości i udowadniamy, że CBC jest gładkie; następnie pokazujemy, że gładkość implikuje twierdzenie.

Niech $P = \{X_1, \dots, X_q\}$ bę dzie wolnym od przedrostków zbiorem q wejść, gdzie każde X_i znajduje się w $\{0, 1\}^n$, a najdłuższy ciąg w P zawiera bloki (tzn. każdy $X_i \in P$ zawiera co najwyżej bloki o długości n). Zauważ, że dla dowolnego $t_1, \dots, t_q \in \{0, 1\}^n$ utrzymuje się, że $\Pr[\exists i : f(X_i) = t_i] = 2^{-nq}$, gdzie prawdopodobieństwo jest ponad równomiernym wyborem funkcji f ze zbioru odwzorowań funkcji $\{0, 1\}^n$ do $\{0, 1\}^n$. Mówimy, że CBC jest (q, n, δ) -gładkie, jeśli dla każdego zbiuru bez przedrostków $P = \{X_1, \dots, X_q\}$ jak wyżej i co $t_1, \dots, t_q \in \{0, 1\}^n$, to utrzymuje

$$\Pr[\exists i : \text{CBCg}(X_i) = t_i] = (1 - \delta) \cdot 2^{-nq},$$

gdzie prawdopodobieństwo jest ponad jednolitym wyborem g z Funcn.

Inaczej mówiąc, CBC jest gładkie, jeśli dla każdego ustalonego zbioru par wejście/wyjście $\{(X_i, t_i)\}$, gdzie $\{X_i\}$ tworzą zbiór bez przedrostków, prawdopodobieństwo, że $CBCg(X_i) = t_i$ dla wszystkich i wynosi δ -bliskie prawdopodobieństwo, że $f(X_i) = t_i$ dla wszystkich i (gdzie g jest funkcją losową od $\{0, 1\}^n$ do $\{0, 1\}^n$, a f jest funkcją losową od $\{0, 1\}^n$ do $\{0, 1\}^n$).

Twierdzenie 4.15 $CBCg$ jest (q, δ) -gładkie, dla $\delta = q$

2.2 · 2 - rz.

DOWÓD Dla dowolnego $X \in \{0, 1\}^n$ z $X = x_1, \dots, x_m$ oraz $x_i \in \{0, 1\}^n$, niech $Cg(X)$ oznacza zbiór wejść, na podstawie których g jest oceniane podczas obliczania $CBCg(X)$; tj. jeśli $X \in \{0, 1\}^n$

M Następnie

$$Cg(X) \stackrel{\text{def}}{=} (x_1, CBCg(x_1), x_2, \dots, CBCg(x_1, \dots, x_{m-1}), x_m).$$

Dla $X \in \{0, 1\}^n$ i $X \in \{0, 1\}^n$ z $Cg(X) = (I_1, \dots, I_m)$, $i Cg(X) = (I)$, powiedzmy, że istnieje nietrywialną kolizję w X , jeśli $I_i = I_j$ dla i trochę $i = j$, i powiedzieć, że istnieje nietrywialna kolizja pomiędzy X i X , jeśli (w tym drugim przypadku i może być równe j). We $I_i = I_j$ jeśli ale $(x_1, \dots, x_i) = (x_1, \dots, x_j)$ mówimy, że w P zachodzi nietrywialna kolizja, zachodzi nietrywialna kolizja w jakimś $X \in P$ lub pomiędzy jakąś parą strun $X, X \in P$. Niech Coll bęź dzie zdarzeniem, że istnieje nietrywialna kolizja w P .

Twierdzenie udowodnimy w dwóch etapach. Najpierw pokazujemy, że pod warunkiem, że w P nie zachodzi nietrywialna kolizja, prawdopodobieństwo, że $CBCg(X_i) = t_i$ dla wszystkich i wynosi dokładnie 2^{-nq} . Następnie pokazujemy, że prawdopodobieństwo wystąpienia nietrywialnej kolizji w P jest mniejsze niż $\delta = q$.

Rozważmy wybór jednolitego g , wybierając kolejno jednolite wartości wyjść g na różnych wejściach. Ustalenie, czy istnieje nietrywialna kolizja pomiędzy dwoma ciągami $X, X \in P$, można przeprowadzić wybierając najpierw wartości $g(I_1)$ i $g(I_1) = I_1$, te wartości są takie same), a następnie wybierając wartości $g(I_2)$ i $g(I_2)$ (zauważ, że $I_2 = g(I_1) \neq x_2$ i $I_1 = g(I_1) \neq x_2$ są zdefiniowane raz $g(I_1)$, $g(I_1)$ dopóki nie wybierzemy wartości dla $g(I_m-1)$ i $g(I_m)$ od $g(I_1)$, $g(I_m)$ nietrywialna kolizja pomiędzy X_1 została naprawiona) i kontynuujemy w ten sposób i X . Kontynuując ten tok rozumowania, można ustalić \mathbb{C}_{n-1} . W szczególności nie trzeba czy Coll wybierać wartości w celu ustalenia, czy istnieje występuje, wybierając wartości g we wszystkich wpisach oprócz końcowych każdego z $Cg(X_1), \dots, Cg(X_q)$.

Założymy, że Coll nie wystąpił po ustaleniu wartości g na różnych wejściach, jak opisano powyżej. Rozważmy końcowe wpisy w każdym z $Cg(X_1), \dots, Cg(X_q)$.

Wszystkie te wpisy są różne (wynika to bezpośrednio z faktu, że Coll nie wystąpiło) i twierdzimy, że wartość g w każdym z tych punktów nie została jeszcze ustalona. Rzeczywiście, jedyny sposób, w jaki wartość g mogłaby już zostać ustalona w którymkolwiek z tych punktów, polega na tym, że końcowy wpis I_m jakiegoś $Cg(X)$ jest równy niekońcowemu wpisowi I_j jakiegoś $Cg(X)$. Ale ponieważ Coll nie miał miejsca, może to nastąpić

ma miejsce tylko wtedy, gdy $X = X_1 \dots X_m$. Ale wtedy X by to zrobił, bę dzie przedrostkiem \bar{X} ruszając założenie, że P jest wolne od przedrostków.

Ponieważ g jest funkcją losową, powyższe oznacza, że $CBCg(X_1), \dots, CBCg(X_q)$ są jednolite i niezależne od siebie, podobnie jak wszystkie inne wartości g , które zostały już ustalone. (Dzieje się tak, ponieważ $CBCg(X_i)$ jest wartością g ocenianą na ostatnim wejściu $Cg(X_i)$, wartością wejściową, która różni się od wszystkich pozostałych wejść, dla których g zostało już ustalone.) Zatem dla dowolnego $t_1, \dots, t_q \in \{0, 1\}^n$ mamy:

$$\Pr[i : CBCg(X_i) = t_i | Coll = 2] = \frac{Q}{2^n}. \quad (4.5)$$

Nastę pnie pokazujemy, że $Coll$ wystę puje z dużym prawdopodobieństwem poprzez górne ograniczenie $\Pr[Coll]$. Dla różnych $X_i, X_j \in P$, niech $Colli,j$ oznacza zdarzenie, w którym zachodzi nietrywialna kolizja w X lub X lub nietrywialna kolizja pomiędzy X_i i X_j .

Mamy $Coll = \bigcup_{i,j} Colli,j$, związek połączony daje

$$\Pr[Kolumna] = \Pr[\bigcup_{i,j: i < j} Colli,j] = \frac{Q}{2} \cdot \Pr[Colli,j] = \frac{Q^2}{2} \cdot \Pr[Colli,j]. \quad (4.6)$$

Naprawiając odrę bne $X = X_i$ i $X = X_j$ w P , teraz ograniczamy $Colli,j$. Jak wynika z analizy, prawdopodobieństwo jest maksymalne, gdy oba X_i i X_j są tak długie, jak to możliwe, a zatem zakładamy, że każdy z nich ma długość bloków. Niech $X = (x_1, \dots, x_t) \in X = (xx)$ i niech t bę dzie najwi ką liczbą całkowitą taką, że $(x_1, \dots, x_t) = (x_1, \dots, t)$. (Zauważ, że $t < \text{lub } X = X$.) Zakładamy $t > 0$, ale poniższą analizę można łatwo zmodyfikować, dając ten sam wynik, jeśli $t = 0$. W dalszym ciągu pozwalamy I_1, I_2, \dots (odpowiednio I_2, \dots) oznaczają dane wejściowe do g podczas obliczania $CBCg(X)$ (odpowiednio $CBCg(X)$); zauważ, że $I = (I_1, \dots, I_t)$. Rozważ wybór g , wybierając jednolite wartości dla wyjść g , jeden po drugim. Robimy to w 2-2 krokach w nastę pujący sposób:

$1, \dots, \overset{To}{_}$

Kroki od 1 do $t - 1$ (jeśli $t > 1$): W każdym kroku i wybierz jednolitą wartość dla (które są równe).

$g(I_i)$, definiując w ten sposób I_{i+1} i I_{i+1}

Krok t : Wybierz jednolitą wartość $g(I_t)$, definiując w ten sposób I_{t+1} i I_{t+1}

Kroki $t + 1$ do -1 (jeśli $t < -1$): Wybierz po kolej jednako wartości dla każdego $g(I_{t+1}), g(I_{t+2}), \dots, g(I_{-1})$, definiując w ten sposób $I_{t+2}, I_{t+3}, \dots, I$.

Kroki do -2 ($t < -1$): Wybierz z kolej jednako wartości dla każdego z , definiując w ten sposób $I_{t+1}, g(I_{t+2}), \dots, g(I_{-2})$, I_{t+1} sposob $I_{t+2}, I_{t+3}, \dots, I$.

Niech $Coll(k)$ bę dzie zdarzeniem, w którym w kroku k nastąpi nietrywialna kolizja. Nastę pnie

$$\Pr[Colli,j] = \Pr[\bigcup_{k=2}^{t-1} Coll(k)] = \Pr[Coll(1)] + \Pr[\bigcup_{k=2}^{t-1} Coll(k) | Coll(k-1)], \quad (4.7)$$

korzystając z Twierdzenia A.9. Dla $k < t$ stwierdzamy $\Pr[Coll(k) | Coll(k-1)] = k/2^n$; rzeczywiście, jeśli w kroku $k - 1$ nie wystąpiła żadna nietrywialna kolizja, wartość

$g(I_k)$ wybiera się równomiernie w etapie k ; nietrywialna kolizja ma miejsce tylko wtedy, gdy zdarzy się, że $I_{k+1} = g(I_k)$ i x_{k+1} jest równe jednemu z $\{I_1, \dots, I_k\}$ (wszystkie są różne, ponieważ $\text{Coll}(k-1)$ nie wystąpiło). Z podobnego rozumowania mamy $\Pr[\text{Coll}(t) | \text{Coll}(t-1)]$

$2t/2^n$ (tutaj należy wziąć pod uwagę dwie wartości $I+1, I$; zauważ, że nie mogą być one sobie równe). Wreszcie, argumentując jak poprzednio, dla $k > t$ mamy $\Pr[\text{Coll}(k) | \text{Coll}(k-1)] = (k+1)/2^n$. Korzystając z równania (4.7) mamy zatem

$$\begin{aligned} \Pr[\text{Coll}_{i,j}] &= 2^{-r_{\text{recz}}} \cdot \sum_{k=1}^{t-1} (k + 2t + (k+1)) \cdot \Pr[\text{Coll}(k) | \text{Coll}(k-1)] \\ &= 2^{-r_{\text{recz}}} \cdot \sum_{k=2}^{2t-1} k \cdot (2t+1) \cdot (-1) < 2^{-r_{\text{recz}}} \cdot 2 \cdot 2^{-r_{\text{recz}}}. \end{aligned}$$

Z równania (4.6) otrzymujemy $\Pr[\text{Coll}] < q^2 2^{-r_{\text{recz}}} = \delta$. Na koniec, używając równania widzimy, że

$$\Pr[i : \text{CBC}_g(X_i) = t_i] = \Pr[i : \text{CBC}_g(X_i) = t_i | \text{Coll}] \cdot \Pr[\text{Coll}] = \frac{1}{2^n} \cdot \Pr[\text{Coll}] = (1 - \delta) \cdot 2^{-r_{\text{recz}}},$$

jak twierdzono. ■

Pokażemy teraz, że gładkość implikuje twierdzenie. Założymy bez utraty ogólności, że D zawsze tworzy q (odpowiedź) zapytania, z których każde zawiera co najwyżej bloki. D może wybierać swoje zapytania adaptacyjnie (tj. w zależności od odpowiedzi na poprzednie zapytania), ale zbiór zapytań D musi być wolny od prefiksów.

Dla różnych $X_1, \dots, X_q \in \{0, 1\}^n$ i dowolne $t_1, \dots, t_q \in \{0, 1\}^n$, zdefiniuj $\alpha(X_1, \dots, X_q; t_1, \dots, t_q)$ na 1 wtedy i tylko wtedy, gdy D wyrowadza 1 podczas wykonywania zapytań X_1, \dots, X_q i uzyskanie odpowiedzi t_1, \dots, t_q . (Jeśli, powiedzmy, D nie uczyni zapytania X_1 pierwszym zapytaniem, wówczas $\alpha(X_1, \dots, X_q; t_1, \dots, t_q) = 0$.) Zakładając, że $X = (X_1, \dots, X_q)$ i $t = (t_1, \dots, t_q)$, wówczas mamy

$$\begin{aligned} \Pr[D \text{ CBC}_g(\cdot)(1^n) = 1] &= \alpha(X, t) \cdot \Pr[i : \text{CBC}_g(X_i) = t_i] \\ &\quad \text{X bez prefiku; T} \\ &= \alpha(X, t) \cdot (1 - \delta) \cdot \Pr[i : f(X_i) = t_i] \\ &\quad \text{X bez prefiku; T} \\ &= (1 - \delta) \cdot \Pr[Df(\cdot)(1^n) = 1], \end{aligned}$$

gdzie powyżej g jest wybrane jednolicie z Func_n , a f jest wybrane równomiernie ze zbioru funkcji odwzorowujących $\{0, 1\}^n$ na $\{0, 1\}^n$. To implikuje $\Pr[Df(\cdot)(1^n) = 1]$

$$\Pr[D \text{ CBC}_g(\cdot)(1^n) = 1] = \delta \cdot \Pr[Df(\cdot)(1^n) = 1] = \delta.$$

Argument symetryczny, gdy D daje wartość 0, kończy dowód. ■

4.5 Uwierzytelnione szyfrowanie

W rozdziale 3 zbadaliśmy, w jaki sposób można uzyskać poufność ustawień klucza prywatnego za pomocą szyfrowania. W tym rozdziale pokazaliśmy, jak zapewnić integralność za pomocą kodów uwierzytelniających wiadomości. Można oczywiście chcieć osiągnąć oba cele jednocześnie i to jest problem, do którego się teraz zajmiemy.

Najlepszą praktyką jest zawsze domyślne zapewnienie poufności i integralności w ustawieniu klucza prywatnego. Rzeczywiście, w wielu zastosowaniach, w których wymagana jest tajemnica, okazuje się, że integralność jest również niezbędna. Co więcej, brak uczciwości może czasami prowadzić do naruszenia tajemnicy.

4.5.1 Definicje

Zaczynamy jak zwykle od dokładnego zdefiniowania tego, co chcemy osiągnąć. Na poziomie abstrakcyjnym naszym celem jest stworzenie „idealnie bezpiecznego” kanału komunikacji, który zapewnia zarówno poufność, jak i integralność. Szukanie definicji tego rodzaju wykracza poza zakres tej książki. Zamiast tego zapewniamy prostszy zestaw definicji, które oddzielnie traktują tajemnicę i integralność. Te definicje i nasza późniejsza analiza wystarczą do zrozumienia kluczowych zagadnień.

(Przestrzegamy jednak czytelnika, że – w przeciwieństwie do szyfrowania i kodów uwierzytelniania wiadomości – w tej dziedzinie nie ustalono jeszcze standardowej terminologii i definicji uwierzytelnionego szyfrowania).

Niech $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ będzie schematem szyfrowania klucza prywatnego. Jak już wspomniano, bezpieczeństwo definiujemy odrębnie definiując tajemnicę i integralność. Pojęcie tajemnicy, które rozważamy, widzieliśmy już wcześniej: wymagamy, aby Π było zabezpieczone przed atakami z użyciem wybranego tekstu zaszyfrowanego, tj. aby było zabezpieczone CCA. (Omówienie i definicja bezpieczeństwa CCA można znaleźć w sekcji 3.7.) Niepokoją nas tutaj ataki z użyciem wybranego tekstu zaszyfrowanego, ponieważ wyraźnie rozważamy aktywnego przeciwnika, który może modyfikować dane przesypane od jednej uczciwej strony do drugiej. Nasze pojęcie integralności będzie w istocie pojęciem egzystencjalnej niepodrabialności w obliczu adaptacyjnego ataku wybranego przesłania. Ponieważ jednak Π nie spełnia składni kodu uwierzytelniającego wiadomość, wprowadzamy definicję specyficzną dla tego przypadku. Rozważmy następujący eksperyment zdefiniowany dla schematu szyfrowania kluczem prywatnym $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, przeciwnik A i wartość n dla parametru bezpieczeństwa: Eksperyment z szyfrowaniem

niemożliwym do podrobienia $\text{Enc-ForgeA}, \Pi(n)$:

1. Uruchom $\text{Gen}(1n)$, aby uzyskać klucz k.
2. Przeciwnik A otrzymuje dane wejściowe 1 n i dostęp p do wyroczni szyfrującej $\text{Enck}(\cdot)$. Przeciwnik wyprowadza szyfrogram c.
3. Niech $m := \text{Deck}(c)$ i niech Q oznacza zbiór wszystkich zapytań, które A zadał swojej wyroczni szyfrującej. Wynik eksperymentu wynosi 1 wtedy i tylko wtedy, gdy $(1) m = \dots$ i $(2) m \in Q$.

DEFINICJA 4.16 Schemat szyfrowania klucza prywatnego Π jest niemożliwy do podrobienia , jeśli dla wszystkich probabilistycznych przeciwników A w czasie wielomianowym istnieje pomijalne zaniedbywanie funkcji takie , że:

$$\Pr[\text{Enc-Forge}_A, \Pi(n) = 1] = \text{negl}(n).$$

Równolegle do naszej dyskusji na temat zapytań weryfikacyjnych zgodnie z Definicją 4.2, można tutaj również rozważyć silniejszą definicję , w której A ma dodatkowo dostęp p do wyroczni deszyfrującej. Można sprawdzić, czy bezpieczna konstrukcja, którą prezentujemy poniżej, również spełnia tę mocniejszą definicję .

Definiujemy teraz (bezpieczny) uwierzytelny schemat szyfrowania.

DEFINICJA 4.17 Schemat szyfrowania z kluczem prywatnym jest schematem szyfrowania z uwierzytelnieniem , jeżeli jest bezpieczny według CCA i nie da się go podrobić.

4.5.2 Konstrukcje ogólne

Kuszące może być myślenie, że jakakolwiek rozsądna kombinacja bezpiecznego schematu szyfrowania i bezpiecznego kodu uwierzytelniającego wiadomość powinna skutkować uwierzytelnionym schematem szyfrowania. W tej części pokażemy, że tak nie jest. Pokażemy to, że nawet bezpieczne narzędzia kryptograficzne można łączyć w taki sposób, że wynik jest niepewny, i po raz kolejny podkreśla znaczenie definicji i dowodów bezpieczeństwa. Z drugiej strony pokazujemy, jak można odpowiednio połączyć szyfrowanie i uwierzytelnianie wiadomości, aby osiągnąć wspólną tajemnicę i integralność.

W całym tekście niech $\Pi_E = (\text{Enc}, \text{Dec})$ będzie schematem szyfrowania zabezpieczonym CPA i niech $\Pi_M = (\text{Mac}, \text{Vrfy})$ oznacza kod uwierzytelniający wiadomość, gdzie generowanie klucza w obu schematach polega po prostu na wybraniu jednolitego n-bitowego klucza. Istnieją trzy naturalne podejścia do łączenia szyfrowania i uwierzytelniania wiadomości przy użyciu niezależnych kluczy⁴ kE i kM odpowiednio dla Π_E i Π_M :

1. Szyfruj i uwierzytelniaj: W tej metodzie szyfrowanie i uwierzytelnianie wiadomości są obliczane niezależnie, równolegle. Oznacza to, że biorąc pod uwagę wiadomość w postaci zwykłego tekstu m, nadawca przesyła tekst zaszyfrowany c, t gdzie:

$$c = \text{EnckE}(m) \text{ i } t = \text{MackM}(m).$$

Odbiorca odszyfrowuje c, aby odzyskać m; zakładając, że nie wystąpił żaden błąd, nastąpuje sprawdza znacznik t. Jeżeli $\text{VrfykM}(m, t) = 1$, odbiornik wyprowadza m; w przeciwnym razie wyświetla błąd.

⁴ W przypadku łączenia różnych schematów należy zawsze używać niezależnych kluczy kryptograficznych. Do tego punktu wracamy pod koniec tej sekcji.

2. Uwierzytelni, a nastę pnie zaszyfruj: Tutaj najpierw obliczany jest znacznik MAC t, a nastę pnie wiadomość i znacznik są wspólnie szyfrowane. Oznacza to, że po otrzymaniu wiadomości m nadawca przesyła tekst zaszyfrowany c obliczony jako:

$$t \quad \text{MackM}(m) \text{ i } c \quad \text{EnckE}(mt).$$

Odbiorca deszyfruje c, aby otrzymać mt; zakładając, że nie wystąpi żaden błąd, nastę pnie sprawdza znacznik t. Tak jak poprzednio, jeśli $\text{VrfykM}(m, t) = 1$, odbiornik wyprowadza m; w przeciwnym razie wyświetla błąd.

3. Szyfruj, a nastę pnie uwierzytelni: w tym przypadku wiadomość m jest najpierw szyfrowana, a nastę pnie na podstawie wyniku obliczany jest znacznik MAC. Oznacza to, że szyfrogramem jest para c, t gdzie:

$$c \quad \text{EnckE}(m) \text{ i } t \quad \text{MackM}(c).$$

(Patrz także Konstrukcja 4.18.) Jeśli $\text{VrfykM}(c, t) = 1$, to odbiorca deszyfruje c i podaje wynik; w przeciwnym razie wyświetla błąd.

Analizujemy każde z powyższych podejść, gdy są one tworzone z „ogólnymi” bezpiecznymi komponentami, tj. dowolnym schematem szyfrowania zabezpieczonym CPA i dowolnym (silnie) bezpiecznym adresem MAC. Chcemy podejścia, które zapewni wspólną tajemnicę i integralność podczas korzystania z jakichkolwiek (bezpiecznych) komponentów, dlatego odrzucimy jako „niebezpieczne” każde podejście, dla którego istnieje choćby jeden kontrprzykład bezpiecznego schematu szyfrowania/MAC, dla którego kombinacja jest niepewny. To podejście typu „wszystko albo nic” zmniejsza prawdopodobieństwo błę dów wdrożeniowych. W szczególności uwierzytelniony schemat szyfrowania można wdrożyć poprzez wywołanie „podprogramu szyfrowania” i „podprogramu uwierzytelniania wiadomości”, a implementacja tych podprogramów może zostać zmieniona w późniejszym czasie. (Ma to czę sto miejsce podczas aktualizacji bibliotek kryptograficznych lub modyfikacji standardów). Wdrożenie podejścia, którego bezpieczeństwo zależy od sposobu implementacji jego komponentów (a nie od bezpieczeństwa, jakie zapewniają), jest zatem niebezpieczne.

Podkreślamy, że jeśli podejście zostanie odrzucone, nie oznacza to, że jest ono niepewne dla wszystkich możliwych instancji komponentów; oznacza to jednak, że przed użyciem należy przeanalizować każdą instancję tego podejścia i sprawdzić, czy jest bezpieczna.

Szyfruj i uwierzytelni. Przypomnijmy, że w tym podejściu szyfrowanie i uwierzytelnianie wiadomości odbywa się niezależnie. Biorąc pod uwagę komunikat m, przesyłana wartość to c, t gdzie

$$c \quad \text{EnckE}(m) \text{ i } t \quad \text{MackM}(m).$$

Takie podejście może nie zapewnić nawet najbardziej podstawowego poziomu tajemnicy. Aby to zobaczyć, należy pamiętać, że bezpieczny adres MAC nie gwarantuje żadnej tajemnicy, dlatego też tag MackM(m) może ujawnić informacje o m osobie podsłuchującej.

(Jako trywialny przykład rozważmy bezpieczny adres MAC, w którym znajduje się pierwszy bit znacznika zawsze równy pierwszemu bitowi wiadomości.) Zatem szyfruj i uwierzytelnij podejście może skutkować schematem, który nie zawiera nawet nieroróżnialnych szfrowań w obecności podsłuchującego.

W rzeczywistości podejście polegające na szyfrowaniu i uwierzytelnianiu prawdopodobnie nie będzie bezpieczne ataki z wybranym tekstem jawnym, nawet jeśli są tworzone przy użyciu standardowych komponentów (w przeciwieństwie do wymyślonego kontrprzykładu w poprzednim akapicie). W szczególności, jeśli używany jest deterministyczny MAC, taki jak CBC-MAC, wówczas obliczany jest znacznik w wiadomości (dla pewnego klucza stałego kM) jest za każdym razem taka sama. To pozwala podsłuchiwanie w celu zidentyfikowania, kiedy ta sama wiadomość została wysłana dwukrotnie, i tak program nie jest bezpieczny pod względem CPA. Większość MAC stosowanych w praktyce ma charakter deterministyczny, tzw. stanowi to poważny problem.

Uwierzytelnij, a następnie zaszyfruj. Tutaj najpierw obliczany jest znacznik MAC $t = \text{Mack}_M(m)$; następnie mt jest szyfrowany i przesyłana jest wynikowa wartość $\text{Enck}_E(mt)$. Pokazujemy, że ta kombinacja również niekoniecznie daje uwierzytelny schemat szyfrowania.

Właściwie zetknęliśmy się już ze schematem szyfrowania zabezpieczonym przez CPA dla których to podejście jest niepewne: schemat trybu CBC z dopełnieniem omówiony w sekcji 3.7.2. (W dalszej części zakładamy, że czytelnik jest zaznajomiony z tą sekcją.) Przypomnijmy, że ten schemat działa w ten sposób, że najpierw dopełniamy zwykły tekst (który w naszym przypadku będzie mt) w określony sposób, tak aby wynik był wielokrotnością długości bloku, a następnie szyfrowanie wyniku w trybie CBC. Podczas deszyfrowania, jeśli po wykonaniu deszyfrowania w trybie CBC zostanie wykryty błąd w dopełnianiu, zwracany jest błąd „złe dopełnienie”. Odnośnie do uwierzytelnij, a następnie zaszyfruj, oznacza to, że istnieją teraz dwa źródła potencjału błędu deszyfrowania: dopełnienie może być nieprawidłowe lub znacznik MAC może nie zweryfikować. Schematycznie algorytm deszyfrowania Dec w schemacie połączonym działa w następujący sposób:

grudzień $kE, kM (C)$:

1. Oblicz $\sim m := \text{Talia}_E(c)$. Jeśli zostanie wykryty błąd w dopełnieniu, zwróć „złe dopełnienie” i zatrzymaj się.
2. Przeanalizuj $\sim m$ jako mt . Jeśli $\text{Vrfyk}_M(m, t) = 1$ zwróć m ; inaczej, wyjście „błąd autoryzacji.”

Zakładając, że atakujący potrafi rozróżnić dwa komunikaty o błędach, atakujący może zastosować ten sam atak z wybranym tekstem zaszyfrowanym, opisany w sekcji 3.7.2 do powyższego schematu, aby odzyskać cały oryginalny tekst jawny z danego szyfrogramu. (Wynika to z faktu, że atak padding-oracle pokazany w Sekcji 3.7.2 opiera się wyłącznie na możliwości dowiedzenia się, czy doszło do zdarzenia błędu dopełnienia, co ujawnia ten schemat.) Tego typu atak został pomyślnie przeprowadzony w świecie rzeczywistym w różnych ustawieniach, np. w konfiguracjach IPsec korzystających z metody uwierzytelniania, a następnie szyfrowania.

Jednym ze sposobów naprawienia powyższego schematu byłoby zapewnienie zwrócenia tylko jednego komunikatu o błędzie, niezależnie od źródła niepowodzenia deszyfrowania. Jest to niezadowalające rozwiązanie z kilku powodów: (1) mogą istnieć uzasadnione powody (np. użyteczność, debugowanie), aby pojawiać się wiele komunikatów o błędach; (2) wymuszenie identyczności komunikatów o błędach oznacza, że kombinacja nie jest już w pełni ogólna, tj. wymaga, aby osoba wdrażająca metodę „uwierzytelnię, a następnie pnie zaszyfruję” była świadoma, jakie komunikaty o błędach zwraca podstawowy CPA-bezpieczny schemat szyfrowania; (3) przede wszystkim niezwykle trudno jest zapewnić, że poszczególne błędy nie zostaną rozróżnione, ponieważ np. nawet różnica w czasie zwrotu każdego z tych błędów może zostać wykorzystana przez przeciwnika do ich rozróżnienia (por. nasze wcześniejsze omówienie ataków czasowych na końcu sekcji 4.2). W niektórych wersjach protokołu SSL próbowało używać tylko pojedynczego komunikatu o błędzie w połączeniu z metodą uwierzytelniania, a następnie pnie szyfrowania, ale atak Oracle z dopełnieniem nadal był pomyślnie przeprowadzany przy użyciu tego rodzaju informacji o taktowaniu. Doszliśmy do wniosku, że uwierzytelnianie-następnie-szyfrowanie nie zapewnia ogólnie uwierzytelnionego szyfrowania i nie powinno być stosowane.

Szyfrując, a następnie pnie uwierzytelnię. W tym podejściu wiadomość jest najpierw szyfrowana, a następnie pnie na podstawie wyniku obliczany jest adres MAC. Oznacza to, że wiadomością jest para c, t gdzie

$$c = \text{Enck}_E(m) \text{ i } t = \text{Mack}_M(c).$$

Deszyfrowanie c, t odbywa się poprzez wypisanie c , jeśli $\text{Vrfyk}_M(c, t) = 1$, a w przeciwnym razie wypisanie $\text{Deck}_E(c)$. Formalny opis można znaleźć w rozdziale Konstrukcja 4.18.

KONSTRUKCJA 4.18

Niech $\Pi_E = (\text{Enc}, \text{Dec})$ będzie schematem szyfrowania klucza prywatnego i niech $\Pi_M = (\text{Mac}, \text{Vrfy})$ będzie kodem uwierzytelniającym wiadomość, gdzie w każdym przypadku generowanie klucza odbywa się poprzez prosty wybór jednolitego n -bitowego klucza.

Zdefiniuj schemat szyfrowania klucza prywatnego Π_E , (grudnia) w następujący sposób:

- Gen : na wejściu 1^n , (Gen, Enc) wybierz niezależne, jednolite $k_E, k_M \in \{0, 1\}^n$ i wprowadź klucz (k_E, k_M) . • Enc :
- po wprowadzeniu klucza (k_E, k_M) i wiadomości w postaci zwykłego tekstu m , oblicz
- $$c = \text{Enck}_E(m) \text{ i } t = \text{Mack}_M(c).$$
- Wypisz szyfrogram c, t .
- Dec : po wprowadzeniu klucza (k_E, k_M) i tekstu zaszyfrowanego c, t , najpierw sprawdź, czy $\text{Vrfyk}_M(c, t) = 1$.
- Jeśli tak, wprowadź $\text{Deck}_E(c)$; jeśli nie, to
- wyjście .

Ogólna konstrukcja uwierzytelnionego schematu szyfrowania.

To podejście jest rozsądne, o ile MAC jest silnie bezpieczny, jak w Definicji 4.3. Jako intuicję dla bezpieczeństwa tego podejścia, powiedzmy, że szyfrogram c, t jest ważny, jeśli t jest prawidłowym znacznikiem MAC na c . Silne zabezpieczenia MAC zapewniają, że

przeciwnik nie będzie w stanie wygenerować żadnego prawidłowego tekstu zaszyfrowanego, którego nie otrzymał od swojej wyroczni szyfrującej. To natychmiast oznacza, że Konstrukcja 4.18 jest nie do podrobienia. Jeśli chodzi o bezpieczeństwo CCA, MAC obliczony dla zaszyfrowanego tekstu powoduje, że wyrocznia deszyfrująca staje się bezużyteczna, ponieważ dla każdego zaszyfrowanego tekstu c , t przeciwnik poddaje się swojej wyroczni deszyfrującej, albo już zna sposób odszyfrowania (jeśli otrzymał c , t w własnego szyfrowania lub acle) lub może spodziewać się błędu (ponieważ przeciwnik nie może wygenerować żadnych nowych, prawidłowych tekstów zaszyfrowanych). Oznacza to, że zabezpieczenie CCA połączonego programu ogranicza się do zabezpieczenia CPA PE. Należy również pamiętać, że adres MAC jest weryfikowany przed rozpoczęciem deszyfrowania; w związku z tym weryfikacja MAC nie może ujawnić żadnych informacji na temat zwykłego tekstu (w przeciwieństwie do ataku Oracle, który widzieliśmy w przypadku podejścia „uwierzytelnij, a nastąpi pnie zaszyfruj”). Sformalizujemy teraz powyższe argumenty.

TWIERDZENIE 4.19 Niech PE będzie schematem szyfrowania klucza prywatnego z zabezpieczeniem CPA i niech PM będzie silnie bezpiecznym kodem uwierzytelniającym wiadomość. Zatem Construction 4.18 jest uwierzytelnionym schematem szyfrowania.

DOWÓD Niech Π oznacza schemat wynikający z konstrukcji 4.18. Musimy pokazać, że Π jest nie do podrobienia i że jest bezpieczne dla CCA. Zgodnie z intuicją podaną powyżej, powiedzmy, że szyfrogram c , t jest ważny (w odniesieniu do jakiegoś ustalonego tajnego klucza (kE, kM)), jeśli $VrfykM(c, t) = 1$. Pokazujemy, że silne bezpieczeństwo ΠM oznacza, że (z wyjątkiem znikomej prawdopodobieństwa) wszelkie „nowe” szyfrogramy, które przeciwnik prześle do wyroczni deszyfrującej, będą nieważne.

Jak już wspomniano, oznacza to natychmiastową niepodrabialność. (W rzeczywistości jest silniejszy niż niemożność podrobienia.) Fakt ten czyni również wyrocznię deszyfrującą bezużyteczną i oznacza, że bezpieczeństwo CCA $\Pi = (\text{Gen}, \text{Dec})$ redukuje do Enc bezpieczeństwa CPA PE.

Bardziej szczegółowo, niech A będzie probabilistycznym przeciwnikiem działającym w czasie wielomianowym atakującym Konstrukcję 4.18 w ataku wybranym tekstem zaszyfrowanym (por. Definicja 3.33). Powiedzmy, że szyfrogram c , t jest nowy, jeśli A nie otrzymał c , t od swojej wyroczni deszyfrującej lub jako szyfrogram wyzwania. Niech ValidQuery będzie zdarzeniem, w którym A przesyła nowy zaszyfrowany tekst c , t do swojej wyroczni deszyfrującej, który jest ważny, tj. dla którego $VrfykM(c, t) = 1$. Dowodzimy:

Twierdzenie 4.20 $\Pr[\text{ValidQuery}]$ jest nieistotne.

DOWÓD Intuicyjnie wynika to z faktu, że jeśli wystąpi ValidQuery, to przeciwnik utworzył nową, prawidłową parę (c, t) w eksperymencie Mac-forge. Formalnie, niech $q(\cdot)$ będzie wielomianem górnym ograniczeniem liczby zapytań deszyfrujących-oracle wykonanych przez A i rozważmy następstwującego przeciwnika AM atakującego kod uwierzytelniający wiadomość ΠM (tj. działający w eksperymencie Mac-forgeAM, $\Pi M(N)$):

Przeciwnik AM : AM

otrzymuje wejście $1n$ i ma dostęp p do wyroczni MAC MackM (\cdot).

1. Wybierz jednorodne $kE \in \{0, 1\}^n$ i $i \in \{1, \dots, q(n)\}$.
2. Uruchom A na wejściu $1n$. Kiedy A wysyła zapytanie Oracle dotyczące wiadomości m , odpowiedz na nią w następujący sposób:
 - (i) Oblicz $c = EnckE(m)$.
 - (ii) Zapytaj c do wyroczni MAC i otrzymaj w odpowiedzi t .
 Wróć c, t do A.

Zaszyfrowany tekst wyzwania jest przygotowywany w dokładnie ten sam sposób (z jednolitym bitem $b \in \{0, 1\}$ wybranym w celu wybrania wiadomości mb , która ma zostać zaszyfrowana).

Kiedy A wykonuje zapytanie wyroczni deszyfrującej o zaszyfrowany tekst c, t , odpowiedz na nie w następujący sposób: Jeśli jest to i -te zapytanie wyroczni deszyfrującej, wypisz (c, t) . W przeciwnym

razie: (i) Jeśli c, t było odpowiedzią na poprzednie zapytanie Oracle dotyczące wiadomości m , wróć m . (ii) W przeciwnym
razie wróć \cdot .

W istocie AM „zgaduje”, że i -te zapytanie wyroczni deszyfrującej A będzie pierwszym nowym, prawidłowym zapytaniem wykonanym przez A. W takim przypadku AM wysyła prawidłową sfałszowaną wiadomość c , której nigdy wcześniej nie przesyłała do swojej własnej wyroczni MAC.

Oczywiście AM działa w probabilistycznym czasie wielomianowym. Przeanalizujemy teraz prawdopodobieństwo, że AM stworzy dobre fałszerstwo. Kluczową kwestią jest to, że widok A, gdy jest uruchamiany jako podprogram przez AM, jest rozkładany identycznie jak widok A w eksperymencie PrivKcca $A, \Pi(n)$ aż do wystąpienia zdarzenia ValidQuery. Aby to zobaczyć, zauważ, że zapytania wyroczni deszyfrującej A (jak również obliczenie szyfrogramu stanowiącego wyzwanie) są doskonale symulowane przez AM. Jeśli chodzi o zapytania wyroczni deszyfrującej A, dopóki nie pojawi się ValidQuery, wszystkie są poprawnie symulowane. W przypadku (i) jest to oczywiste. Jeśli chodzi o przypadek (ii), jeśli zaszyfrowany tekst c, t przesłany do wyroczni deszyfrującej jest nowy, to dopóki nie pojawiło się jeszcze ValidQuery, poprawną odpowiedzią na zapytanie wyroczni deszyfrującej jest oczywiście \cdot . (Zauważ, że przypadek (i) jest dokładnie przypadem, gdy c, t nie jest nowy, a przypadek (ii) jest dokładnie przypadem, gdy c, t jest nowy.) Przypomnijmy, że A nie wolno przesyłać zaszyfrowanego tekstu wezwania do wyroczni deszyfrującej.

Ponieważ widok A uruchamiany jako podprogram przez AM jest rozkładany identycznie jak widok A w eksperymencie PrivKcca $A, \Pi(n)$ do momentu wystąpienia zdarzenia ValidQuery (n), prawdopodobieństwo zdarzenia ValidQuery w eksperymencie Mac-forgeAM, ΠM wynosi takie samo jak prawdopodobieństwo tego zdarzenia w eksperymencie PrivKcca $A, \Pi(n)$.

Jeśli AM poprawnie odgadnie pierwszy indeks i , gdy wystąpi ValidQuery, wówczas AM wyprowadzi (c, t) , dla którego $VrfyK(n)(c, t) = 1$ (ponieważ c, t jest prawidłowe) i dla którego nigdy nie otrzymał znacznika t w odpowiedzi na zapytanie MackM(c) (ponieważ c, t jest nowe). Zatem w tym przypadku AM udaje się przeprowadzić eksperyment Mac-sforgeAM, ΠM (n).

Prawdopodobieństwo, że AM odgadnie i wynosi $1/q(n)$. Dlatego

$$\Pr[\text{Mac-sforgeAM}, \Pi(n) = 1] = \Pr[\text{ValidQuery}] / q(n).$$

Ponieważ Π jest silnie bezpiecznym MAC, a q jest wielomianem, dochodzimy do wniosku, że $\Pr[\text{ValidQuery}]$ jest nieistotne. ■

Twierdzenia 4.20 używamy do udowodnienia bezpieczeństwa Π . Łatwiejszym przypadkiem jest udowodnienie, że Π jest nie do podrobienia. Wynika to bezpośrednio z twierdzenia, dlatego przedstawiamy jedynie nieformalne uzasadnienie, a nie formalny dowód. Należy najpierw zauważyć, że przeciwnik A w eksperymencie z szyfrowaniem niemożliwym do podrobienia jest ograniczoną wersją przeciwnika w eksperymencie z wybranym tekstem zaszyfrowanym (w pierwszym przypadku przeciwnik ma dostęp p jedynie do wyroczni szyfrującej). Kiedy A pod koniec eksperymentu wysyła zaszyfrowany tekst c, t , „udaje się” tylko wtedy, gdy c, t jest ważny i nowy. Ale poprzednie twierdzenie pokazuje właśnie, że prawdopodobieństwo takiego zdarzenia jest znikome.

Nieco bardziej skomplikowane jest udowodnienie, że Π jest bezpieczne dla CCA. Niech A ponownie bę dzie probabilistycznym przeciwnikiem działającym w czasie wielomianowym atakującym Π w ataku wybranym tekstem zaszyfrowanym. Mamy

$$\Pr[\text{PrivKcca } A, \Pi(n) = 1] = \frac{\Pr[\text{ValidQuery}] + \Pr[\text{PrivKcca } A, \Pi(n) = 1 \text{ and ValidQuery}]}{\Pr[\text{ValidQuery}]} \quad (4,8)$$

Pokazaliśmy już, że $\Pr[\text{ValidQuery}]$ jest nieistotne. Następnie pujące twierdzenie kończy w ten sposób dowód twierdzenia.

Twierdzenie 4.21. Istnieje pomijalna funkcja negl taka, że

$$\Pr[\text{PrivKcca } A, \Pi(n) = 1 \text{ and ValidQuery}] \xrightarrow[1]{\neg} \text{zaniechanie}(n). 2$$

Aby udowodnić to twierdzenie, opieramy się na bezpieczeństwie CPA Π . Rozważmy następujące przeciwnika AE atakującego Π w ataku wybranym tekstem jawnym:

Przeciwnik AE : AE
otrzymuje wejście 1^n i ma dostęp p do $\text{Enc}_E(\cdot)$.

1. Wybierz jednorodny $kM \in \{0, 1\}^n$.
2. Uruchom A na wejściu 1^n . Kiedy A wysyła zapytanie Oracle dotyczące wiadomości m , odpowiedz na nią w następujący sposób:

(i) Zapytaj m do $\text{Enc}_E(\cdot)$ i otrzymaj c w odpowiedzi. (ii) Oblicz $t = \text{Mack}_M(c)$ i zwróć c, t do A.

Kiedy A wysyła zapytanie wyroczni deszyfrującej o zaszyfrowany tekst c, t , odpowiedz na nie w następujący sposób:

- Jeśli c, t było odpowiedzią na poprzednie zapytanie Oracle dotyczące wiadomości m , zwróć m . W przeciwnym razie zwróć \perp .
3. Kiedy A wysyła wiadomości (m_0, m_1) , wysyła te same wiadomości i otrzymuje w odpowiedzi zaszyfrowany tekst wyzwanego c . Oblicz $t = \text{Mack}_M(c)$ i zwróć c, t jako szyfr-tekst wyzwanego dla A. Kontynuuj odpowiadanie na zapytania wyroczni A jak powyżej.
4. Wyprowadź ten sam bit b , który jest wysyłany przez A.

Zauważ, że AE nie potrzebuje wyroczni deszyfrującej, ponieważ po prostu zakłada, że każde zapytanie deszyfrujące wysłane przez A, które nie było wynikiem poprzedniego zapytania wyroczni deszyfrującej, jest nieprawidłowe.

Oczywiście AE działa w probabilistycznym czasie wielomianowym. Co więcej, widok A, gdy jest uruchamiany jako podprogram przez AE, jest rozkładany identycznie jak widok A w eksperymencie $\text{PrivKcca}_{A,\Pi}(n)$, ale zdarzenie ValidQuery nigdy nie wystąpi. Dlatego prawdopodobieństwo, że AE powiedzie się \perp , gdy nie wystąpi ValidQuery , jest takie samo, jak prawdopodobieństwo, że A powiedzie się \perp , gdy nie wystąpi ValidQuery ; tj. $\Pr[\text{PrivKcpa}_{AE,\Pi}(n) = 1 \mid \text{ValidQuery}] = \Pr[\text{PrivKcca}_{A,\Pi}(n) = 1 \mid \text{ValidQuery}]$.

sugerując to

$$\Pr[\text{PrivKcpa}_{AE,\Pi}(n) = 1] = \Pr[\text{PrivKcpa}_{AE,\Pi}(n) = 1 \mid \text{ValidQuery}] = \Pr[\text{PrivKcca}_{A,\Pi}(n) = 1 \mid \text{ValidQuery}].$$

Ponieważ Π jest bezpieczne dla CPA, istnieje pomijalna funkcja negl taka, że $\Pr[\text{PrivKcpa}_{AE,\Pi}(n) = 1] = \frac{1}{2} + \text{negl}(n)$. To potwierdza tezę. ■

Potrzeba niezależnych kluczy. Kończymy tę sekcję, podkreślając podstawową zasadę kriptografii: różne instancje prywatnych kryptograficznych powinny zawsze używać niezależnych kluczy. Aby to zilustrować, rozważmy, co może się stać z metodologią szyfrowania, a następnie uwierzytelniania, gdy ten sam klucz k jest używany zarówno do szyfrowania, jak i uwierzytelniania. Niech F będzie silną permutacją pseudolosową. Wynika z tego, że F jest również silną permutacją pseudolosową¹.

Zdefiniuj $\text{Enck}(m) = F_k(mr)$ dla $m \in \{0, 1\}^{n/2}$ i jednolitego $r \in \{0, 1\}^{n/2}$ i zdefiniuj $\text{Mack}(c) = F_{k^{-1}}(C)$. Można wykazać, że ten schemat szyfrowania jest rzeczywiście to jest nawet zabezpieczony CCA; zobacz ćwiczenie 4.25) i wiemy, że podany kod uwierzytelniający wiadomość to bezpieczny adres MAC. Jednak kombinacja szyfrowania, a następnie uwierzytelniania przy użyciu tego samego klucza k , który zastosowano do wiadomości m , daje:

$$\text{Enck}(m), \text{Mack}(\text{Enck}(m)) = F_k(mr), F_{k^{-1}}(F_k(mr)) = F_k(mr), \text{pan},$$

a wiadomość m zostaje ujawniona wyraźnie! Nie jest to w żaden sposób sprzeczne z Twierdzeniem 4.19, ponieważ Konstrukcja 4.18 wyraźnie wymaga, aby kM i E były wybierane (jednociwie i) niezależnie. Zachęcamy czytelnika do sprawdzenia, gdzie ta niezależność została wykorzystana w dowodzie Twierdzenia 4.19.

4.5.3 Bezpieczne sesje komunikacyjne

Pokrótko opisujemy zastosowanie uwierzytelnionego szyfrowania w sytuacji, gdy dwie strony chcą komunikować się „bezpiecznie” – mianowicie przy zachowaniu wspólnej tajemnicy i integralności – w trakcie sesji komunikacyjnej. (Na potrzeby tej sekcji sesja komunikacyjna to po prostu okres czasu, podczas którego komunikujące się strony utrzymują stan). W naszym potraktowaniu jesteśmy tutaj celowo nieformalni; formalna definicja jest dość skomplikowana i temat ten prawdopodobnie dotyczy bardziej bezpieczeństwa sieci niż kryptografii.

Niech $\Pi = (\text{Enc}, \text{Dec})$ będzie uwierzytelnym schematem szyfrowania. Rozważmy dwie strony A i B, które mają wspólny klucz k i chcą używać tego klucza do zabezpieczenia swojej komunikacji w trakcie sesji. Oczywistą rzeczą do zrobienia jest użycie Π : Ilekroć, powiedzmy, A chce przesłać wiadomość m do B, oblicza $c = \text{Enc}_k(m)$ i wysyła c do B; z kolei B odszyfrowuje c , aby odzyskać wynik (ignorując wynik, jeśli odszyfrowanie zwróci \perp). Tę samą procedurę stosuje się, gdy B chce wysłać wiadomość do A. To proste podejście jednak nie wystarczy, ponieważ istnieje wiele potencjalnych ataków:

Zmiana kolejności ataku Osoba atakująca może zamienić kolejność wiadomości. Na przykład, jeśli A przesyła c_1 (szyfrowanie m_1), a następnie przesyła c_2 (szyfrowanie m_2), osoba atakująca, która ma pewną kontrolę nad siecią, może dostarczyć c_2 przed c_1 i w ten sposób spowodować, że B wyśle wiadomości w złej kolejności. Powoduje to rozbieżność między poglądami obu stron na temat ich sesji komunikacyjnej.

Atak powtórzeniowy Osoba atakująca może odtworzyć (ważny) tekst zaszyfrowany c wysłany wcześniej przez jedną ze stron. Ponownie powoduje to rozbieżność między tym, co jest wysyłane przez jedną stronę i odbierane przez drugą.

Atak odbicia Osoba atakująca może przyjąć zaszyfrowany tekst c wysłany od A do B i odesłać go z powrotem do A. To z kolei może spowodować rozbieżność między transkrypcjami sesji komunikacyjnej obu stron: A może wysłać wiadomość m , mimo że B nigdy nie wysłał taką wiadomość.

Na szczeble powyższym atakom można łatwo zapobiec, używając liczników do rozwiązania dwóch pierwszych i bitu kierunkowości, aby zapobiec trzeciemu.⁵ Opisujemy je łącznie. Każda ze stron utrzymuje dwa liczniki $\text{ctr}_{A,B}$ i $\text{ctr}_{B,A}$ śledzące liczbę komunikatów wysłanych od A do B (odpowiednio B do A) podczas sesji. Liczniki te są inicjowane na 0 i zwiększone za każdym razem, gdy strona wysyła lub odbiera (ważną) wiadomość. Strony ustalają także bit $b_{A,B}$ i definiują $b_{B,A}$ jako jego uzupełnienie. (Jednym ze sposobów osiągnięcia tego jest ustalenie $b_{A,B} = 0$, jeśli tożsamość A jest leksykograficznie mniejsza niż tożsamość B.)

⁵ W praktyce kwestię kierunkowości często rozwiązuje się po prostu poprzez posiadanie oddzielnych kluczy dla każdego kierunku (tzn. strony używają klucza k_A dla komunikatów wysyłanych z A do B i innego klucza k_B dla komunikatów wysyłanych z B do A).

Kiedy A chce przesłać wiadomość m do B, oblicza zaszyfrowany tekst c = $\text{Enck}(bA, BctrA, Bm)$ i wysyła c; następnie pnie zwiększa ctrA,B. Po otrzymaniu c strona B odszyfrowuje; jeśli wynik wynosi c, natychmiast odrzuca.

W przeciwnym razie analizuje odszyfrowaną wiadomość jako bctrm. Jeśli b = bA,B i ctr = ctrA,B, wówczas B wyprowadza m i zwiększa ctrA,B; w przeciwnym razie B odrzuca.

Powyższe kroki, mutatis mutandis, stosuje się c, gdy B wysyła wiadomość do A.

Zauważamy, że skoro strony i tak utrzymują stan (mianowicie liczniki ctrA,B i ctrB,A), strony mogą z łatwością zastosować schemat szyfrowania z uwierzytelnianiem stanowym Π.

4.5.4 Szyfrowanie CCA-Secure Z definicji wynika

Bezpośrednio, że każdy uwierzytelniony schemat szyfrowania jest również zabezpieczony przed atakami z wykorzystaniem wybranego tekstu zaszyfrowanego. Czy istnieją schematy szyfrowania klucza prywatnego zabezpieczone przez CCA, których nie da się podrobić? Rzeczywiście, istnieją; patrz ćwiczenie 4.25.

Można sobie wyobrazić aplikacje, w których potrzebne jest bezpieczeństwo CCA, ale uwierzytelnione szyfrowanie już nie. Jednym z przykładów może być użycie szyfrowania klucza prywatnego do transportu klucza. Jako konkretny przykład założymy, że serwer udostępnia użytkownikowi token sprzętowy odporny na manipulacje, w którym osadzony jest klucz długoterminowy k. Serwer może przesłać do tego tokena nowy, krótkoterminowy klucz k, podając użytkownikowi Enck(k); użytkownik powinien przekazać ten zaszyfrowany tekst tokenowi, który go odszyfruje i używa k przez następny okres czasu. Atak wybranym szyfrogramem przy tym ustawnieniu może pozwolić użytkownikowi nauczyć się k, czego użytkownik nie powinien być w stanie zrobić. (Zauważ, że w tym przypadku atak typu padding-oracle, który opiera się jedynie na zdolności użytkownika do określenia, czy doszło do niepowodzenia deszyfrowania, mógłby potencjalnie zostać przeprowadzony dość łatwo.) Z drugiej strony, niewiele szkody wyrządzi się c, jeśli użytkownik będzie w stanie to zrobić wygenerując „prawidłowy” tekst zaszyfrowany, który spowoduje, że token używa dowolnego (niepowiązanego) klucza k przez następny okres czasu. (Oczywiście zależy to od tego, co token robi z tym kluczem.)

Niezależnie od powyższego, w przypadku szyfrowania kluczem prywatnym najbardziej „naturalne” konstrukcje schematów bezpiecznych CCA, które i tak znamy, spełniają silniejszą definicję szyfrowania uwierzytelnionego. Inaczej mówiąc, nie ma prawdziwego powodu, aby kiedykolwiek używać schematu bezpiecznego CCA, który nie jest uwierzytelnionym schematem szyfrowania, po prostu dlatego, że tak naprawdę nie mamy żadnych konstrukcji spełniających pierwsze, które byłyby bardziej wydajne niż konstrukcje osiągające drugie.

Jednak z koncepcyjnego punktu widzenia ważne jest, aby zachować odrębnosć pojęć bezpieczeństwa CCA i szyfrowania uwierzytelnionego. Jeśli chodzi o bezpieczeństwo CCA, nie jesteśmy zainteresowani integralnością wiadomości jako taką; raczej chcemy zapewnić prywatność nawet w przypadku aktywnego przeciwnika, który może zakłócać komunikację od nadawcy do odbiorcy. Natomiast w odniesieniu do uwierzytelnionego szyfrowania interesują nas podwójne cele: tajemnica i integralność. Podkreślamy to tutaj, ponieważ w przypadku ustawnienia klucza publicznego, które omówimy w dalszej części książki, różnica między uwierzytelnionym szyfrowaniem a bezpieczeństwem CCA jest bardziej wyraźna.

4.6 *Mac z teorii informacji

W poprzednich sekcjach omawialiśmy kody uwierzytelniania wiadomości z zabezpieczeniem obliczeniowym, tj. tam, gdzie zakłada się, że atakujący ma ograniczony czas działania. Przywołując wyniki rozdziału 2, naturalne jest pytanie, czy możliwe jest uwierzytelnianie wiadomości w obecności nieograniczonego przeciwnika.

W tej części pokazujemy, w jakich warunkach możliwe jest osiągnięcie bezpieczeństwa informacyjnego (a nie obliczeniowego).

Pierwszą obserwacją jest to, że w tym kontekście niemożliwe jest osiągnięcie „idealnego” bezpieczeństwa: mianowicie nie możemy mieć nadziei na kod uwierzytelniający wiadomość, dla którego prawdopodobieństwo, że przeciwnik wyśle prawidłowy znacznik do wcześniej nieuwierzytelnionej wiadomości, wynosi 0. Powód polega na tym, że przeciwnik może po prostu odgadnąć prawidłowy znacznik t w dowolnej wiadomości i odgadnięć ciebie z prawidłowej z prawdopodobieństwem (co najmniej $\frac{1}{2}$) oznaczającą długość znacznika schematu.

Powyższy przykład mówi nam, co możemy osiągnąć: MAC z parą formularzy znaczników długości $|t|$, gdzie prawdopodobieństwo fałszerstwa wynosi co najwyżej $\frac{1}{2}|t|$ nieograniczonych przeciwników. Zobaczmy, że jest to możliwe do osiągnięcia, ale tylko pod warunkiem ograniczenia liczby wiadomości uwierzytelnianych przez uczciwe strony.

Najpierw zdefiniujemy bezpieczeństwo informatyczne dla kodów uwierzytelniających wiadomości. Punktem wyjścia jest eksperyment Mac-forgeA, $\Pi(n)$, który służy do zdefiniowania bezpieczeństwa obliczeniowo bezpiecznych adresów MAC (por. Definicja 4.2), ale porzucenie parametru bezpieczeństwa n i wymaganie, aby $\Pr[\text{Mac-forgeA}, \Pi = 1]$ powinien być „mały” dla wszystkich przeciwników A (a nie tylko przeciwników działających w czasie wielomianowym). Jak jednak wspomniano powyżej (i co zostanie formalnie wykazane w podrozdziale 4.6.2), osiągnięcie ciebie takiej definicji jest niemożliwe. Przeciwnie, bezpieczeństwo oparte na teorii informacji można osiągnąć tylko wtedy, gdy nałożymy pewne ograniczenia na liczbę wiadomości uwierzytelnianych przez uczciwe strony. Patrzmy tutaj na najbardziej podstawowe ustawienie, w którym strony uwierzytelniają tylko pojedynczą wiadomość. Nazywamy to jednorazowym uwierzytelnianiem wiadomości. Poniższy eksperyment modyfikuje Mac-forgeA, $\Pi(n)$ zgodnie z powyższą dyskusją:

Jednorazowy eksperyment z uwierzytelnianiem wiadomości Mac-forge1-time A, Π :

1. Klucz k jest generowany poprzez uruchomienie Gen .
2. Przeciwnik A wysyła wiadomość ma tag t , i jest dawane w zamian $\text{Mack}(m)$.
3. Wyjście A (m, t).
4. Wynik eksperymentu definiuje się jako 1 wtedy i tylko wtedy
 - (1) $\text{Vrfy}(m, t) = 1$ i (2) $m = m$.

DEFINICJA 4.22 Kod uwierzytelniający wiadomość $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ jest jednorazowy ϵ -bezpieczny lub po prostu ϵ -bezpieczny, jeśli dla wszystkich (nawet nieograniczonych) przeciwników A:

$$\Pr[\text{Mac-forge1-time}, \Pi = 1] \leq \epsilon.$$

4.6.1 Konstruowanie MAC w teorii informacyjnej

W tej sekcji pokażemy, jak zbudować ϵ -bezpieczny MAC w oparciu o dowolny silny, funkcja uniwersalna.⁶ Następnie pokażemy prostą konstrukcję tego ostatniego.

Niech $h : K \times M \rightarrow T$ będzie funkcją z kluczem, której pierwszym wejściem jest klucz $k \in K$, a drugim wejściem jest wzięty z jakiejś dziedziny M . Jak zwykle piszemy $hk(m)$ zamiast $h(k, m)$. Następnie h jest silnie uniwersalne (lub niezależne od par), jeśli dla dowolnych dwóch różnych danych wejściowych m, m' wartości $hk(m)$ i $hk(m')$ są równomiernie i niezależnie rozłożone w T , gdy k jest kluczem jednolitym. Jest to równoznaczne ze stwierdzeniem, że prawdopodobieństwo, że $hk(m), hk(m')$ przyjmą dowolne wartości t, t' wynosi dokładnie $1/|T|^2$. To jest:

DEFINICJA 4.23 Funkcja $h : K \times M \rightarrow T$ jest silnie uniwersalna, jeśli dla wszystkich odrębnych $m, m' \in M$ i wszystkich $t, t' \in T$ zachodzi to

$$\Pr[hk(m) = t \quad hk(m') = t'] = \frac{1}{|T|^2},$$

gdzie przyjmuje się prawdopodobieństwo jednolitego wyboru $k \in K$.

Powyższe powinno motywować do budowy jednorazowego kodu uwierzytelniającego wiadomość z dowolnej silnie uniwersalnej funkcji h . Znacznik t w wiadomości m uzyskuje się poprzez obliczenie $hk(m)$, gdzie klucz k jest jednolity; patrz Konstrukcja 4.24. Intuicyjnie, nawet gdy przeciwnik zaobserwuje znacznik $t := hk(m)$ dla dowolnej wiadomości m , poprawny znacznik $hk(m')$ dla dowolnej innej wiadomości m' jest nadal równomiernie rozłożony w T z punktu widzenia przeciwnika. Zatem przeciwnik nie może zrobić nic innego, jak tylko na ślepo odgadnąć znacznik, a to przypuszczenie będzie prawidłowe tylko z prawdopodobieństwem $1/|T|$.

KONSTRUKCJA 4.24

Niech $h : K \times M \rightarrow T$ będzie funkcją silnie uniwersalną. Zdefiniuj adres MAC dla wiadomości $m \in M$ w następujący sposób:

- Gen: wybierz uniform $k \in K$ i wypisz go jako klucz.
- Mac: po wprowadzeniu klawisza $k \in K$ i wiadomości $m \in M$, wyprowadź znacznik $t := hk(m)$.
- Vrfy: na wejściu klucz $k \in K$, wiadomość $m \in M$ i znacznik $t \in T$ wyjście 1 wtedy, i tylko wtedy, gdy $t := hk(m)$. (Jeśli $m \notin M$, to wynik 0.)

MAC z dowolnej silnie uniwersalnej funkcji.

⁶ Nazywa się je często uniwersalnymi funkcjami skrótu, ale w kontekście kryptograficznym termin „hasz” ma inne znaczenie, o którym przekonamy się w dalszej części książki.

Powyższą konstrukcję można postrzegać jako analogiczną do konstrukcji 4.5. Dzieje się tak, ponieważ silnie uniwersalna funkcja h jest identyczna z funkcją losową, o ile jest oceniana tylko dwukrotnie.

TWIERDZENIE 4.25 Niech $h : K \times M \rightarrow T$ bęź dzie funkcją silnie uniwersalną.

Zatem Construction 4.24 jest bezpiecznym MAC $1/|T|$ -dla wiadomości w M .

DOWÓD Niech A bęź dzie przeciwnikiem. Jak zwykle w teorii informacji, możemy założyć, że A jest deterministyczne bez utraty ogólności. Zatem komunikat m , który A wysyła na początek eksperymentu, jest stały. Co więcej, para (m, t) , którą A wyprowadza na koniec eksperymentu, jest deterministyczną funkcją znacznika t na m , który A otrzymuje. Mamy zatem

$$\begin{aligned} \Pr[\text{Mac-forge1-time} = 1] &= \Pr_{\substack{t \in T \\ (m, t) := A(t)}} [\text{Mac-forge1-time} = 1 \quad h(m) = t] \\ &= \Pr_{\substack{t \in T \\ (m, t) := A(t)}} [h(m) = t \quad h(m) = t] \\ &= \frac{1}{|T|^2} = \frac{1}{|T|}. \end{aligned}$$

To dowodzi twierdzenia. ■

Przejdzmy teraz do klasycznej konstrukcji funkcji silnie uniwersalnej. Zakładamy podstawową wiedzę na temat arytmetyki modulo liczby pierwszej; czytelnicy mogą zapoznać się z sekcjami 8.1.1 i 8.1.2, aby uzyskać niezbędną dne informacje. = przyjmujemy pewną liczbę $\def{p}{\{0, \dots, p-1\}}$. Za naszą przestrzeń wiadomości pierwszą p i niech $Zp = Zp$; przestrzeń możliwych znaczników bęź dzie również wynosić $T = Zp$. Klucz (a, b) składa się z pary elementów z Zp ; zatem $K = Zp \times Zp$. Zdefiniuj h jako

$$h_{a,b}(m) \stackrel{\text{def}}{=} [a \cdot m + b \bmod p],$$

gdzie zapis $[X \bmod p]$ odnosi się do redukcji liczby całkowitej X modulo p (a zatem $[X \bmod p] \in Zp$ zawsze).

TWIERDZENIE 4.26 Dla dowolnej liczby pierwszej p funkcja h jest silnie uniwersalna.

DOWÓD Ustal dowolne odrę bne $m, m \in Zp$ i dowolne $t, t \in Zp$. Dla jakich kluczy (a, b) zachodzi stwierdzenie, że zarówno $h_{a,b}(m) = t$, jak i $h_{a,b}(m') = t$? To obowiązuje tylko wtedy, gdy

$$a \cdot m + b = t \bmod p \text{ i } a \cdot m' + b = t \bmod p.$$

Mamy zatem dwa równania liniowe z dwiema niewiadomymi a, b . Obydwa te równania są dokładnie spełnione, gdy $a = [(t - t) \cdot (m - m) \bmod p]$ i $b = [t - a \cdot m \bmod p]$; zauważ, że $[(m - m) \bmod p]$ istnieje, ponieważ $m = m$, a więc $c \cdot m - m = 0 \bmod p$. Przekształcon¹ oznacza to, że dla dowolnego m, m, t, t jak powyżej istnieje unikalny klucz (a, b) z $ha, b(m) = t \cdot i \cdot ha, b(m) = t$. Ponieważ istnieją klucze $|T|$, wnioskujemy, że prawdopodobieństwo (przy wyborze klucza), że $ha, b(m) = t \cdot i \cdot ha, b(m) = t$ wynosi dokładnie $1/|K| = 1/|T|$ zgodnie z wymaganiami.



Parametry konstrukcyjne 4.24. Pokrótkę omówimy parametry Konstrukcji 4.24 utworzonej za pomocą silnie uniwersalnej funkcji opisanej powyżej, ignorując fakt, że p nie jest potęgą 2. Konstrukcja jest bezpiecznym MAC $1/|T|$ ze znacznikami długości $\log |T|$; długość znacznika jest optymalna dla osiągnięcia tego poziomu bezpieczeństwa.

Niech M będzie stałą przestrzenią komunikatów, dla której chcemy skonstruować jednorazowy bezpieczny adres MAC. Powyższa konstrukcja daje bezpieczny adres MAC $1/|M|$ z kluczami dwukrotnie dłuższymi od wiadomości. Czytelnik może zauważać tutaj dwa problemy, na przeciwległych krańcach spektrum: Po pierwsze, jeśli $|M|$ jest mały niż $1/|M|$ prawdopodobieństwo fałszerstwa może być niedopuszczalnie duże. Z drugiej strony, jeśli $|M|$ jest wiele ksybi niż $1/|M|$ prawdopodobieństwo fałszerstwa może być przesadzone; można zaakceptować (nieco) wiele ksybi prawdopodobieństwo fałszerstwa, jeśli ten poziom bezpieczeństwa można osiągnąć za pomocą krótszych kluczy. Pierwszy problem (kiedy $|M|$ jest mały) można łatwo rozwiązać, po prostu osadzając M w wiele ksybi przestrzeni komunikatów M , na przykład dopełniając wiadomości zerami. Drugi problem można również rozwiązać; zobacz odniesienia na końcu tego rozdziału.

4.6.2 Ograniczenia dotyczące informacyjno-teoretycznych adresów MAC

W tej sekcji zbadamy ograniczenia dotyczące teoretycznego uwierzytelniania wiadomości. Pokazujemy, że każdy 2-n-zabezpieczony MAC musi mieć klucze o długości co najmniej $2n$. Rozszerzenie dowodu pokazuje, że każdy -time 2-n-secure MAC (gdzie bezpieczeństwo jest zdefiniowane poprzez naturalną modyfikację definicji 4.23) wymaga kluczy o długości co najmniej $(+1) \cdot n$. Konsekwencją tego jest to, że żaden adres MAC z kluczami o ograniczonej długości nie może zapewnić bezpieczeństwa informatycznego podczas uwierzytelniania nieograniczonej liczby wiadomości.

Poniżej zakładamy, że przestrzeń komunikatów zawiera co najmniej dwa komunikaty: m_1, m_2 ; jeśli nie, nie ma sensu komunikować się, a co dopiero uwierzytelniać.

TWIERDZENIE 4.27 Niech $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ będzie 2-n-zabezpieczonym MAC, w którym wszystkie klucze wprowadzane przez Gen mają tę samą długość. Następnie klucze wprowadzane przez Gen muszą mieć długość co najmniej $2n$.

DOWÓD Napraw dwie różne wiadomości m_0, m_1 w przestrzeni wiadomości. Intuicja dowodu jest taka, że dla znacznika musi istnieć co najmniej $2n$ możliwości

m_0 (w przeciwnym razie przeciwnik mógłby to odgadnąć z prawdopodobieństwem więcej niż 2^{-n}); ponadto, nawet w zależności od wartości znacznika dla m_0 , musi istnieć 2 n możliwości dla znacznika m_1 (w przeciwnym razie przeciwnik mógłby sfałszować znacznik na m_1 z prawdopodobieństwem więcej niż 2^{-n}). Ponieważ każdy klucz definiuje znaczniki dla m_0 i m_1 , oznacza to, że musi istnieć co najmniej $2n \times 2^n$ kluczy. Poniżej robimy to formalnie.

Niech K oznacza przestrzeń kluczy (tj. zbiór wszystkich możliwych kluczy, które mogą zostać wyprowadzone przez Gen). Dla dowolnego możliwego znacznika t_0 , niech $K(t_0)$ oznacza zbiór kluczy, dla którego t_0 jest prawidłowym znacznikiem na m_0 ; tj,

$$K(t_0) \stackrel{\text{def}}{=} \{k \mid Vrfyk(m_0, t_0) = 1\}.$$

Dla dowolnego t_0 musimy mieć $|K(t_0)| \geq 2 \cdot |K|^{1/2}$. W przeciwnym razie przeciwnik mógłby po prostu wypisać (m_0, t_0) jako fałszerstwo; byłoby to prawidłowe fałszerstwo z prawdopodobieństwem co najmniej $|K(t_0)| / |K| > 2^{-n}$, co jest sprzeczne z deklarowanym bezpieczeństwem.

Rozważmy teraz przeciwnika A, który żąda znacznika w wiadomości m_0 , otrzymuje w zamian znacznik t_0 , wybiera jednolity klucz $k \in K(t_0)$ i wyprowadza $(m_1, Mack(m_1))$ jako swoje fałszerstwo. Prawdopodobieństwo, że A wygeneruje prawidłowe fałszerstwo, wynosi co najmniej

$$\Pr_{t_0}[\text{Mack}(m_0) = t_0] \cdot \frac{1}{|K(m_0, t_0)|} = \frac{\Pr_{t_0}[\text{Mack}(m_0) = t_0] \cdot \frac{2^n}{|K|}}{\frac{2^n}{|K|}} = \frac{2^n}{|K|}.$$

Dzięki deklarowanemu bezpieczeństwu schematu prawdopodobieństwo, że przeciwnik może przedstawić prawidłowe fałszerstwo, wynosi co najwyżej 2^{-n} . Zatem musimy mieć $|K| \geq 2^{2n}$. Ponieważ wszystkie klucze mają tę samą długość, każdy klucz musi mieć długość co najmniej 2 ■

Referencje i dodatkowe lektury

Definicja bezpieczeństwa kodów uwierzytelniających wiadomości została dostosowana przez Bellare i in. [18] z definicji bezpieczeństwa podpisów cyfrowych podanej przez Goldwassera i in. [81] (patrz rozdział 12). Więcej na temat wariantu definicyjnego, w którym dozwolone są zapytania weryfikacyjne, można znaleźć w [17].

Paradygmat wykorzystania funkcji pseudolosowych do uwierzytelniania wiadomości (jak w Construction 4.5) został wprowadzony przez Goldreicha i in. [77]. Konstrukcja 4.7 jest dziełem Goldreicha [76].

CBC-MAC został ujednolicony na początku lat 80. XX wieku [94, 178] i nadal jest szeroko stosowany. Bellare i in. udowodnili, że podstawowy CBC-MAC jest bezpieczny (do uwierzytelniania wiadomości o stałej długości). [18]. Bernstein [26, 27] podaje bardziej bezpośredni (choć być może mniej intuicyjny) dowód, a także omawia pewne uogólnione

wersje CBC-MAC. Jak zauważono w tym rozdziale, podstawowy protokół CBC-MAC jest niepewny gdy jest używany do uwierzytelniania wiadomości o różnej długości. Jeden sposób, aby to naprawić polega na dodaniu długości wiadomości. Ma to tę wadę, że nie możliwość poradzenia sobie ze strumieniowym przesyaniem danych tam, gdzie długość wiadomości nie jest wystarczająca znane z góry. Petrank i Rackoff [137] proponują alternatywę „on-line” podejście do tego problemu. Dalsze ulepszenia zostały wprowadzone przez firmę Black i Rogaway [32] oraz Iwata i Kurosawa [95]; doprowadziło to do nowej propozycji standard o nazwie CMAC.

Po raz pierwszy wyraźnie podkreślono znaczenie uwierzytelnionego szyfrowania w [100, 19], którzy proponują definicje podobne do tych, które tutaj podaliśmy. Bellare i Namprempre [19] analizują trzy omawiane podejścia ogólne tutaj, chociaż pomysł użycia encrypt-then-authenticate w celu osiągnięcia bezpieczeństwa CCA się ga przynajmniej prac Doleva i in. [61]. Krawczyk [108] bada inne metody osiągania poufności i uwierzytelniania, a także analizuje metodę uwierzytelniania, a nastepnie szyfrowania stosowaną w SSL. Degabriele i Paterson [54] pokazuje atak na IPsec, gdy jest skonfigurowany do uwierzytelniania, a nastepnie szyfrowania (domyślnym szyfrowaniem uwierzytelnionym jest w rzeczywistości szyfrowanie, a następnie uwierzytelnianie; jednakże możliwe jest osiągnięcie uwierzytelnienia, a następnie szyfrowania w niektórych konfiguracjach). Zaproponowano również kilka nieogólnych schematów uwierzytelnionego szyfrowania; szczegółowe porównanie można znaleźć w [110].

MAC z teorii informacji zostały po raz pierwszy zbadane przez Gilberta i in. [73]. Weg-man i Carter [177] wprowadzili pojęcie funkcji silnie uniwersalnych, i odnotowali ich zastosowanie do jednorazowego uwierzytelniania wiadomości. Oni też pokazali, jak zmniejszyć długość klucza dla tego zadania, używając prawie mocno funkcja uniwersalna. W szczególności konstrukcja, którą tu podajemy, zapewnia bezpieczeństwo dla komunikatów o długości n z kluczami o długości $O(n)$; Wegmana i Cartera pokazali, jak skonstruować 2-n-zabezpieczony adres MAC dla wiadomości o długości z kluczami (zasadniczo) $O(n \cdot \log n)$. Prosta konstrukcja o charakterze mocno uniwersalnym funkcja, którą tutaj podajemy, jest (z niewielkimi różnicami) funkcją Cartera i Weg-mana [42]. Czytelnik zainteresowany pogłębieniem wiedzy na temat teorii informacji MACs odwołuje się do artykułu Stinsona [166], ankiety Simmonsa [162], lub pierwsze wydanie podręcznika Stinsona [167, rozdział 10].

Ćwiczenia

- 4.1 Powiedzmy, że $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ jest bezpiecznym MAC i dla $k \in \{0, 1\}^n$ algorytm generowania znaczników Mack zawsze generuje znaczniki o długości $t(n)$. Udowodnić, że t musi być superlogarytmiczne lub równoważnie, że jeśli $t(n) = O(\log n)$, wówczas Π nie może być bezpiecznym adresem MAC.

Wskazówka: rozważ prawdopodobieństwo przypadkowego odgadnięcia prawidłowego tagu.

4.2 Rozważ rozszerzenie definicji bezpiecznego uwierzytelniania wiadomości, w przypadku gdy przeciwnik ma do dyspozycji zarówno komputer Mac , jak i wyrocznię Vrfy .

(a) Podaj formalną definicję zabezpieczenia w tym przypadku. (b)

Założmy, że Π jest deterministycznym MAC przy użyciu weryfikacji kanonicznej, która spełnia definicję 4.2. Udowodnić, że Π spełnia także definicję z częścią (a).

4.3 Założmy, że istnieją bezpieczne adresy MAC. Podaj konstrukcję MAC, która jest bezpieczna w odniesieniu do definicji 4.2, ale która nie jest bezpieczna, gdy przeciwnik uzyska dodatkowo dostęp do wyroczni Vrfy (por. poprzednie ćwiczenie).

4.4 Dowód twierdzenia 4.4.

4.5 Założmy, że istnieją bezpieczne adresy MAC. Udowodnić, że istnieje adres MAC, który jest bezpieczny (zgodnie z definicją 4.2), ale nie jest silnie bezpieczny (zgodnie z definicją 4.3).

4.6 Rozważmy następujący MAC dla komunikatów o długości $(n) = 2n - 2$ przy użyciu funkcji pseudolosowej F : Po wprowadzeniu komunikatu m_0m_1 (z $|m_0| = |m_1| = n - 1$) i klucza $k \in \{0, 1\}^n$, algorytm Mac wykonywa t = $F_k(m_0) F_k(m_1)$. Algorytm Vrfy jest zdefiniowany w sposób naturalny.

Czy (Gen, Mac, Vrfy) jest bezpieczne? Udowodnij swoją odpowiedź.

4.7 Niech F będzie funkcją pseudolosową. Pokaż, że każdy z poniższych adresów MAC jest niezabezpieczony, nawet jeśli jest używany do uwierzytelniania wiadomości o stałej długości. (W każdym przypadku Gen generuje jednolite $k \in \{0, 1\}^n$. Niech i oznacza n/2-bitowe kodowanie liczby całkowitej i .)

(a) Aby uwierzytelnić wiadomość $m = m_1, \dots, m_i$, gdzie $m_i \in \{0, 1\}^{n/2}$, oblicz $t := F_k(m_1) \dots F_k(m_i)$.

(b) Aby uwierzytelnić wiadomość $m = m_1, \dots, m_i$, gdzie $m_i \in \{0, 1\}^{n/2}$, oblicz $t := F_k(m_1) \dots F_k(m_i)$.

(c) Aby uwierzytelnić wiadomość $m = m_1, \dots, m_i$, gdzie $m_i \in \{0, 1\}^{n/2}$, wybierz uniform $r \in \{0, 1\}^n$, oblicz

$$t := F_k(r) F_k(m_1) \dots F_k(m_i),$$

i niech znacznikiem będzie r, t .

4.8 Niech F będzie funkcją pseudolosową. Pokaż, że następujący MAC dla komunikatów o długości $2n$ jest niepewny: Gen generuje jednolite $k \in \{0, 1\}^n$.

Aby uwierzytelnić wiadomość m_1m_2 za pomocą $|m_1| = |m_2| = n$, oblicz znacznik $F_k(m_1) F_k(F_k(m_2))$.

4.9 Biorąc pod uwagę dowolny deterministyczny MAC (Mac, Vrfy), możemy postrzegać Mac jako funkcję z kluczem. Zarówno w konstrukcjach 4.5, jak i 4.11 Mac jest funkcją pseudolosową. Podaj konstrukcję bezpiecznego, deterministycznego MAC, w którym Mac nie jest funkcją pseudolosową.

- 4.10 Czy Konstrukcja 4.5 jest koniecznie bezpieczna, gdy jest tworzona przy użyciu słabej funkcji pseudolosowej (por. ćwiczenie 3.26)? Wyjaśnić.
- 4.11 Udowodnij, że Konstrukcja 4.7 jest bezpieczna nawet wtedy, gdy przeciwnik uzyska dodatkowo dostęp p do wyroczni Vrfy (por. Ćwiczenie 4.2).
- 4.12 Udowodnij, że Konstrukcja 4.7 jest bezpieczna, jeśli zmienimy ją w następujący sposób: Ustaw $t := F_k(r b i m)$ gdzie b jest pojedynczym bitem takim, że $b = 0$ we wszystkich blokach oprócz ostatniego i $b = 1$ w ostatnim bloku. (Założmy dla uproszczenia, że długość wszystkich uwierzytelnianych wiadomości jest zawsze całkowitą wielokrotnością $n/2 - 1$.) Jaka jest zaleta tej modyfikacji?
- 4.13 Badamy, co się dzieje, gdy podstawowa konstrukcja CBC-MAC jest używany z wiadomościami o różnej długości.
- (a) Założymy, że nadawca i odbiorca nie zgadzają się z góra co do długości wiadomości (a zatem $Vrfy_k(m, t) = 1$ jeśli $t \equiv Mack(m)$, niezależnie od długości m), ale nadawca jest ostrożny do uwierzytelniania tylko wiadomości o długości $2n$. Pokaż, że przeciwnik może sfałszować prawidłowy znacznik w wiadomości o długości $4n$. (b) Założymy, że odbiorca akceptuje tylko wiadomości 3-blokowe (więcej o $Vrfy_k(m, t)$)
- = 1 tylko jeśli m ma długość $3n$ i $t \equiv Mack(m)$, ale nadawca uwierzytelnia wiadomości o dowolnej długości będącej wielokrotnością n . Pokaż, że przeciwnik może sfałszować prawidłowy tag w nowej wiadomości.
- 4.14 Udowodnić, że poniższe modyfikacje podstawowego CBC-MAC nie dają rezultatu bezpieczny adres MAC (nawet dla wiadomości o stałej długości):
- (a) Mac wyprowadza wszystkie bloki t_1, \dots, t_n , a nie tylko t . (Weryfikacja sprawdza tylko, czy t jest poprawne.)
- b) Za każdym razem, gdy wiadomość jest uwierzytelniana, używany jest losowy blok początkowy. Oznacza to, że wybierz uniform $t_0 \in \{0, 1\}^n$, uruchom podstawowy CBC-MAC nad „wiadomością” t_0, m_1, \dots, m_i i wypisz znacznik t_0, t . Weryfikacja odbywa się w sposób naturalny.
- 4.15 Pokaż, że dodanie długości wiadomości na końcu wiadomości przed zastosowaniem podstawowego CBC-MAC nie skutkuje bezpiecznym adresem MAC dla wiadomości o dowolnej długości.
- 4.16 Pokaż, że kodowanie wiadomości o dowolnej długości opisane w rozdz. Wersja 4.4.2 nie zawiera prefiksów.
- 4.17 Rozważmy następujące kodowanie, które obsługuje wiadomości, których długość jest mniejsza niż $n - 2n$: Kodujemy ciąg $m \in \{0, 1\}^n$ najpierw dołączając tyle zer, ile potrzeba, aby długość wynikowego ciągu m była niezerową wielokrotnością n . Następnie dodajemy liczbę bloków w m (równoważnie dodajemy liczbę całkowitą $|m|/n$), zakodowaną jako ciąg n-bitowy.
- Pokaż, że to kodowanie nie jest wolne od prefiksów.

- 4.18 Udowodnić, że poniższa modyfikacja podstawowego CBC-MAC daje bezpieczny MAC dla komunikatów o dowolnej długości (dla uproszczenia założmy, że długość wszystkich komunikatów jest wielokrotnością długości bloku). Mack(m) najpierw oblicza $k = F_k()$, gdzie jest długością m . Znacznik jest następnie obliczany przy użyciu podstawowego CBC-MAC z kluczem k . Weryfikacja odbywa się w sposób naturalny.
- 4.19 Niech F będzie funkcją z kluczem, która jest bezpiecznym (deterministycznym) MAC dla komunikatów o długości n . (Zauważ, że F nie musi być pseudolosową mutacją.) Pokaż, że podstawowy CBC-MAC niekoniecznie jest bezpiecznym MAC (nawet dla wiadomości o stałej długości), gdy jest tworzony za pomocą F .
- 4.20 Pokaż, że konstrukcja 4.7 jest silnie bezpieczna.
- 4.21 Pokaż, że konstrukcja 4.18 może nie być zabezpieczona CCA, jeśli zostanie utworzona z bezpiecznym adresem MAC, który nie jest silnie bezpieczny.
- 4.22 Udowodnij, że Konstrukcji 4.18 nie można podrobić, jeśli zostanie utworzona z dowolnym schematem szyfrowania (nawet jeśli nie jest zabezpieczony CPA) i dowolnym bezpiecznym adresem MAC (nawet jeśli adres MAC nie jest silnie bezpieczny).
- 4.23 Rozważmy wzmocnioną wersję niepodrabialności (Definicja 4.16), w której A ma dodatkowo dostęp p do wyroczni deszyfrującej.
 (a) Napisz formalną definicję tej wersji niepodrabialności. (b) Udowodnij, że Konstrukcja 4.18 spełnia tę silniejszą definicję, jeśli ΠM jest silnie bezpiecznym MAC. (c) Pokaż za pomocą kontrprzykładu, że Konstrukcja 4.18 nie musi spełniać tej silniejszej definicji, jeśli ΠM jest bezpiecznym MAC, który nie jest silnie bezpieczny. (Porównaj z poprzednim ćwiczeniem.)
- 4.24 Udowodnić, że metoda uwierzytelniania, a następnie szyfrowania, realizowana przy użyciu dowolnego schematu szyfrowania zabezpieczonego CPA i dowolnego bezpiecznego MAC, daje schemat szyfrowania zabezpieczony CPA, którego nie da się podrobić.
- 4.25 Niech F będzie silną permutacją pseudolosową i zdefiniuje następujący schemat szyfrowania o stałej długości: Po wprowadzeniu wiadomości $m \in \{0, 1\}^{n/2}$ i klucza $k \in \{0, 1\}^n$ algorytm Enc wybiera jednolity $r \in \{0, 1\}^{n/2}$ i oblicza $c := F_k(mr)$. (Patrz ćwiczenie 3.18.) Udowodnij, że ten schemat jest zabezpieczony CCA, ale nie jest uwierzytelnionym schematem szyfrowania.
- 4.26 Pokaż schemat szyfrowania klucza prywatnego z zabezpieczeniem CPA, którego nie da się podrobić, ale nie jest bezpieczny dla CCA.
- 4.27 Fix > 0 i liczba pierwsza p . Niech $K = Z + 1$
 $p, M = Z p, T = Z p$. Definiować
 $h: K \times M \rightarrow T$ jako

$$h(k_0, k_1, \dots, k_n) (m_1, \dots, m_n) = k_0 + \sum_{i=1}^n k_i m_i \pmod{p}$$

Udowodnić, że h jest silnie uniwersalne.

4.28 Poprawka , $n > 0$. Niech $K = \{0, 1\} \times n \times \{0, 1\}$ (interpretowane jako macierz boolowska $\times n$ i wektor -wymiarowy), niech $M = \{0, 1\}^n$ i niech $T = \{0, 1\}^n$. Zdefiniuj $h : K \times M \rightarrow T$ jako $h_{K,M}(m) = K \cdot m \equiv v$, gdzie wszystkie operacje są wykonywane modulo 2. Udowodnij, że h jest silnie uniwersalne.

4.29 Macierz Toeplitza K jest macierzą, w której $K_{i,j} = K_{i-1,j-1}$, gdy $i, j > 1$; tj. wartości wzdłuż dowolnej przekątnej są równe. Zatem macierz $\times n$ Toeplitza (dla $n > n$) ma postać

$$\begin{matrix} K_{1,1} & K_{1,2} & K_{1,3} & \cdots & K_{1,n} \\ K_{2,1} & K_{1,1} & K_{1,2} & \cdots & K_{1,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{1,n-1,1} & K_{1,n-1,2} & \cdots & K_{1,n-1,n} \end{matrix}$$

Niech $K = T$ oznacza ~~to jest~~ ^{$\times n$} macierz Toeplitza, niech $M = \{0, 1\}^n$ i niech $T = \{0, 1\}^n$. Zdefiniuj $h : K \times M \rightarrow T$ jako $h_{K,T}(m) = K \cdot m \equiv v$, gdzie wszystkie operacje wykonywane są modulo 2. Udowodnić, że h jest silnie uniwersalne. Jaka jest przewaga w porównaniu z konstrukcją z poprzedniego ćwiczenia?

4.30 Zdefiniuj odpowiednie pojęcie MAC z podwójnym zabezpieczeniem ϵ i podaj konstrukcję spełniającą Twoją definicję .

4.31 Niech $\{h_n : K_n \times \{0, 1\}^{10 \cdot n} \rightarrow \{0, 1\}^n\}_{n \in \mathbb{N}}$ będzie taki, że h_n będzie silnie uniwersalny dla wszystkich n i niech F będzie funkcją pseudolosową. (Kiedy $K_n = \{0, 1\}^{10 \cdot n}$ piszemy $h_K(\cdot)$ zamiast $h_n, K(\cdot)$). Rozważmy następujący MAC: Gen(1^n) wybiera jednorodne $K_n \in K_n$ i $k \in \{0, 1\}^n$ i wyjścia (K, k) . Aby uwierzytelnić wiadomość $m \in \{0, 1\}^{10 \cdot n}$, wybierz uniform $r \in \{0, 1\}^n$ i wprowadź $r, h_K(m) \in F_k(r)$. Weryfikacja odbywa się w sposób naturalny. Udowodnić, że daje to (obliczeniowo) bezpieczny adres MAC dla wiadomości o długości $10n$.

Machine Translated by Google

Rozdział 5

Funkcje skrótu i zastosowania

W tym rozdziale przedstawiamy kryptograficzne funkcje mieszające i odkrywamy kilka ich zastosowań. Na najbardziej podstawowym poziomie funkcja skrótu umożliwia mapowanie długiego ciągu wejściowego na krótszy ciąg wyjściowy, czasami nazywany skrótem.

Podstawowym wymaganiem jest unikanie kolizji lub dwóch danych wejściowych, które są mapowane na ten sam skrót. Odporne na kolizje funkcje skrótu mają wiele zastosowań.

Jednym z przykładów, który tu zobaczymy, jest inne podejście —ustandardyzowane jako HMAC —umożliwiające rozszerzenie domeny dla kodów uwierzytelniania wiadomości.

Poza tym funkcje skrótu stały się wszechobecne w kryptografii i czę sto są używane w scenariuszach wymagających właściwości znacznie silniejszych niż odporność na kolizje. Powszechnie stało się modelowanie kryptograficznych funkcji skrótu jako „całkowicie nieprzewidywalnych” (zwanych także losowymi wyroczniami), dlatego szczegółowo omawiamy ten model —i związane z nim kontrowersje —w dalszej części rozdziału. Dotykamy tutaj tylko kilku zastosowań modelu losowej wyroczni, ale spotkamy się z nim ponownie, gdy przejdziemy do ustawień kryptografii klucza publicznego.

Funkcje skrótu są intrugujące, ponieważ można je postrzegać jako leżące pomiędzy światami kryptografii klucza prywatnego i publicznego. Z jednej strony, jak zobaczymy w rozdziale 6, są one (w praktyce) konstruowane przy użyciu technik klucza symetrycznego, a wiele kanonicznych zastosowań funkcji skrótu ma ustawienie klucza symetrycznego. Jednakże z teoretycznego punktu widzenia istnienie odpornych na kolizje funkcji skrótu wydaje się stanowić jakościowo silniejsze założenie niż istnienie funkcji pseudolosowych (jednak słabsze założenie niż istnienie szyfrowania kluczem publicznym).

5.1 Definicje

Funkcje mieszające to po prostu funkcje, które pobierają dane wejściowe o określonej długości i kompresują je w krótkie dane wyjściowe o stałej długości. Klasyczne zastosowanie funkcji skrótu ma miejsce w strukturach danych, gdzie można ich użyć do zbudowania tabel skrótów, które umożliwiają wyszukiwanie czasu $O(1)$ podczas przechowywania zestawu elementów. W szczególności, jeśli zakres funkcji skrótu H ma rozmiar N , wówczas element x jest przechowywany w wierszu $H(x)$ tabeli o rozmiarze N . Aby odzyskać x , wystarczy obliczyć $H(x)$ i sprawdzić

ten wiersz tabeli dla przechowywanych tam elementów. „Dobrą” funkcją skrótu do tego celu jest taka, która daje niewiele kolizji, gdzie kolizją jest para różnych elementów x i x , dla których $H(x) = H(x)$; w tym przypadku mówimy również, że x i x zderzają się. (W przypadku kolizji dwa elementy zostają zapisane w tej samej komórce, co wydłuża czas wyszukiwania).

Odporne na kolizje funkcje skrótu mają podobny duch. Ponownie celem jest uniknięcie kolizji. Istnieją jednak zasadnicze różnice. Po pierwsze, chcemy zminimalizowania kolizji w ustawieniach struktur danych staje się wymogiem uniknięcia kolizji w ustawieniach kryptografii. Ponadto w kontekście struktur danych można założyć, że zbiór elementów danych wybierany jest niezależnie od funkcji skrótu i bez zamiaru powodowania kolizji. Z kolei w kontekście kryptografii mamy do czynienia z przeciwnikiem, który może wybierać elementy, mając wyraźny cel spowodowania kolizji.

Oznacza to, że odporne na kolizje funkcje skrótu są znacznie bardziej trudne do zaprojektowania.

5.1.1 Odporność na kolizje

Nieformalnie funkcja H jest odporna na kolizje, jeśli żaden probabilistyczny algorytm czasu wielomianowego nie jest w stanie znaleźć kolizji w H . Będziemy zainteresowani tylko funkcjami skrótu, których dziedzina jest wiele kroksza niż ich zakres. W tym przypadku muszą istnieć kolizje, ale takie kolizje powinny być trudne do znalezienia.

Formalnie rozważamy funkcje skrótu z kluczem. Oznacza to, że H jest funkcją z dwoma wejściami, która przyjmuje jako dane wejściowe klucz s i ciąg znaków x , a na wyjściu otrzymuje ciąg znaków $H(s, x)$.^{def} Wymaganiem jest to, że znalezienie kolizji w H dla losowo wygenerowanego klucza s musi być trudne. Istnieją co najmniej dwie różnice między kluczami w tym kontekście a kluczami, których używaliśmy do tej pory. Po pierwsze, nie wszystkie ciągi muszą koniecznie odpowiadać prawidłowym kluczom (tj. H mogą nie być zdefiniowane dla niektórych s), dlatego też klucze będą zazwyczaj generowane przez algorytm Gen , a nie wybierane jednolicie. Po drugie, co być może ważniejsze, ten klucz s nie jest (zazwyczaj) trzymany w tajemnicy i wymagana jest odporność na kolizje, nawet jeśli przeciwnik otrzyma s . Aby to podkreślić, indeksujemy górną klucz i piszemy H_s zamiast H .

DEFINICJA 5.1 Funkcja mieszająca (o długości wyjściowej n) to para probabilistycznych algorytmów wielomianowych (Gen , H) spełniających poniższe warunki:

- Gen jest algorymem probabilistycznym, który przyjmuje na wejściu parametr bezpieczeństwa 1^n i wyprowadza klucz s . Zakładamy, że 1^N jest ukryte w s .
- H przyjmuje jako dane wejściowe klucz s i ciąg znaków $x \in \{0, 1\}^n$ i wyprowadza ciąg znaków $H_s(x) \in \{0, 1\}^N$ (gdzie N jest wartością parametru bezpieczeństwa ukrytą w s).

Jeśli H_s jest zdefiniowane tylko dla wejść $x \in \{0, 1\}^n$ i $|n| > N$, to mówimy, że (Gen , H) jest funkcją mieszającą o stałej długości dla wejść o długości n . W tym przypadku H nazywamy także funkcją kompresji.

W przypadku stałej długości wymagamy, aby była ona większa niż ℓ . Zapewnia to, że funkcja kompresuje dane wejściowe. W ogólnym przypadku funkcja przyjmuje jako wejściowe ciągi znaków o dowolnej długości. Zatem kompresuje również (aczkolwiek tylko ciągi o długości większej niż $\ell(n)$). Należy zauważyć, że bez kompresji odporność na kolizje jest trywialna (ponieważ można po prostu przyjąć funkcję tożsamości $H_s(x) = x$).

Przejdźmy teraz do zdefiniowania bezpieczeństwa. Jak zwykle, najpierw definiujemy eksperyment dla funkcji skrótu $\Pi = (\text{Gen}, H)$, przeciwnika A i parametru bezpieczeństwa n : Eksperyment mający na

celu znalezienie kolizji $\text{Hash-coll}_{A,\Pi}(n)$:

1. Klucz s jest generowany poprzez uruchomienie $\text{Gen}(1n)$.
2. Przeciwnik A ma dane s i wyprowadza x, x . (Jeśli Π jest funkcją skrótu o stałej długości dla danych wejściowych o długości (n) , to potrzebujemy $x, x \in \{0, 1\}^n$).
3. Wynik eksperymentu definiuje się jako 1 wtedy i tylko wtedy, gdy $x = x$ i $H_s(x) = H_s(x)$. Mówimy wówczas, że A zaobserwował kolizję.

Definicja odporności na kolizje stwierdza, że żaden skuteczny przeciwnik nie jest w stanie tego zrobić znając kolizję w powyższym eksperymencie, chyba że z znikomym prawdopodobieństwem.

DEFINICJA 5.2 Funkcja mieszająca $\Pi = (\text{Gen}, H)$ jest odporna na kolizje, jeśli dla wszystkich probabilistycznych przeciwników A w czasie wielomianowym istnieje pomijalna funkcja negl taka, że

$$\Pr[\text{Hash-coll}_{A,\Pi}(n) = 1] = \text{negl}(n).$$

Dla uproszczenia czasami nazywamy H lub H_s „odporną na kolizje funkcją skrótu”, chociaż technicznie rzeczą biorąc, powinniśmy tylko powiedzieć, że (Gen, H) tak jest. Nie powinno to powodować żadnych nieporozumień.

Kryptograficzne funkcje skrótu zaprojektowane z wyraźnym celem zapewnienia odporności na kolizje (między innymi). W Rozdziale 6 omówimy niektóre popularne funkcje mieszające w świecie rzeczywistym. W Podrozdziale 8.4.2 zobaczymy, jak można skonstruować funkcje mieszające ze sprawdzoną odpornością na kolizje w oparciu o założenie o trudności pewnego problemu z teorii liczb.

Funkcje skrótu bez klucza. Kryptograficzne funkcje mieszające stosowane w praktyce mają zazwyczaj stałą długość wyjściową (tak jak szyfry blokowe mają stałą długość klucza) i zwykle nie są kluczowane, co oznacza, że funkcja mieszająca jest po prostu stałą funkcją $H : \{0, 1\}^n \rightarrow \{0, 1\}^m$. Jest to problematyczne z teoretycznego punktu widzenia, ponieważ dla każdej takiej funkcji zawsze istnieje algorytm działający w czasie stałym, który generuje kolizję w H : algorytm po prostu wyprowadza kolidującą parę (x, x) zakodowaną na stałe w samym algorytmie. Użycie funkcji skrótu z kluczem rozwiązuje ten problem techniczny, ponieważ nie jest możliwe zakodowanie na stałe pary kolidującej dla każdego możliwego klucza przy użyciu rozsądnej ilości miejsca (a przy ustaleniu asymptotycznym niemożliwe byłoby zakodowanie na stałe pary kolidującej dla każdej wartości parametru bezpieczeństwa).

Niezależnie od powyższego (niekluczowej) kryptograficzne funkcje skrótu używane w świecie rzeczywistym są odporne na kolizje ze wszystkich praktycznych powodów, ponieważ kolidujące pary są nieznanne (i trudne do znalezienia obliczeniowo), mimo że muszą istnieć. Dowody bezpieczeństwa niektórych konstrukcji opartych na odporności na kolizje funkcji skrótu są znacznie nawet wtedy, gdy używana jest funkcja skrótu H bez klucza, o ile dowód pokazuje, że każdy skuteczny przeciwnik „łamiący” prymityw może zostać wykorzystany do skutecznego znalezienia kolizji w H. (Wszystkie dowody w tej książce spełniają ten warunek.) W tym przypadku interpretacja dowodu bezpieczeństwa jest taka, że jeśli przeciwnik może złamać schemat w praktyce, to można go wykorzystać do znalezienia kolizji w praktyce, coś, co naszym zdaniem jest trudne do zrobienia.

5.1.2 Słabsze pojęcia bezpieczeństwa

W niektórych zastosowaniach wystarczy oprzeć się na wymaganiach bezpieczeństwa słabszych niż odporność na kolizje. Obejmują one:

- Drugi obraz wstępny lub odporność na kolizję z celem: Nieformalnie funkcja mieszająca jest odporna na drugi obraz wstępny, jeśli podano s i jednolite x. Przeciwnik ppt nie jest w stanie znaleźć $x = x$ takiego, że $Hs(x) = Hs(x)$.
- Odporność na obraz wstępny: Nieformalnie funkcja skrótu jest odporna na obraz wstępny, jeśli ma podane s i jednolite y, przeciwnik ppt nie jest w stanie znaleźć wartości x takiej, że $Hs(x) = y$. (Patrząc w przyszłość do rozdziału 7, oznacza to zasadniczo, że Hs jest jednokierunkowy.)

Każda funkcja skrótu odporna na kolizje jest również odporna na drugi obraz wstępny.

Dzieje się tak, ponieważ jeśli przy danym jednolitym x przeciwnik może znaleźć $x = x$, dla którego $Hs(x) = Hs(x)$, to może wyraźnie znaleźć kolidującą parę $x \neq x$.

Podobnie każda funkcja skrótu odporna na drugi obraz wstępny jest również odporna na obraz wstępny. Dzieje się tak dlatego, że gdyby przy danym y można było znaleźć x takie, że $Hs(x) = y$, to można by również przyjąć dane wejściowe x, obliczyć $y := Hs(x)$, a następnie otrzymać $x \neq Hs(x) = y$. Z dużym prawdopodobieństwem $x = x$ (w oparciu o fakt, że H podlega kompresji, a więc z wiele wejść jest mapowanych na ten sam wynik), w którym to przypadku znaleziono drugi obraz wstępny.

Nie definiujemy formalnie powyższych pojęć ani nie udowadniamy powyższych implikacji, ponieważ nie są one używane w dalszej części książki. Prosimy o sformalizowanie powyższego w ćwiczeniu 5.1.

5.2 Rozszerzenie domeny: transformacja Merkle'a-Damgarda

Funkcje skrótu są często konstruowane poprzez najpierw zaprojektowanie odpornej na kolizje funkcji kompresji obsługującej dane wejściowe o stałej długości, a następnie przy użyciu domeny

rozszerzenie do obsługi wejść o dowolnej długości. W tej sekcji pokazujemy jedno rozwiążanie problemu rozszerzenia domeny. Do kwestii projektowania funkcji kompresji odpornych na kolizje wracamy w podrozdziale 6.3.

Transformacja Merkle'a – Damgård jest powszechnym podejściem do rozszerzania funkcji kompresji do pełnoprawnej funkcji skrótu, przy jednoczesnym zachowaniu właściwości odporności na kolizje tej pierwszej. Jest szeroko stosowany w praktyce dla funkcji skrótu, w tym MD5 i rodzinę SHA (patrz sekcja 6.3). Istnienie tej transformacji oznacza, że projektując odporne na kolizje funkcje skrótu, możemy ograniczyć naszą uwagę do wielkości liter o stałej długości. To z kolei znacznie ułatwia projektowanie odpornych na kolizje funkcji skrótu.

Transformacja Merkle'a – Damgård jest również interesująca z teoretycznego punktu widzenia, ponieważ implikuje, że kompresja o pojedynczy bit jest tak samo łatwa (lub równie trudna) jak kompresja o dowolną wielkość.

Dla konkretności założmy, że funkcja kompresji (Gen, h) kompresuje dane wejściowe o połowę; powiedzmy, że jego długość wejściowa wynosi $2n$, a długość wyjściowa wynosi n . (Konstrukcja działa niezależnie od długości wejścia/wyjścia, o ile h kompresuje.) Konstruujemy odporną na kolizje funkcję skrótu (Gen, H) , która odwzorowuje wejścia o dowolnej długości na wyjścia o długości n . (Gen pozostaje niezmieniony.)

Transformata Merkle'a – Damgård jest zdefiniowana w Konstrukcji 5.3 i przedstawiona na rysunku 5.1. Wartość $z0$ zastosowana w kroku 2 konstrukcji, zwana wektorem inicjującym lub IV, jest dowolna i można ją zastąpić dowolną stałą.

KONSTRUKCJA 5.3

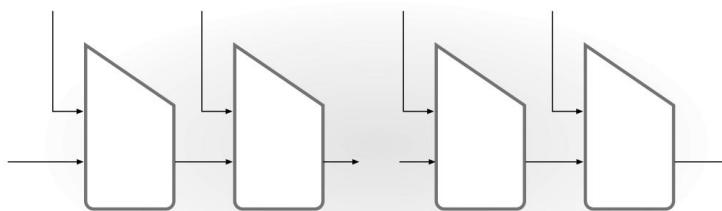
Niech (Gen, h) będzie funkcją skrótu o stałej długości dla danych wejściowych o długości $2n$ i długości wyjściowej n . Skonstruuj funkcję skrótu (Gen, H) w następujący sposób:

- Gen: pozostaje niezmieniony.
- H: na wejściu klawisz s i ciąg znaków $x \in \{0, 1\}^n$ o długości $L < 2^n$, Do
 1. Zestaw $B := \frac{L}{n}$ (tj. liczba bloków w x). Dopełnienie x zerami, tak aby jego długość była wielokrotnością n . Przeanalizuj dopełniony wynik jako sekwencję n -bitowych bloków x_1, \dots, x_B . Ustaw $xB+1 := L$, gdzie L jest zakodowane jako ciąg n -bitowy.
 2. Ustaw $z0 := 0^n$. (Nazywa się to również IV.)
 3. Dla $i = 1, \dots, B+1$, oblicz $zi := h(s \oplus xi)$.
 4. Wyjście $zB+1$.

Transformacja Merkle'a – Damgård.

TWIERDZENIE 5.4 Jeśli (Gen, h) jest odporne na kolizje, to odporne jest także (Gen, H) .

DOWÓD Pokazujemy, że dla dowolnego s kolizja w Hs skutkuje kolizją w hs . Niech $x \in \{0, 1\}^n$ i $s \in \{0, 1\}^n$. Dla dowolnych różnych ciągów o długości odpowiadających L i L' , takimi, że $Hs(x) = Hs(x')$. Niech x_1, \dots, x_B i $x'_1, \dots, x'_{B'}$ będą ciągami o długościach L i L' odpowiednio dla x i x' . Wtedy $hs = h(s \oplus x_1 \oplus \dots \oplus x_B) = h(s \oplus x'_1 \oplus \dots \oplus x'_{B'}) = hs'$.



RYSUNEK 5.1: Transformacja Merkle'a - Damgarda.

niech x_1, \dots, x_B będą dą blokami B wyściełanego x . Przypomnijmy, że $xB+1 = L$ i L . Należy rozważyć dwa przypadki: $xB+1$

1. Przypadek 1: $L = L$. W tym przypadku ostatnim krokiem obliczenia $Hs(x)$ jest $zB+1 := hs(zBL)$, a ostatnim krokiem obliczenia $Hs(x)$ jest $z := hs(zBL)$. Ponieważ $Hs(x) = Hs(x)$ wynika, że $hs(zBL) = hs(zBL)$. Jednakże $L = L$, a więc zBL i zBL to dwa różne ciągi, które zderzają się pod hs

2. Przypadek 2: $L = L$. Oznacza to, że $B = B$. Niech z_0, \dots, z_{B+1} będą wartościami zdefiniowanymi podczas obliczania $Hs(x)$, niech i i $\stackrel{\text{def}}{=} z_{i-1}x_i$ oznaczają wprowadzimy do h ~~s~~^{def} z_i w ~~z~~^{def} z_{i+1} z_{i+2} ... z_{B+1} indeksem, dla którego $IN = IN$. Ponieważ $|x| = |x|$ ale $x = x$ istnieje i $z_i = x$, więc takie N z pewnością istnieje. Ponieważ

$$IB+2 = zB+1 = H \quad s(x) = H. s(x) = z \quad B+1 = ja_{B+2},$$

mamy $N = B + 1$. Przez maksymalizację N mamy $IN+1 = I$ konkretny zN_{N+1} i w $= zN$. Ale to oznacza że $IN = I_N$ to zderzenie w hs

Pozostawiamy to jako ćwiczenie polegające na przekształceniu powyższego w formalną redukcję . ■

5.3 Uwierzytelnianie wiadomości za pomocą funkcji skrótu

W poprzednim rozdziale przedstawiliśmy dwie konstrukcje kodów uwierzytelniających wiadomości dla wiadomości o dowolnej długości. Pierwsze podejście było ogólne, ale nieskuteczne. Drugi, CBC-MAC, opierał się na funkcjach pseudolosowych. Tutaj zobaczymy inne podejście, które nazywamy „hash-and-MAC”, które opiera się na odpornym na kolizje mieszaniu wraz z dowolnym kodem uwierzytelniającym wiadomość. Następnie omawiamy znormalizowaną i powszechnie stosowaną konstrukcję zwaną HMAC, którą można postrzegać jako specyfczną realizację tego podejścia.

5.3.1 Hash i MAC

Idea stojąca za podejściem hash-and-MAC jest prosta. Najpierw dowolnie dłużą wiadomość m jest skróty dołańca $Hs(m)$ o stałej długości przy użyciu odpornej na kolizje funkcji skrótu. Następnie do wyniku stosowany jest adres MAC (o stałej długości). Formalny opis można znaleźć w części Konstrukcja 5.5.

KONSTRUKCJA 5.5

Niech $\Pi = (\text{Mac}, \text{Vrfy})$ będzie MAC dla komunikatów o długości (n) i niech $\Pi_H = (\text{Gen}_H, H)$ będzie funkcją skrótu o długości wyjściowej (n) . Skonstruuj MAC $\Pi = (\text{Gen}, \text{Mac}, \text{Vrfy})$ dla komunikatów o dowolnej długości w następujący sposób:

- Gen : na wejściu $1n$ wybierz uniform $k \in \{0, 1\}^n$ i uruchom $\text{Gen}_H(1n)$, aby otrzymać s ; kluczem jest $k := k, s$.
- Mac : na wejściu klawisz k, s i wiadomość $m \in \{0, 1\}^*$, wyjście $t = \text{Mack}(Hs(m))$.
- Vrfy : na wejściu klawisz k, s , wiadomość $m \in \{0, 1\}^*$, i MAC znacznik t , wyjście 1 wtedy i tylko wtedy, gdy $\text{Vrfyk}(Hs(m), t) \stackrel{?}{=} 1$.

Paradygmat hash i MAC.

Konstrukcja 5.5 jest bezpieczna, jeśli Π jest bezpiecznym adresem MAC dla wiadomości o stałej długości i (Gen, H) jest odporne na kolizje. Intuicyjnie, ponieważ funkcja skrótu jest odporna na kolizje, uwierzytelnianie $Hs(m)$ jest tak samo dobre, jak uwierzytelnianie samego m : jeśli nadawca może zapewnić, że odbiorca otrzyma prawidłową wartość $Hs(m)$, odporność na kolizje gwarantuje, że atakujący nie będzie mógł znaleźć inny komunikat m' , który ma skrót do tej samej wartości. Nieco bardziej formalnie, założymy, że nadawca używa konstrukcji 5.5 do uwierzytelnienia pewnego zestawu wiadomości Q , a atakujący A może następnie sfałszować prawidłowy znacznik w nowej wiadomości $m' \in Q$. Istnieją dwa możliwe przypadki:

Przypadek 1: istnieje komunikat $m \in Q$ taki, że $Hs(m) = Hs(m')$. Wtedy A ma, co jest sprzeczne z odpornością na kolizję (Gen, H) . stwierdził kolizję w Hs

Przypadek 2: dla każdej wiadomości $m \in Q$ zachodzi zasada, że $Hs(m) \neq Hs(m')$. Niech $Hs(Q) \stackrel{\text{def}}{=} \{Hs(m) \mid m \in Q\}$. Następnie $Hs(m) \neq Hs(m') \in Hs(Q)$. W tym przypadku A sfałszował ważny znacznik na „nowej wiadomości” $Hs(m')$ w odniesieniu do kodu uwierzytelniającego wiadomości o stałej długości Π . Jest to sprzeczne z założeniem, że Π jest bezpiecznym adresem MAC.

Zamienimy teraz powyższy dowód na dowód formalny.

TWIERDZENIE 5.6 Jeśli Π jest bezpiecznym MAC dla wiadomości o dowolnej długości, a Π_H jest odporny na kolizje, to Konstrukcja 5.5 jest bezpiecznym MAC (dla wiadomości o dowolnej długości).

DOWÓD Niech Π oznacza Konstrukcję 5.5 i niech A będzie zdecydomy przeciwnikiem atakującym Π . W wykonaniu eksperymentu $\text{Mac-forgeA}, \Pi(n)$ niech $k = k$, s oznacza klucz MAC, niech Q oznacza zbiór wiadomości, których znaczniki zostały ponownie i niech (m, t) będzie taki, że $m \in Q$. Końcowym wyjściem A jest $\text{Zakładamy bez pytania przez } A \text{ utratę ogólności, Zdefiniuj coll jako zdarzenie, które w eksperymencie } (n) \text{ istnieje } m \in Q, \text{ dla którego } Hs(m) = Hs(m)$.

$$\begin{aligned} & \Pr[\text{Mac-forgeA}, \Pi(n) = 1] \\ &= \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \mid \text{coll}] + \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \mid \text{not coll}] \quad \text{st.} \quad (5.1) \\ &= \Pr[\text{coll}] + \Pr[\text{Mac-forgeA}, \Pi(n) = 1 \mid \text{not coll}] \end{aligned}$$

Pokazujemy, że oba wyrazy w równaniu (5.1) są nieistotne, co kończy dowód. Intuicyjnie pierwszy człon jest pomijalny ze względu na odporność na kolizję Π , a drugi człon jest pomijalny ze względu na bezpieczeństwo Π .

Rozważmy następujący algorytm C znajdujący kolizję w Π :

Algorytm C:

Algorytm otrzymuje na wejściu s (z ukrytym n).

- Wybierz jednorodność $k \in \{0, 1\}$
- n. • Uruchom $A(1n)$. Kiedy A żąda znacznika w i -tej wiadomości $m_i \in \{0, 1\}$, oblicz $t_i = Mack(Hs(m_i))$ i podaj t_i do A . • Gdy A wyprowadza (m, t) , to jeśli istnieje i dla którego $Hs(m) = Hs(m_i)$, moc wyjściowa (m, t, mi) .

Jest oczywiste, że C przebiega w czasie wielomianowym. Przeanalizujmy jego zachowanie. Kiedy dane wejściowe do C są generowane poprzez uruchomienie $\text{GenH}(1n)$ w celu uzyskania s , widok A po uruchomieniu jako podprogram przez C jest rozkładany identycznie jak widok A w eksperymencie $\text{Mac-forgeA}, \Pi(n)$. W szczególności znaczniki nadane A przez C mają taki sam rozkład jak znaczniki, które A otrzymuje w $\text{Mac-forgeA}, \Pi(n)$. Ponieważ C generuje kolizję dokładnie w momencie wystąpienia kolizji, mamy

$$\Pr[\text{Hash-collC}, \Pi(n) = 1] = \Pr[\text{coll}].$$

Ponieważ Π jest odporny na kolizje, dochodzimy do wniosku, że $\Pr[\text{coll}]$ jest nieistotny.

Przejźmy teraz do udowodnienia, że drugi człon równania (5.1) jest pomijalny. Rozważmy następującego przeciwnika A atakującego Π w $\text{Mac-forgeA}, \Pi(n)$:

Przeciwnik A:

Przeciwnik otrzymuje dostęp p do wyroczni MAC $Mack(\cdot)$.

- Oblicz $\text{GenH}(1n)$, aby otrzymać s .
- Uruchom $A(1n)$. Kiedy A żąda znacznika w i -tej wiadomości $m_i \in \{0, 1\}$ wtedy: (1) oblicz $\hat{m}_i := Hs(m_i)$; (2) uzyskać znacznik t_i na \hat{m}_i od wyroczni MAC; i (3) dać t_i do A.
- Gdy A wyprowadza (m, t) , to wyprowadza $(Hs(m), t)$.

Oczywiście A przebiega w czasie wielomianowym. Rozważmy eksperyment Mac-forgeA, $\Pi(n)$. W tym eksperymencie widok A, gdy jest uruchamiany jako podprogram przez A, rozkłada się identycznie jak widok w eksperymencie Mac-forgeA, $\Pi(n)$. Co więcej, gdy $(n) = 1$ i coll nie Mac-forgeA, Π . (W takim przypadku t jest prawidłowym znacznikiem na Hs(m)) w schemacie Π w odniesieniu do k. Fakt, że coll nie wystąpił, oznacza, że Hs(m) nigdy nie był zadany przez A swojej własnej wyroczni MAC, więc to jest rzeczywiście fałszerstwem.) Dlatego też

$$\Pr[\text{Mac-forgeA}, \Pi(n) = 1] = \Pr[\text{Mac-forgeA}, \Pi(n) \neq \overline{\text{col}}],$$

a bezpieczeństwo Π oznacza, że pierwsze prawdopodobieństwo jest znikome. To kończy dowód twierdzenia. ■

5.3.2 HMAC

Wszystkie konstrukcje kodów uwierzytelniających wiadomości, które widzieliśmy do tej pory, ostatecznie opierają się na jakimś szyfrze blokowym. Czy możliwe jest skonstruowanie bezpiecznego adresu MAC (dla wiadomości o dowolnej długości) w oparciu bezpośrednio o funkcję skrótu? Pierwszą myślą może być zdefiniowanie Mack(m) = H(km); możemy się spodziewać, że jeśli H jest „dobrą” funkcją skrótu, atakującemu powinno być trudno przewidzieć wartość H(km) biorąc pod uwagę wartość H(km), dla dowolnego m = m, zakładając, że k wynosi wybrany losowo (i nieznany atakującemu). Niestety, jeśli H jest skonstruowany przy użyciu transformaty Merkle'a-Damgård - jak ma to miejsce w przypadku wiele kogości funkcji skrótu w świecie rzeczywistym - wówczas MAC zaprojektowany w ten sposób jest całkowicie niepewny, co zostanie pokazane w ćwiczeniu 5.10.

Zamiast tego możemy spróbować użyć dwóch warstw mieszania. Patrz Konstrukcja 5.7 dla ustandaryzowanego schematu zwany HMAC oparty na tym pomyśle.

KONSTRUKCJA 5.7

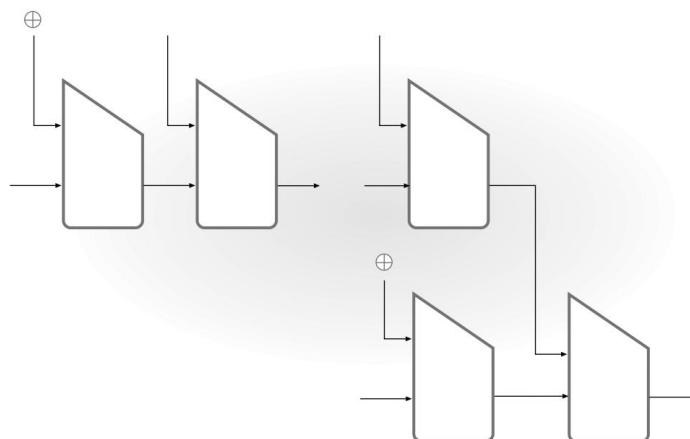
Niech (GenH, H) będzie funkcją skrótu skonstruowaną poprzez zastosowanie transformaty Merkle'a - Damgård do funkcji kompresji (GenH, h) przyjmującej dane wejściowe o długości $n + n$. (Zobacz tekst.) Niech opad i ipad będą stałymi stałymi o długości n. Zdefiniuj MAC w następujący sposób:

- sposób: • Gen: na wejściu n oblicz GenH(n), aby uzyskać klucz s. Wybierz także uniform k $\{0, 1\}^n$. Wyprowadź klucz s, k. • Mac:

na wejściu klawisz s, k i wiadomość m $\{0, 1\}^*$, wyjście

$$t := H^s(k \oplus \text{opad}) H^{s^s}(k \oplus \text{iPad}) m.$$

- Vrfy: na wejściu klawisz s, k, wiadomość m $\{0, 1\}^*$ wyjście , i znacznik t, 1 wtedy i tylko wtedy, gdy $t \stackrel{?}{=} H^s(k \oplus \text{opad}) H^{s^s}(k \oplus \text{iPad}) m$.



RYSUNEK 5.2: HMAC, obrazowo.

Dlaczego powinniśmy mieć pewność, że HMAC jest bezpieczny? Jednym z powodów jest to, że możemy postrzegać HMAC jako specyficzną instancję paradygmatu mieszania i MAC z poprzedniej sekcji. Aby to zobaczyć, zajrzymy „pod maskę”, co dzieje się po uwierzytelnieniu wiadomości; patrz rysunek 5.2. Musimy także dokładniej określić parametry i nieco bardziej szczegółowo opisać sposób, w jaki transformacja Merkle'a – Damgard'a jest realizowana w praktyce.

Say (GenH, H) jest skonstruowany w oparciu o funkcję kompresji (GenH, h) , w której h odwzorowuje dane wejściowe o długości $n + n$ na wyjścia o długości n (gdzie formalnie n jest funkcją n). Kiedy opisywaliśmy transformację Merkle'a – Damgard'a w podrozdziale 5.2, założyliśmy, że $n = n$, ale nie zawsze tak musi być.

Powiedzieliśmy również, że długość haszowanej wiadomości została zakodowana jako dodatkowy blok wiadomości dołączany do wiadomości. W praktyce długość jest zamiast tego kodowana w części bloku przy użyciu n bitów. Oznacza to, że obliczenia $\text{Hs}(x)$ rozpoczynają się od dopełnienia x zerami do ciągu o długości dokładnie mniejszej niż wielokrotność n ; nastepnie dołącza długość $L = |x|$, zakodowaną przy użyciu dokładnie bitów. Hash wynikowej sekwencji n -bitowych bloków x_1, \dots jest następnie obliczany jak w Konstrukcji 5.3. Założymy, że $n + n$. Oznacza to w szczególności, że jeśli zahasujemy dane wejściowe x o długości $n + n$, wówczas dopełniony wynik (łącznie z długością) będzie miał długość dokładnie $2n$ bitów. Dowód Twierdzenia 5.4 pokazujący, że (GenH, H) jest odporny na kolizje, jeśli (GenH, h) jest odporny na kolizje, pozostaje niezmieniony.

Wracając do HMAC i patrząc na rysunek 5.2, widzimy, że ogólna forma HMAC polega na mieszaniu wiadomości o dowolnej długości do krótkiego ciągu $y = \text{Hs}(\text{k} \oplus \text{ipad} \text{m})$, a następnie obliczamy (tajnie zapisaną) funkcję $\text{Hs}((\text{k} \oplus \text{opad}) \text{y})$ wyniku. Ale możemy powiedzieć więcej o tej. Należy najpierw zauważyć, że obliczenia „wewnętrzne”.

$$\text{Hs}(\text{m}) \stackrel{\text{def}}{=} \text{Hs}((\text{k} \oplus \text{iPad}) \text{m})$$

jest odporny na kolizje (zakładając, że h jest), dla dowolnej wartości k \oplus pad. Ponadto pierwszym krokiem w „zewnętrznym” obliczeniu $Hs((k \oplus \text{pad}) \oplus y)$ jest obliczenie wartości $kout = hs(\text{IV}(k \oplus \text{pad}))$. Następnie obliczamy $hs(kout \oplus y)$, gdzie y odnosi się do dopełnionej wartości y (tj. włączając długość $(k \oplus \text{pad}) \oplus y$, która zawsze wynosi $n + n$ bitów, zakodowanych przy użyciu dokładnie bitów). Zatem, jeśli traktujemy $kout$ jako jednolite – poniżej będą dziemy bardziej formalni – i założymy to

$$\text{Mac } k(y) \stackrel{\text{def}}{=} \text{godz. } s(k \oplus y) \quad (5.2)$$

jest bezpiecznym adresem MAC o stałej długości, wówczas HMAC można postrzegać jako instancję podejścia opartego na mieszaniu i MAC z

$$\text{HMAC}, k(m) = \text{Mac} \quad kout(H^{\sim} \oplus s(m)) \quad (5.3)$$

(gdzie $kout = h(s(\text{IV}(k \oplus \text{pad})))$). Ze względu na sposób, w jaki zwykle projektowana jest funkcja kompresji h (patrz sekcja 6.3.1), rozsądne jest założenie, że Mac jest bezpiecznym MAC o stałej długości.

Role iPada i Opada. Biorąc powyższe pod uwagę, można się zastanawiać, dlaczego konieczne jest uwzględnienie k w „zewnętrznym” obliczeniu $Hs((k \oplus \text{pad}) \oplus m)$.

(W szczególności, aby podejście hash-and-MAC było bezpieczne, w pierwszym kroku wymagana jest odporność na kolizje, która nie wymaga żadnego tajnego klucza). Powodem jest to, że pozwala to na oparcie bezpieczeństwa HMAC na potencjalnie słabsze założenie, że (GenH, H) jest słabo odporne na kolizje, gdzie słabą odporność na kolizje definiuje się za pomocą następującego eksperymentu: klucz s jest generowany przy użyciu GenH i wybierany jest jednolity sekretny krewny $\{0, 1\}^n$. Następnie przeciwnik może wejść w interakcję z „wyrocznią mieszającą”, która zwraca jego krewnych (m) w odpowiedzi na zapytanie m , gdzie $Hs(kin \oplus m)$ odnosi się do obliczenia Hs przy użyciu transformaty Merkle'a-Damgård'a zastosowanej do hs , ale przy użyciu sekretnej wartości kin jako IV. (Patrz ponownie rysunek 5.2.) Przeciwnik odnosi sukces, jeśli może wyprowadzić odrebną $bny(m)$ takie, że $Hs(kin \oplus m) = Hs(kin \oplus \text{crewy})$ i mówimy, że (GenH, H) ma wartości m, m' słabo odporną na kolizje, jeśli każdy punkt A powiedzie się w tym eksperymencie z jedynie znikomym prawdopodobieństwem. Jeśli (GenH, H) jest odporny na kolizje, to jest wyraźnie odporny na kolizje; ten ostatni jest jednak warunkiem słabszym i potencjalnie łatwiejszym do spełnienia. To dobry przykład solidnej inżynierii bezpieczeństwa. Ta defensywna strategia projektowania opłaciła się, gdy odkryto, że funkcja skrótu MD5 (patrz sekcja 6.3.2) nie jest odporna na kolizje. Ataki wykrywające kolizje na MD5 nie naruszyły słabej odporności na kolizje, a HMAC-MD5 nie został uszkodzony, mimo że MD5 tak. Dało to programistom czas na zastąpienie MD5 we wdrożeniach HMAC, bez bezpośredniej obawy przed atakiem. (Mimo to, HMAC-MD5 nie powinien być już używany, ponieważ znane są słabe strony MD5.)

Powyższa dyskusja sugeruje, że w obliczeniach zewnętrznych i wewnętrznych należy używać niezależnych kluczy. Ze względu na wydajność w HMAC używany jest pojedynczy klawisz k , ale klucz ten jest używany w połączeniu z iPadem i Opadem w celu uzyskania

dwa inne klucze. Definiować

$$G_s(k) \stackrel{\text{def}}{=} \text{godz}^s IV(k \quad \text{opad}) \text{ godz}^s IV(k \quad \text{ipad}) = \text{kout kin.} \quad (5.4)$$

Jeśli założymy, że G_s jest generatorem pseudolosowym dla dowolnego s , to kout i kin można traktować jako niezależne i jednolite klucze, gdy k jest jednorodne. Bezpieczeństwo HMAC sprowadza się następuje do bezpieczeństwa następującej konstrukcji:

$$\text{Mac}, \text{kin}, \text{kout}(m) = \text{godz}^s \text{kout } Hs \text{ kin } (M).$$

(Porównaj z równaniem (5.3).) Jak zauważono wcześniej, można udowodnić, że konstrukcja ta jest bezpieczna (przy użyciu wariantu dowodu dla podejścia hash-and-MAC), jeśli H jest słabo odporny na kolizje, a MAC jest zdefiniowany w równaniu (5.2) to bezpieczny adres MAC o stałej długości.

TWIERDZENIE 5.8 Założmy, że G_s zdefiniowany w Równaniu (5.4) jest generatorem pseudolosowym dla dowolnego s , MAC zdefiniowany w Równaniu (5.2) jest bezpiecznym MAC o stałej długości dla komunikatów o długości n , a (GenH, H) jest słabo odporny na kolizje. Zatem HMAC jest bezpiecznym adresem MAC (dla wiadomości o dowolnej długości).

HMAC w praktyce. HMAC jest standardem branżowym i jest szeroko stosowany w praktyce. Jest bardzo wydajny i łatwy do wdrożenia, a także jest poparty dowodem bezpieczeństwa opartym na założeniach, które uważa się za przydatne w przypadku praktycznych funkcji skrótu. Znaczenie HMAC wynika częściowo z aktualności jego pojawienia się. Przed wprowadzeniem HMAC wielu praktyków odmawiało stosowania CBC-MAC (twierdząc, że jest „zbyt wolne”) i zamiast tego stosowało niepewne konstrukcje heurystyczne. HMAC zapewnił ustalony i bezpieczny sposób uwierzytelniania wiadomości w oparciu o funkcje skrótu.

5.4 Ogólne ataki na funkcje mieszające

Jakie jest najlepsze bezpieczeństwo, jakie może zapewnić funkcja skrótu H ? Zbadamy to pytanie, pokazując dwa ataki, które są ogólne w tym sensie, że mają zastosowanie do dowolnych funkcji skrótu. Istnienie tych ataków implikuje dolne ograniczenia długości wyjściowej H potrzebnej do osiągnięcia pożądanego poziomu bezpieczeństwa, a zatem ma ważne konsekwencje praktyczne.

5.4.1 Ataki urodzinowe w celu znalezienia kolizji

Niech $H : \{0, 1\}^n \rightarrow \{0, 1\}^n$ być dziesiątką funkcją skrótu. (W tym miejscu i w pozostałej części rozdziału pominiemy wyraźną wzmiankę o kluczu skrótu s , ponieważ nie jest on bezpośrednio istotny. Można również postrzegać klucze s jako generowane i naprawiane przed nimi)

stosowane są algorytmy.) Istnieje trywialny atak polegający na wykryciu kolizji przeprowadzany w czasie $O(2)$: po prostu oceń H na $2 + 1$ różnych danych wejściowych; zgodnie z zasadą szufladki dwa wyjścia muszą być równe. Czy możemy zrobić lepiej?

Uogólniając powyższy algorytm, powiedzmy, że wybieramy q różnych wejść x_1, \dots, x_q , oblicz $y_i := H(x_i)$ i sprawdź, czy którakolwiek z dwóch wartości y_i jest równa. Jakie jest prawdopodobieństwo, że ten algorytm znajdzie kolizję? Jak już powiedzieliśmy, jeśli $q > 2$, to do zderzenia dochodzi z prawdopodobieństwem 1. Jakie jest prawdopodobieństwo zderzenia, gdy q jest mniejsze? Dokładna analiza tego prawdopodobieństwa jest dość trudna, dlatego zamiast tego przeanalizujemy wyidealizowany przypadek, w którym H jest traktowane jako funkcja losowa.¹ Oznacza to, że dla każdego i zakładamy, że wartość $y_i = H(x_i)$ ma rozkład jednostajny w $\{0, 1\}$ i niezależnie od poprzednich wartości wyjściowych $\{y_j\}_{j < i}$ (przypomnijmy, że zakładamy, że wszystkie $\{x_i\}$ są różne). W ten sposób zredukowaliśmy nasz problem do następującego: jeśli wybierzymy wartości $y_1, \dots, y_q \in \{0, 1\}$ równomiernie losowo, jakie jest prawdopodobieństwo, że istnieją różne i, j takie, że $y_i = y_j$?

Problem ten został szeroko zbadany i jest powiązany z tzw. problemem urodzin, szczegółowo omówionym w Załączniku A.4. Z tego powodu opisany przez nas algorytm wyszukiwania kolizji jest często nazywany atakiem urodzinowym.

Problem urodzin jest następny: jeśli w pokoju znajduje się q osób, jakie jest prawdopodobieństwo, że dwie z nich obchodzą urodziny tego samego dnia? (Załóżmy, że urodziny są równomiernie i niezależnie rozłożone na 365 dni roku nieprzestępne.) Jest to dokładnie analogiczne do naszego problemu: jeśli y_i reprezentuje urodziny osoby i , to mamy $y_1, \dots, y_q \in \{1, \dots, 365\}$ wybrane jednolicie, a pasujące daty urodzin odpowiadają odrębnym i, j , gdzie $y_i = y_j$ (tzn. pasujące daty urodzin odpowiadają kolizjom).

W dodatku A.4 pokazujemy, że dla y_1, \dots, y_q wybrane jednolicie w $\{1, \dots, N\}$, prawdopodobieństwo zderzenia wynosi w przybliżeniu $1/2$, gdy $q = \Theta(N^{1/2})$. W przypadku urodzin, gdy są tylko 23 osoby, prawdopodobieństwo, że jakieś dwie z nich będą obchodzić urodziny tego samego dnia, jest więcej niż $1/2$. W naszym ustawieniu oznacza to, że gdy funkcja mieszająca ma długość wyjściową (a zatem zakres ma rozmiar 2), wówczas przyjęto, że $q = \Theta(2^{1/2})$ daje kolizję z prawdopodobieństwem w przybliżeniu $1/2$.

Z konkretnego punktu widzenia bezpieczeństwa powyższe oznacza, że aby funkcja mieszająca była w stanie oprzeć się atakom mającym na celu wykrycie kolizji, które działają w czasie T (gdzie poświęcamy czas na ocenę H jako naszej jednostki czasu), długość wyjściowa funkcji skrótu funkcja musi mieć co najmniej $2 \log T$ bitów (ponieważ $2(2 \log T)/2 = T$). Biorąc pod uwagę określone parametry, oznacza to, że jeśli chcemy, aby znalezienie kolizji było tak trudne, jak wyczerpujące przeszukiwanie kluczy 128-bitowych, to długość wyjściowa funkcji mieszającej musi wynosić co najmniej 256 bitów. Podkreślamy, że posiadanie tak długiego produktu jest jedynie warunkiem koniecznym, a nie wystarczającym. Zauważamy również, że ataki urodzinowe działają tylko w celu znalezienia kolizji. Nie ma ataków ogólnych

¹ Można wykazać, że jest to (w zasadzie) najgorszy przypadek, a kolizje występują z większym prawdopodobieństwem, jeśli H różni się od losowego, a $\{x_i\}$ są wybrane równomiernie.

dla drugiej odporności na obraz wstę pny lub odporności na obraz wstę pny funkcji skrótu H, które wymagają mniej niż 2 ocen H (chociaż patrz sekcja 5.4.3).

Znajdowanie znaczących kolizji. Opisany właśnie atak urodzinowy powoduje kolizję , która niekoniecznie jest bardzo przydatna. Ale ten sam pomysł można zastosować również do znalezienia „znaczących” kolizji. Założmy, że Alicja chce znaleźć dwie wiadomości x i x takie, że $H(x) = H(x')$, a ponadto x powinien być listem od pracodawcy wyjaśniającym, dlaczego została wyrzucona z pracy, natomiast x powinien być pochlebnym listem polecającym. (Może to pozwolić Alicji na sfałszowanie odpowiedniego znacznika w liście polecającym, jeśli jej pracodawca używa do uwierzytelniania wiadomości metody hash-and-MAC). Zaobserwowano, że atak urodzinowy wymaga jedynie danych wejściowych skrótu x_1, \dots, x_q być odrę bny; nie muszą być przypadkowe. Alicja może przeprowadzić atak typu urodzinowego, generując $q = \Theta(2/2)$ komunikatów pierwszego typu i q komunikatów drugiego typu, a następnie szukając kolizji pomiędzy komunikatami obu typów. Niewielka zmiana w analizie z Załącznika A.4 pokazuje, że daje to kolizję pomiędzy komunikatami różnych typów z prawdopodobieństwem w przybliżeniu 1/2.

Krótki namysł pokazuje, że łatwo jest napisać tę samą wiadomość na wiele różnych sposobów. Weźmy na przykład pod uwagę następujące kwestie:

Trudno/trudne/trudne/trudne/niemożliwe jest wyobrażenie/uwierzenie, że znajdziemy/zlokalizujemy/zatrudnimy innego pracownika/osobę o podobnych zdolnościach/umiejętnościach/charakterze jak Alicja. Wykonała świetną/super robotę .

Możliwa jest dowolna kombinacja słów zapisanych kursywą i wyraża ona ten sam punkt. Zatem zdanie można zapisać na $4,2,3,2,3,2 = 288$ różnych sposobów.

To tylko jedno zdanie, więc w zasadzie łatwo jest wygenerować komunikat, który można przepisać na 264 różne sposoby — wystarczą 64 słowa z jednym synonimem każde. Alicja może przygotować 2/2 listów wyjaśniających, dlaczego została zwolniona, i kolejne 2/2 listów polecających; z dużym prawdopodobieństwem zostanie stwierdzona kolizja pomiędzy obydwoema typami liter.

5.4.2 Ataki urodzinowe na małej przestrzeni

Opisane powyżej ataki urodzinowe wymagają dużej ilości pamięci; w szczególności wymagają od atakującego przechowywania wszystkich wartości $O(q) = O(2/2) \{y_i\}$, ponieważ atakujący nie wie z góry, która para wartości spowoduje kolizję . Jest to znacząca wada, ponieważ pamięć jest na ogół zasobem rzadszym niż czas. Prawdopodobnie trudniej jest przydzielić i zarządzać pamięcią dla 260 bajtów niż wykonać 260 instrukcji procesora. Co więcej, zawsze można pozwolić, aby obliczenia wykonywały się w nieskończoność, podczas gdy wymagania algorytmu dotyczące pamięci muszą zostać spełnione, gdy tylko taka ilość pamięci będzie potrzebna.

Pokazujemy tutaj lepszy atak urodzinowy z drastycznie zmniejszonymi wymaganiami dotyczącymi pamięci. W rzeczywistości ma podobną złożoność czasową i prawdopodobieństwo sukcesu jak poprzednio, ale wykorzystuje tylko stałą pamięć. Atak rozpoczyna się od wybrania a

losową wartość x_0 , a następnego obliczenie $x_i := H(x_{i-1})$ i $x_{2i} := H(H(x_{2(i-1)))$ dla $i = 1, 2, \dots$ (Zauważ, że $x_i = H(i)(x_0)$ dla wszystkich i , gdzie $H(i)$ odnosi się do i -krotnej iteracji H .) W każdym kroku porównywane są wartości x_i i x_{2i} ; jeśli są równe, to gdzieś w ciągu x_0, x_1, \dots, x_{2n}

Następnie algorytm znajduje najmniejszą wartość j , dla której $x_j = x_{j+1}$ (zauważ, że $j < i$, ponieważ $j = i$ działa) i wyprowadza x_{j-1}, x_{j+1} jako kolizję. Atak ten, opisany formalnie jako Algorytm 5.9 i przeanalizowany poniżej, wymaga jedynie przechowywania dwóch wartości skrótu w każdej iteracji.

ALGORYTM 5.9 Urodzinowy

atak na małą przestrzeń

Dane wejściowe: funkcja skrótu $H : \{0, 1\} \rightarrow \{0, 1\}$

Wynik: Odrębnie $bne x, x \in H(x) = H(x) \neq x$

```

x := x := x0
dla i = 1, 2, ...
    zrób: x :=
        H(x) x :=
            H(H(x)) // teraz x = H(i)(x0) i x = H(2i)(x0)
            jeśli x = x przerwa
        x := x, x := x0
    dla j = 1 do i:
        jeśli H(x) = H(x) zwróć x, x i
        zatrzymaj inaczej x :=
            H(x), x := H(x) // teraz x = H(j)(x0) i x = H(i+j)(x0)
```

Ile iteracji pierwszej pełni spodziewamy się przed $x = x$? Rozważmy, gdzie $x_i = H(i)(x_0)$, jak modelujemy H jako funkcję losową, zdefiniowaną wcześniej. Jeśli ciąg wartości x_1, x_2, \dots , następna każda z tych wartości jest równomiernie i niezależnie rozkładana w $\{0, 1\}$, aż do wystąpienia pierwszego powtórzenia. Zatem spodziewamy się, że powtórzenie nastąpi z prawdopodobieństwem $1/2$ w pierwszych wyrazach $q = \Theta(2/2)$ ciągu. Pokazujemy, że gdy w pierwszych q elementach występuje powtórzenie, algorytm znajduje powtórzenie w co najwyżej q iteracjach pierwszej pełni:

Twierdzenie 5.10 Niech x_1, \dots, x_q będą częścią ciągiem wartości, gdzie $x_m = H(x_{m-1})$.

Jeśli $x_I = x_J$ gdzie $1 \leq I < J \leq q$, to istnieje $i < J$ takie, że $x_i = x_{2i}$.

DOWÓD Ciąg x_I, x_{I+1}, \dots powtarza się z okresem l , dla wszystkich $i \geq I - l$. To i $k \geq 0$ zachodzi zasada, że $x_i = x_{i+k+l}$. Niech i będzie najmniejszą wielokrotnością l , która jest również większa lub równa I . Mamy $i < J$, ponieważ ciąg wartości $x_I, x_{I+1}, \dots, x_{J-1} = J - l$ zawiera wielokrotność l .

Ponieważ $i \geq I$ oraz $2i - l = i$ jest wielokrotnością l , wynika z tego, że $x_i = x_{2i}$. ■

Zatem, jeśli w ciągu x_1 występuje powtarzająca się wartość \dots, x_q , to istnieje pewne $i < q$, dla którego $x_i = x_{2i}$. Ale wtedy w iteracji i naszego algorytmu mamy $x = x$ i algorytm wychodzi z pierwszej pętli. W tym momencie algorytmu wiemy, że $x_i = x_{i+i}$. Algorytm następuje ustawiając $x := x (= x_i)$ i $x := x_0$ i przystępuje do znajdowania najmniejszego $j \geq 0$, dla którego $x_j = x_{j+i}$.

(Zauważ, że $j = 0$, ponieważ $|x_0| = +1$). Daje $x_j = 1, x_{j+i} = 1$ jako kolizję.

Znajdowanie znaczących kolizji. Opisany właśnie algorytm może nie nadawać się do znajdowania znaczących kolizji, ponieważ nie ma kontroli nad próbowanymi elementami. Niemniej jednak pokazujemy, jak możliwe jest znalezienie znaczących kolizji. Sztuka polega na tym, aby znaleźć kolizję we właściwej funkcji!

Załóżmy, tak jak poprzednio, że Alicja chce znaleźć kolizję między wiadomościami dwóch różnych „typów”, np. listem wyjaśniającym, dlaczego Alicja została zwolniona, i pochlebnym listem polecającym, których skrót ma tę samą wartość. Następnie Alicja zapisuje każdą wiadomość tak, aby w każdej znajdowało się - 1 wymienne słowa; tj. istnieje 2-1 komunikatów każdego typu. Zdefiniuj funkcję jeden do jednego $g : \{0, 1\}^n \rightarrow \{0, 1\}^n$ tak, że th bit wejścia wybiera pomiędzy komunikatami typu 0 lub typu 1, a i-ty bit (dla $1 \leq i \leq n-1$) dokonuje wyboru pomiędzy opcjami i-tego słowa wymienionego w komunikatach odpowiedniego typu. Rozważmy na przykład zdania:

- 0: Bob jest dobrym/pracowitym i uczciwym/godnym zaufania pracownikiem.
- 1: Bob jest trudnym/problematycznym i obciążającym/irytującym pracownikiem.

Zdefiniuj funkcję g , która przyjmuje 4-bitowe dane wejściowe, gdzie ostatni bit określa typ wyniku zdania, a pierwsze trzy bity określają wybór słów w tym zdaniu. Na przykład:

- $g(0000) = \text{Bob jest dobrym i uczciwym pracownikiem.}$
- $g(0001) = \text{Bob jest trudnym i wymagającym pracownikiem.}$
- $g(1010) = \text{Bob jest pracowitym i uczciwym pracownikiem. } g(1011) = \text{Bob jest problematycznym i wymagającym pracownikiem.}$

Teraz zdefiniuj $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ przez $f(x) = \overset{\text{def}}{=} H(g(x))$. Alicja może znaleźć kolizję w f , korzystając z pokazanego wcześniej ataku urodzinowego na małą przestrzeń. Chodzi o to, że każda kolizja $x, x \in f$ daje dwie wiadomości $g(x), g(x)$, które kolidują pod H . Jeśli x, x jest przypadkową kolizją, to spodziewamy się, że z prawdopodobieństwem $1/2$ kolidujące wiadomości $g(x), g(x)$ będą różnych typów (ponieważ x i x różnią się w swoim ostatnim bicie z takim prawdopodobieństwem). Jeżeli kolidujące komunikaty nie są różnego typu, proces można powtórzyć od nowa.

5.4.3 *Kompromisy czasu i przestrzeni w przypadku funkcji odwracających

W tej części rozważamy kwestię oporu obrazu wstępnego, czyli interesującą nas algorytmy dla problemu inwersji funkcji. Tutaj podany jest algorytm $y = H(x)$ dla jednolitego x , a celem jest znalezienie dowolnego takiego x

że $H(x) = y$. Zaczynamy od założenia, że długości wejściowe i wyjściowe H są równe, a na końcu krótko rozważymy bardziej ogólny przypadek.

Niech $H : \{0, 1\}^n \rightarrow \{0, 1\}^n$ beź dzie funkcją. Bez wykorzystywania jakichkolwiek słabości H , znalezienie obrazu wstę pnego punktu y można przeprowadzić w czasie $O(2)$ poprzez wyczerpujące przeszukiwanie dziedziny. Pokazujemy, że przy znacznym przetwarzaniu wstę pnym i stosunkowo dużej ilości pamięci można zrobić lepiej.

Dla jasności: traktujemy obróbkę wstę pną jako operację jednorazową i nie beź dziemy przesadnie przejmować się jej kosztami. Zamiast tego interesuje nas czas online wymagany do odwrócenia H w punkcie y , po zakończeniu przetwarzania wstę pnego. Jest to uzasadnione, jeśli koszt wstę pnego przetwarzania może zostać zamortyzowany przez inwersję wielu punktów lub jeśli jesteśmy skłonni zainwestować zasoby obliczeniowe w przetwarzanie wstę pnego, zanim znane będą dzie y , z korzyścią dla szyszej późniejszej.

W rzeczywistości użycie przetwarzania wstę pnego w celu umożliwienia inwersji funkcji w bardzo krótkim czasie jest trywialne. Wszystko, co musimy zrobić, to ocenić H w każdym punkcie fazy wstę pnego przetwarzania, a następnie zapisać pary $\{(x, H(x))\}$ w tabeli, posortowane według ich drugiego wpisu. Po otrzymaniu dowolnego punktu y , obraz wstę pnego można łatwo znaleźć, przeszukując tabelę w poszukiwaniu pary z drugim wpisem y . Wadą jest to, że musimy przydzielić miejsce do przechowywania par $O(2)$ w tabeli, co może być wygórowane, jeśli nie niemożliwe w przypadku dużych liczb (np. = 80).

Początkowy atak brute-force wykorzystuje stałą pamięć i czas $O(2)$, podczas gdy właśnie opisany atak przechowuje $O(2)$ punktów i umożliwia inwersję w zasadniczo stałym czasie. Przedstawiamy teraz podejście, które umożliwia atakującemu wymianę czasu i pamięci. W szczególności pokazujemy, jak przechowywać punkty $O(22/3)$ i znajdować obrazy wstę pnego w czasie $O(22/3)$; możliwe są inne kompromisy.

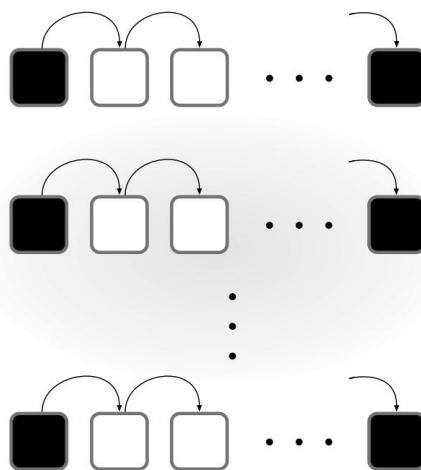
Rozgrzewka. Zaczynamy od rozważenia prostego przypadku, w którym funkcja H definiuje cykl, co oznacza, że $x, H(x), H(H(x)), \dots$ obejmuje wszystkie $\{0, 1\}^n$ dla dowolnego punktu początkowego x (zauważ, że większość funkcji nie definiuje cyklu, ale zakładamy to, aby zademonstrować tę ideę w bardzo prostym przypadku). Dla przejrzystości niech $N = 2$ oznacza rozmiar domeny.

W fazie wstę pnego przetwarzania atakujący po prostu wyczerpuje cały cykl, zaczynając od dowolnego punktu początkowego x_0 i obliczając $x_1 := H(x_0)$, $x_2 := H(H(x_0))$, aż do $x_N := H(N)$ (x_0), gdzie $H(i)$ odnosi się do i-krotnej oceny H .

$x_i \stackrel{\text{def}}{=} H(i)(x_0)$. Wyobraźmy sobie podział cyklu na N segmentów o długości \overline{N} . Niech każdy i posiadanie przez atakującego przechowywanie punktów na początku i na końcu każdego takiego segmentu. Oznacza to, że atakujący przechowuje w tabeli pary $(x_i \cdot \overline{N}, x_{(i+1) \cdot \overline{N}})$ dla $i = 0 \dots N - 1$, posortowane według drugiej postaci pary. Wynikowa tabela zawiera $O(\overline{N})$ punktów.

Kiedy atakujący otrzymuje punkt y do odwrócenia w fazie on-line, sprawdza, który z $y, H(y), H(2)y, \dots$ odpowiada punktowi końcowemu segmentu. (Każde sprawdzenie polega po prostu na sprawdzeniu w tabeli drugiego składnika przechowywanych par.) Ponieważ y leży w pewnym segmencie, gwarantowane jest osiągnięcie punktu końcowego w ciągu

N kroków. Po zidentyfikowaniu punktu końcowego $x = x_{(i+1) \cdot \overline{N}}$, atakujący przyjmuje punkt początkowy $x = x_i \cdot \overline{N}$ odpowiedniego segmentu



RYSUNEK 5.3: Generowanie tabeli. Przechowywane są tylko pary (SP_i, EP_i) .

i oblicza $H(x), H(2)(x), \dots$ aż do osiągnięcia y ; daje to natychmiast pożądany obraz wstępny. Zauważ, że wymaga to co najwyżej N ocen H .

Podsumowując, ten atak przechowuje $O(N)$ punktów i znajduje obrazy wstępne za pomocą prawdopodobieństwa 1 przy użyciu obliczeń skrótu $O(N)$.

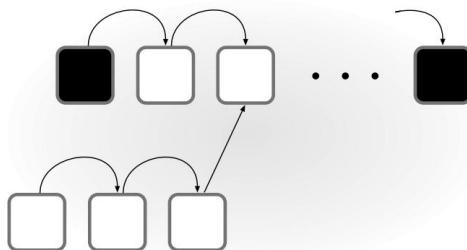
Kompromis czasu i przestrzeni Hellmana. Martin Hellman wprowadził bardziej ogólny kompromis czas/przestrzeń mający zastosowanie do dowolnej funkcji H (choć analiza traktuje H jako funkcję losową). Atak Hellmana nadal przechowuje punkt początkowy i końcowy kilku segmentów, ale w tym przypadku segmenty są raczej „niezależne” niż stanowią część jednego dużego cyklu. Mówiąc bardziej szczegółowo: niech s, t będą parametrami, które ustawiemy później. Atak najpierw wybiera jednolite punkty startowe SP_1, \dots, SP_s z $\{0, 1\}$. Dla każdego takiego punktu SP_i oblicza odpowiedni punkt końcowy $EP_i := H(t)(SP_i)$ przy użyciu t -krotnego zastosowania H .

(Patrz rysunek 5.3.) Następnie atakujący przechowuje tabelę wartości $\{(SP_i, EP_i)\}_{i=1}^s$ w posortowanej według drugiego wpisu z każdej pary.

Po otrzymaniu wartości y do odwrócenia atak przebiega jak w prostym przypadku omówionym wcześniej. W szczególności sprawdza, czy którykolwiek z $y, H(y), \dots, H(t-1)(y)$ jest równy punktowi końcowemu pewnego segmentu (zatrzymuje się, gdy tylko zostanie znalezione pierwsze takie dopasowanie). Możliwe jest, że żadna z tych wartości nie jest równa punktowi końcowemu (co omówimy poniżej). Jeżeli jednak $H(j)(y) = EP_i = H(t)(SP_i)$ dla jakiegoś i, j , to atakujący oblicza $H(t-j-1)(SP_i)$ i sprawdza, czy jest to obraz wstępny y . Cały proces wymaga co najwyżej t ocen H .

Wydaje się, że to działa, ale istnieje kilka subtelności, które zignorowaliśmy. Po pierwsze, może się zdarzyć, że żadne z $y, H(y), \dots, H(t-1)(y)$ jest punktem końcowym segmentu. Może się to zdarzyć, jeśli y nie znajduje się w zbiorze wartości $s \cdot t$ (nie licząc punktów początkowych) uzyskanych podczas początkowego procesu generowania tabeli. Możemy ustawić $s \cdot t = N$, próbując uwzględnić każdy ciąg -bitowy w

tabeli, ale to nie rozwiązuje problemu, ponieważ w samej tabeli mogą wystąpić kolizje — w rzeczywistości dla $s \cdot t = N1/2$ z naszej poprzedniej analizy problemu daty urodzenia wynika, że kolizje są prawdopodobne — co zmniejszy liczba odrębnych punktów w zbiorze wartości. Drugi problem, który pojawia się nawet jeśli y jest w tabeli, polega na tym, że nawet jeśli znajdziemy pasujący punkt końcowy, a zatem $H(j)(y) = EP_i = H(t)(SP_i)$ dla niektórych i, j , to nie gwarantuje, że $H(t-j-1)(SP_i)$ jest zapowiedzią y . Problem polega na tym, że segment $y, H(y), \dots, H(t-1)(y)$ może kolidować z i -tym segmentem, mimo że y samo w sobie nie znajduje się w tym segmencie; patrz rysunek 5.4. (Nawet jeśli y leży w jakimś segmencie, pierwszy pasujący punkt końcowy może nie znajdować się w tym segmencie). Nazywamy to wynikiem fałszywie dodatnim. Można by pomyśleć, że jest to mało prawdopodobne, jeśli H jest odporny na kolizje; znowu jednak mamy do czynienia z sytuacją, gdy w grę wchodzi wiele cej niż N punktów, a zatem kolizje stają się rzeczywiście prawdopodobne.



RYSUNEK 5.4: Zderzenie w fazie on-line.

Problem fałszywych alarmów można rozwiązać, modyfikując algorytm tak, aby zawsze obliczał całą sekwencję $y, H(y), \dots, H(t-1)(y)$ i sprawdza, czy $H(t-j-1)(SP_i)$ jest zapowiedzią y dla każdego i, j taką, że $H(j)(y) = EP_i$. Gwarantuje to znalezienie obrazu wstępnie pnego, o ile y znajduje się w zbiorze wartości (nie licząc punktów początkowych) wygenerowanych podczas przetwarzania wstępnie pnego. Obecnie problemem jest to, że czas działania algorytmu może się wydłużać, ponieważ każdy fałszywy alarm wiąże się z dodatkową oceną skrótu $O(t)$.

Mogą wykazać, że oczekiwana liczba wyników fałszywie dodatnich wynosi $O(st^2/N)$. (W ciągu $y, H(y), \dots, H(t-1)(y)$ znajduje się t wartości i co najwyżej s różnych punktów w tabeli). Traktując H jako funkcję losową, prawdopodobieństwo, że dowolny punkt w sekwencji równa się pewnemu punktowi w tabeli wynosi $1/N$. Oczekiwana liczba wyników fałszywie dodatnich wynosi zatem $t \cdot st \cdot 1/N = st^2/N$. Zatem dopóki $st \ll N$, co zapewnimy dla innych z poniższych powodów oczekiwana liczba fałszywych alarmów jest stała i oczekuje się, że radzenie sobie z fałszywymi alarmami będzie wymagało tylko $O(t)$ dodatkowych obliczeń skrótu.

Biorąc pod uwagę powyższą modyfikację, prawdopodobieństwo odwrócenia $y = H(x)$ jest co najmniej prawdopodobieństwem, że x znajduje się w zbiorze punktów (nie licząc punktów końcowych) wygenerowanych podczas przetwarzania wstępnie pnego. Teraz obniżamy granicę tego prawdopodobieństwa, biorąc pod uwagę losowość etapu przetwarzania wstępnie pnego

jednolity wybór x , traktując H jako funkcję losową w analizie. Najpierw obliczamy oczekiwanyą liczbę różnych punktów w tabeli. Zastanów się, co się stanie, gdy zostanie wygenerowany i -ty wiersz tabeli. Punkt początkowy SP_i jest jednolity i istnieje co najwyżej ($i - 1$) · t różnych punktów (nie licząc punktów końcowych) już w tabeli, więc prawdopodobieństwo, że SP_i jest „nowy” (tzn. nie jest równy żadnej poprzedniej wartości) wynosi co najmniej $1 - ((i - 1) \cdot t / N)$. Jakie jest prawdopodobieństwo, że $H(SP_i)$ jest nowy? Jeśli SP_i nie jest nowością, to prawie na pewno $H(SP_i)$ też nie jest. Z drugiej strony, jeśli SP_i jest nowy, to $H(SP_i)$ jest jednorodny (ponieważ H traktujemy jako funkcję losową), a więc c jest nowy z prawdopodobieństwem co najmniej $1 - ((i - 1) \cdot t + 1 / N)$. (Mamy teraz dodatkowy punkt SP_i .) Zatem prawdopodobieństwo, że $H(SP_i)$ jest nowy, wynosi co najmniej

$$\Pr[SP_i \text{ jest nowy}] \cdot \Pr[H(SP_i) \text{ jest nowy} \mid SP_i \text{ jest nowy}] = \\ \frac{(i-1) \cdot t \cdot (i-1) \cdot t + 1}{N} \cdot \frac{1}{N} \\ 1 - \frac{(i-1) \cdot t + 1}{N} > \frac{2}{N}.$$

Kontynuując w ten sposób, prawdopodobieństwo, że $H(t-1)(SP_i)$ jest nowe, wynosi co najmniej

$$t-1 \quad \frac{\text{ja} \cdot \frac{t}{N}}{N} \quad t=1 \quad \frac{\text{ja} \cdot \frac{N-t}{N}}{N} \quad \frac{\frac{2}{N}}{\text{mi it } 2/N}.$$

Należy tutaj zauważyć, że gdy $it \leq N/2$, prawdopodobieństwo to wynosi co najmniej $1/2$; z drugiej strony, gdy $it > N$, prawdopodobieństwo jest raczej małe. Biorąc pod uwagę ostatni wiersz, gdy $i = s$, oznacza to, że nie zyskamy zbyt wiele dodatkowego pokrycia, jeśli $st_2 > N$. Dobrym ustawieniem parametrów jest zatem $st_2 = N/2$.

Zakładając to, oczekiwana liczba różnych punktów w tabeli wynosi

$$\Pr_{\substack{i=1 \\ j=0}}^{s-t-1} H(j)(SP_i) \text{ jest nowy} = \frac{1}{2} = \frac{ul}{2}.$$

Prawdopodobieństwo, że x zostanie „pokryte” wynosi wtedy co najmniej $\frac{ul}{2N} = \frac{1}{T^4t}$.

Daje to słaby kompromis czas/przestrzeń, w którym możemy wykorzystać więcej przestrzeni (a w konsekwencji mniej czasu) kosztem zmniejszenia prawdopodobieństwa odwrócenia y . Ale możemy zrobić lepiej, generując „niezależne” tabelę $T = 4t$. (To zwiększa zarówno przestrzeń, jak i czas o współczynnik T .) Tak długo jak prawdopodobieństwa wystąpienia x w każdej z powiązanych tabel możemy traktować jako niezależne, prawdopodobieństwo, że co najmniej jedna z tych tabel zawiera x wynosi

$$1 - \Pr[\text{żadna tabela nie zawiera } x] = 1 - \left(1 - \frac{1}{4t}\right)^{4t} = 1 - e^{-1} \approx 0,63.$$

Pozostaje tylko pytanie, jak wygenerować niezależną tabelę. (Pamiętaj, że wygenerowanie tabeli dokładnie tak samo jak poprzednio jest równoznaczne z dodaniem dodatkowych

rows do naszej oryginalnej tabeli, co już widzieliśmy, nie pomaga.) Możemy to zrobić dla i-tej takiej tabeli, stosując jakąś funkcję F_i po każdej ocenie H , gdzie F_1, \dots, F_T są różne. (Dobrym wyborem może być ustalenie $F_i(x) = x$ — ci dla pewnej stałej stałej c_i , która jest różna w każdym

tabela.) Niech $H_i = F_i^{\text{def}} H$, tj. $H_i(x) = F_i(H(x))$. Następnie dla i-tej tabeli ponownie wybieramy losowe punkty początkowe, ale dla każdego takiego punktu obliczamy teraz $H_i(SP), H_i(2)$

\vdots (SP) i tak dalej. Po otrzymaniu wartości $y = H(x)$ do odwrócenia, atakujący najpierw oblicza $y = F_i(y)$, a następnie sprawdza, czy którakolwiek z $H(t-1)(y)$, $y, Cześć(y), \dots, \vdots$ odpowiada punktowi końcowemu w i-tej tabeli; to jest t . (Pomijamy dalsze powtarzane dla $i = 1, \dots, \vdots$ szczegóły.) Choć jest to trudne formalnie argumentując niezależność, takie podejście prowadzi do dobrych wyników w praktyce.

Wybór parametrów. Podsumowując powyższą dyskusję, widzimy, że dopóki $st_2 = N/2$ mamy algorytm, który przechowuje $O(s \cdot T) = O(s \cdot t) = O(N/t)$ punktów podczas fazy przetwarzania wstępne i może następnie odwrócić y ze stałym prawdopodobieństwem w czasie $O(t \cdot T) = O(tt = N/3 = 2/3)^2$. Jedno z ustawię parametrów polega

\vdots , na tym, że mamy algorytm przechowujący $O(2/3)$ punktów, który znajduje obrazy wstępne za pomocą obliczeń skrótu $O(2/3)$. Jeśli używana jest funkcja skrótu z 80 bitami danych wyjściowych, jest to wykonalne w praktyce.

Obsługa różnych domen i zakresów. W praktyce często spotyka się sytuację, w której dziedzina i zakres H są różne. Jednym z przykładów jest kontekst łamania haseł (patrz sekcja 5.6.3), gdzie atakujący ma $H(pw)$, ale $|pw|$. W ogólnym przypadku powiedzmy, że x jest wybrane z pewnej domeny D , która może być wieksza lub mniejsza niż $\{0, 1\}$. Choć możliwe jest oczywiście sztuczne rozszerzenie domeny/zakresu w celu ich dopasowania, nie będzie to przydatne w przypadku ataku opisanego powyżej. Aby zobaczyć dlaczego, rozważ przykład hasła. Aby atak się powiodł chcemy, aby pw znalazło się w jakiejś tabeli wartości wygenerowanych podczas przetwarzania wstępne. Jeśli wygenerujemy każdy wiersz tabeli, po prostu obliczając $H(SP), H(2)(SP), \dots$, dla $SP \in D$, to żadna z tych wartości (z wyjątkiem ewentualnie samego SP) nie będzie równa pw .

Możemy rozwiązać ten problem, stosując funkcję F_i jak poprzednio, poniżej dla każdej oceny H , chociaż teraz wybieramy mapowanie $F_i : \{0, 1\} \rightarrow D$. To rozwiązuje powyższy problem, ponieważ $F_i(H(SP)), F_i(H(2)(SP)), \dots$ teraz wszystko leży w D .

Aplikacje do ataków polegających na odzyskiwaniu klucza. Kompromisy czasu i przestrzeni umożliwiają ataki na prymitywy kryptograficzne inne niż funkcje skrótu. Jednym z kanonicznych zastosowań – w rzeczywistości zastosowaniem pierwotnie rozważanym przez Hellmana – jest atak na dowolny szyfr blokowy F , który prowadzi do odzyskania klucza. Zdefiniuj dokładnie $H(k) \stackrel{\text{def}}{=} F_k(x)$ gdzie x jest dowolnym, ale stałym wejściem, które zostanie użyte do zbudowania tabeli. Jeśli atakujący może uzyskać $F_k(x)$ dla nieznanego klucza k — albo poprzez atak z wybranym tekstem jawnym, albo wybierając x w taki sposób, że prawdopodobne jest uzyskanie $F_k(x)$ w ataku ze znanym tekstem jawnym — wówczas poprzez odwrócenie H atakującego uczy się (wartości kandydującej dla k). Należy zauważać, że długość klucza F może różnić się od długości jego bloku, ale w tym przypadku możemy zastosować technikę opisaną właśnie do obsługi H z inną dziedziną i zakresem.

5.5 Model losowej wyroczni

Istnieje kilka przykładów konstrukcji opartych na kryptograficznych funkcjach skrótu, których bezpieczeństwa nie można udowodnić w oparciu jedynie o założenie, że funkcja skrótu jest odporna na kolizje lub obraz wstę pny. (Zobaczmy niektóre z nich w następnej sekcji). W wielu przypadkach wydaje się, że nie ma prostego i rozsądnego założenia dotyczącego funkcji skrótu, które byłoby wystarczające do udowodnienia bezpieczeństwa konstrukcji.

W obliczu tej sytuacji istnieje kilka opcji. Jednym z nich jest poszukiwanie schematów, których bezpieczeństwo można udowodnić w oparciu o rozsądne założenia dotyczące podstawowej funkcji skrótu. Jest to dobre podejście, pozostawia jednak otwartą kwestię, co zrobić do czasu znalezienia takich schematów. Ponadto konstrukcje, które można udowodnić, że są bezpieczne, mogą być znacznie mniej wydajne niż inne podejścia, których bezpieczeństwa nie udowodniono. (Jest to istotny problem, który napotkamy w kontekście kryptografii klucza publicznego.)

Inną możliwością jest oczywiście wykorzystanie istniejącego kryptosystemu, nawet jeśli nie ma on innego uzasadnienia dla jego bezpieczeństwa niż być może fakt, że projektanci próbowali go zaatakować, ale bezskutecznie. Jest to sprzeczne ze wszystkim, co powiedzieliśmy na temat znaczenia rygorystycznego, nowoczesnego podejścia do kryptografii i powinno być jasne, że jest to niedopuszczalne.

Podejście, które okazało się niezwykle skuteczne w praktyce i które oferuje „środek” pomiędzy pełni rygorystycznym dowodem bezpieczeństwa z jednej strony a jego brakiem z drugiej strony, polega na wprowadzeniu wyidealizowanego modelu udowadniania bezpieczeństwa schematy kryptograficzne. Chociaż idealizacja może nie być dokładnym odzwierciedleniem rzeczywistości, możemy przynajmniej uzyskać pewną miarę pewności co do słuszności projektu schematu z dowodu zawartego w wyidealizowanym modelu. Dopóki model jest rozsądny, takie dowody są z pewnością lepsze niż brak dowodów.

Najpopularniejszym przykładem tego podejścia jest model random-oracle, który traktuje kryptograficzną funkcję skrótu H jako funkcję prawdziwie losową. (Widzieliśmy już tego przykład w naszej dyskusji na temat kompromisów czas/przestrzeń, chociaż tam analizowaliśmy atak, a nie konstrukcję.) Mówiąc dokładniej, model losowej wyroczni zakłada istnienie publicznej, losowej funkcji H, która można ocenić jedynie poprzez „zapytanie” wyroczni – którą można traktować jako „czarną skrzynkę” – która zwraca H(x) po podaniu danych wejściowych x. (Omówimy, jak należy to interpretować w następnej sekcji.) Dla rozróżnienia modelu, którego używaliśmy do tej pory (w którym nie ma przypadkowej wyroczni) jest czę sto nazywany „modelem standardowym”.

Nikt nie twierdzi, że istnieje przypadkowa wyrocznia, chociaż pojawiły się sugestie, że przypadkową wyrocznię można wdrożyć w praktyce przy użyciu zaufanej strony (tj. jakiegoś serwera w Internecie). Model losowej wyroczni zapewnia raczej formalną metodologię, która można zastosować do projektowania i sprawdzania poprawności schematów kryptograficznych przy użyciu następującego dwuetapowego podejścia:

1. Najpierw projektuje się schemat i sprawdza jego bezpieczeństwo w modelu losowej wyroczni. Oznacza to, że zakładamy, że świat zawiera przypadkową wyrocznię oraz konstruujemy i analizujemy schemat kryptograficzny w ramach tego modelu. Standardowe założenia kryptograficzne, jakie widzieliśmy do tej pory, można również wykorzystać w dowodzie bezpieczeństwa.
2. Gdy chcemy wdrożyć schemat w realnym świecie, losowa wyrocznia nie jest dostępna pna. Zamiast tego tworzona jest losowa wyrocznia z odpowiednio zaprojektowaną kryptograficzną funkcją skrótu H^* . (Wróćmy do tego punktu na końcu tej sekcji). Oznacza to, że w każdym punkcie, w którym schemat nakazuje, aby strona zapytała wyrocznię o wartość $H(x)$, strona zamiast tego samodzielnie oblicza $H^*(x)$.

Istnieje nadzieję, że kryptograficzna funkcja skrótu zastosowana w drugim etapie będzie „wystarczająco dobrą” w emulowaniu losowej wyroczni, tak że dowód bezpieczeństwa podany w pierwszym kroku zostanie przeniesiony do rzeczywistej instancji schematu. Trudność polega na tym, że nie ma teoretycznego uzasadnienia dla tej nadziei i w rzeczywistości istnieją (wymyślone) schematy, których bezpieczeństwo można wykazać w modelu losowej wyroczni, ale są one niepewne niezależnie od tego, w jaki sposób losowa wyrocznia zostanie utworzona w drugim kroku. Co więc cej, nie jest jasne (matematycznie lub heurystycznie), co oznacza, że funkcja skrótu jest „wystarczająco dobrą” w emulowaniu losowej wyroczni, ani nie jest jasne, czy jest to osiągalny cel. W szczególności brak konkretnej funkcji H^* , która może kiedykolwiek zachowywać się jak przypadkowy, ponieważ H^* jest deterministyczne i stałe. Z tych powodów dowód bezpieczeństwa w modelu random-oracle należy postrzegać jako dowód na to, że schemat nie ma „wrodzonych wad projektowych”, ale nie jest rygorystycznym dowodem na to, że jakakolwiek instancja schematu w świecie rzeczywistym jest bezpieczna. Dalszą dyskusję na temat interpretacji dowodów w modelu losowej wyroczni podano w rozdziale 5.5.2.

5.5.1 Szczegółowy model Random-Oracle

Zanim przejdziemy dalej, ustalmy dokładnie, na czym polega model losowej wyroczni. Dobry sposób myślenia o modelu losowej wyroczni jest następujący: „Wyrocznia” to po prostu pudełko, które przyjmuje ciąg binarny na wejściu i zwraca ciąg binarny na wyjściu. Wewnątrz trzne działanie pudełka jest nieznane i nieprzeniknione. Każdy – zarówno uczciwa strona, jak i przeciwnik – może wejść w interakcję ze skrzynką, gdzie taka interakcja polega na podaniu ciągu binarnego x na wejściu i otrzymaniu ciągu binarnego y na wyjściu; nazywamy to zapytaniem do wyroczni na temat x , a samo x nazywamy zapytaniem skierowanym do wyroczni. Zakłada się, że zapytania kierowane do wyroczni są prywatne, więc jeśli jakaś strona zadaje pytanie wyroczni po wprowadzeniu x , nikt inny nie dowiaduje się o x , a nawet nie dowiaduje się, że ta strona w ogóle zadawała pytania wyroczni. Ma to sens, ponieważ wywołania wyroczni odpowiadają (w instancji w świecie rzeczywistym) lokalnym ocenom kryptograficznej funkcji skrótu.

Wałą właściwością tego „pudełka” jest to, że jest spójne. Oznacza to, że jeśli kiedykolwiek pudełko wyświetli y dla określonego wejścia x , to zawsze wyświetli tę samą odpowiedź y , gdy ponownie otrzyma to samo wejście x . Oznacza to, że możemy przeglądać

box jako implementujący dobrze zdefiniowaną funkcję H ; tj. definiujemy funkcję H w oparciu o charakterystykę wejścia/wyjścia skrzynki. Dla wygody mówimy zatem o „zapytaniu H ”, a nie o pudle. Nikt nie „zna” całej funkcji H (z wyjątkiem samego pudełka); w najlepszym przypadku znane są tylko wartości H w ciągach znaków, które do tej pory były jawnie sprawdzane.

Omówiliśmy już w Rozdziale 3, co to znaczy wybrać funkcję losową H . Tutaj tylko powtórzymy, że istnieją dwa równoważne sposoby myślenia o jednolitym wyborze H : albo obraz H jest wybierany „za jednym razem” w sposób jednolity ze zbioru wszystkich funkcji w określonej dziedzinie i zakresie lub wybierz sobie generowanie wyników dla H , „w locie”, jeśli zajdzie taka potrzeba. W drugim przypadku możemy postrzegać funkcję jako zdefiniowaną przez tabelę, która początkowo jest pusta. Kiedy wyrocznia otrzymuje zapytanie x , najpierw sprawdza, czy $x = xi$ dla jakiejś pary (xi, yi) w tabeli; jeśli tak, zwracana jest odpowiednia wartość yi .

W przeciwnym razie wybierany jest jednolity ciąg $y \in \{0, 1\}$ (dla jakiegoś określonego), zwarcana jest odpowiedź y , a wyrocznia zapisuje (x, y) w swojej tablicy. Ten drugi punkt widzenia jest często łatwiejszy do uzasadnienia pod względem koncepcyjnym i jest również łatwiejszy do zrozumienia z technicznego punktu widzenia, jeśli H jest zdefiniowane w dziedzinie nieskończonej (np. $\{0, 1\}^\omega$).

Kiedy definiowaliśmy funkcje pseudolosowe w podrozdziale 3.5.1, rozważaliśmy także algorytmy mające dostęp do Oracle do funkcji losowej. Aby nie było żadnych nieporozumień, zauważamy, że użycie funkcji losowej w tym miejscu znacznie różni się od użycia funkcji losowej tutaj. Tam zastosowano funkcję losową jako sposób zdefiniowania, co to znaczy, że (konkretna) funkcja z kluczem jest pseudolosowa. Natomiast w modelu losowej wyroczni funkcja losowa jest używana jako część samej konstrukcji i musi w jakiś sposób zostać utworzona w świecie rzeczywistym, jeśli chcemy konkretnej realizacji konstrukcji. Funkcja pseudolosowa nie jest losową wyrocznią, ponieważ jest pseudolosowa tylko wtedy, gdy klucz jest tajny. Jednak w modelu losowej wyroczni wszystkie strony muszą być w stanie obliczyć funkcję; dlatego nie może być żadnego tajnego klucza.

Definicje i dowody w modelu Random-Oracle

Definicje w modelu losowej wyroczni różnią się nieco od swoich odpowiedników w modelu standardowym, ponieważ przestrzenie prawdopodobieństwa rozpatrywane w każdym przypadku nie są takie same. W modelu standardowym schemat Π jest bezpieczny, jeśli dla wszystkich ppt adwersarzy A prawdopodobieństwo jakiegoś zdarzenia jest poniżej pewnego progu, gdzie prawdopodobieństwo to uwzględnia się losowe wybory stron startujących Π i przeciwnika A . Zakładając, że uczciwe strony, które używają Π w świecie rzeczywistym, dokonują losowych wyborów zgodnie ze schematem, spełnienie tego rodzaju definicji gwarantuje bezpieczeństwo użycia Π w świecie rzeczywistym.

Natomiast w modelu losowej wyroczni schemat Π może opierać się na wyroczni H . Tak jak poprzednio, Π jest bezpieczny, jeśli dla wszystkich ppt przeciwników A prawdopodobieństwo jakiegoś zdarzenia jest poniżej pewnego progu, ale teraz to prawdopodobieństwo jest brane pod uwagę nad losowym wyborem H , a także przypadkowymi wyborami stron rządzących Π i przeciwnika A . Kiedy używasz Π w świecie rzeczywistym, niektóre (instancje) H muszą zostać ustalone. Niestety bezpieczeństwo Π nie jest gwarantowane

dla dowolnego konkretnego wyboru H. Wskazuje to na jeden powód, dla którego trudno jest argumentować, że jakakolwiek konkretna instancja wyroczni H za pomocą funkcji deterministycznej daje bezpieczny schemat. (Dodatkowa trudność techniczna polega na tym, że po ustaleniu konkretnej funkcji H przeciwnik A nie jest już ograniczony do zadawania pytań H jak wyrocznia, ale może zamiast tego spojrzeć na kod H i użyć go w trakcie ataku.)

Dowody w modelu losowej wyroczni mogą wykorzystywać fakt, że H jest wybierany losowo i że jedynym sposobem oceny $H(x)$ jest jawne zapytanie x do H. Szczególnie przydatne są szczególnie trzy właściwości; szkicujemy je tutaj nieformalnie i pokazujemy kilka prostych zastosowań poniżej i w następnej sekcji, ale ostrzegamy, że pełne zrozumienie bęzie prawdopodobnie musiało poczekać, aż w późniejszych rozdziałach przedstawimy formalne dowody w modelu losowej wyroczni.

Pierwszą użyteczną właściwością modelu losowej wyroczni jest:

Jeśli x nie zostało zapytane do H, wówczas wartość $H(x)$ jest jednolita.

Może się to wydawać powierzchownie podobne do gwarancji zapewnianej przez generator pseudolosowy, ale w rzeczywistości jest znacznie silniejsze. Jeśli G jest generatorem pseudolosowym, to $G(x)$ jest pseudolosowe dla obserwatora, zakładając, że x jest wybrane jednolicie losowo i jest całkowicie nieznane obserwatorowi. Jeśli jednak H jest przypadkową wyrocznią, wówczas $H(x)$ jest rzeczywiście jednolite dla obserwatora, o ile obserwator nie zadał pytania o x. Jest to prawda, nawet jeśli x jest znane lub jeśli x nie jest jednolite, ale trudno je odgadnąć. (Na przykład, jeśli x jest n-bitowym ciągiem znaków, którego pierwsza połowa x jest znana, a ostatnia połowa jest losowa, wówczas $G(x)$ może być łatwe do odróżnienia od losowego, ale $H(x)$ już nie.)

Pozostałe dwie właściwości odnoszą się wyraźnie do dowodów poprzez redukcję w modelu losowej wyroczni. (Pomocne może być tutaj przejrzenie Sekcji 3.3.2.) W ramach redukcji należy symulować losową wyrocznię, z którą przeciwnik A wchodzi w interakcję. To znaczy: A bęzie zadawał pytania i otrzymywał odpowiedzi od tego, co uważa za wyrocznię, ale sama redukcja musi teraz odpowiedzieć na te pytania. Okazuje się, że daje to dużą moc. Dla początkujących:

Jeśli A zapyta x do H, redukcja może zobaczyć to zapytanie i dowiedzieć się x.

Nazywa się to czasem „ekstrakwalnością”. (Nie przeczy to wspomnianemu wcześniej faktowi, że zapytania kierowane do losowej wyroczni są „prywatne”. Chociaż jest to prawda w samym modelu losowej wyroczni, tutaj używamy A jako podprogramu w ramach redukcji symulującej losową wyrocznię wyrocznią dla A.) Wreszcie:

Redukcja może ustawić wartość $H(x)$ (tj. odpowiedź na zapytanie x) na wybraną przez nią wartość, o ile wartość ta ma prawidłowy rozkład, tj. jest jednorodna.

Nazywa się to „programowalnością”. Nie ma odpowiednika ekstrakcji lub programowalności po utworzeniu instancji H z jakąkolwiek konkretną funkcją.

Proste ilustracje modelu Random-Oracle

W tym momencie pomocne mogą okazać się przykłady. Podane tutaj przykłady są stosunkowo proste i nie wykorzystują pełnej mocy, jaką zapewnia model losowej wyczerpania. Przykłady te przedstawiono raczej w celu delikatnego wprowadzenia do modelu. Poniżej zakładamy, że losowa wyczerpania mapuje wejścia bitowe na bitowe wyjścia, gdzie $\text{in} > \text{out} > n$, parametr bezpieczeństwa (więc $c_{\text{in}}, c_{\text{out}}$ są funkcjami n).

Losowa wyczerpania jako generator pseudolosowy. Najpierw pokazujemy, że dla $\text{out} > \text{in}$ losowa wyczerpania może zostać użyta jako generator pseudolosowy.

(Nie mówimy, że losowa wyczerpania jest generatorem pseudolosowym, ponieważ losowa wyczerpania nie jest funkcją stałą.) Formalnie twierdzimy, że dla dowolnego przeciwnika ppt A istnieje pomijalna funkcja negl taka, że

$$\Pr[\text{A } H(\cdot)(y) = 1] = \Pr[\text{A } H(\cdot)(H(x)) = 1] = \text{negl}(n),$$

gdzie w pierwszym przypadku przyjmuje się prawdopodobieństwo jednolitego wyboru H , jednolitego wyboru $y \in \{0, 1\}^{\text{out}(n)}$ i losowość A , a w drugim przypadku przyjmuje się prawdopodobieństwo jednolitego wyboru H , jednolity wybór $x \in \{0, 1\}^n$ i losowość A . Wyraźnie wskazaliśmy, że A ma w każdym przypadku dostęp do H ; po wybraniu H , A może swobodnie zadawać do niego zapytania.

Niech S oznacza zbiór punktów, o które A zapytał H ; oczywiście, $|S|$ jest wielomianem w n . Zauważmy, że w drugim przypadku prawdopodobieństwo, że $x \in S$ jest znikome. Dzieje się tak, ponieważ A zaczyna się bez informacji o x (zauważ, że samo $H(x)$ nie ujawnia niczego na temat x , ponieważ H jest funkcją losową) i ponieważ S jest wykładniczo mniejsze niż $\{0, 1\}^n$. Co więcej, w drugim przypadku uwarunkowane $x \in S$, danymi A są w każdym przypadku jednolitym ciągiem znaków, niezależnym od odpowiedzi na zapytania A .

Losowa wyczerpania jako funkcja skrótu odporna na kolizje. Jeśli $\text{out} < \text{in}$, losowa wyczerpania jest odporna na kolizje. Oznacza to, że prawdopodobieństwo sukcesu dowolnego przeciwnika ppt A w następstwie eksperymentu jest znikome:

1. Wybrano funkcję losową H .
2. A powiedzie się, jeśli wygeneruje różne x , $x \in H(x) = H(x)$.

Aby to zobaczyć, założmy bez utraty ogólności, że A wyprowadza tylko wartości x, x , o które wcześniej pytał wyczerpanie, i że A nigdy nie wysyła do wyczerpania tego samego zapytania dwa razy. Niech zapytania Oracle A będą x_1, \dots, x_q , gdy $q = \text{poli}(n)$, jasne jest, że prawdopodobieństwo, że A się powiedzie, jest ograniczone od góry prawdopodobieństwem, że $H(x_i) = H(x_j)$ dla pewnego $i = j$. Ale to jest dokładnie równe prawdopodobieństwu, że jeśli wybierzemy q ciągów $y_1, \dots, y_q \in \{0, 1\}^{\text{out}}$ niezależnie i równomiernie losowo, mamy $y_i = y_j$ dla pewnego $i = j$. To jest dokładnie problem urodzin, więc z wykorzystując wyników Załącznika A.4 mamy, że A powiedzie się z znikomym prawdopodobieństwem $O(q^{2/2} / \text{out})$.

Konstruowanie funkcji pseudolosowej na podstawie losowej wyroczni. W modelu losowej wyroczni dość łatwo jest również skonstruować funkcję pseudolosową. Założymy, że $\text{in}(n) = 2^n$ i $\text{out}(n) = n$ i zdefiniuj

$$F_k(x) \stackrel{\text{def}}{=} H(kx),$$

gdzie $|k| = |x| = rz$. W ćwiczeniu 5.11 musisz wykazać, że jest to funkcja pseudolosowa, a mianowicie, że dla dowolnego czasu wielomianowego A prawdopodobieństwo powodzenia A w następującym eksperymentem wynosi nie więcej niż $1/2$ plus funkcja nieistotna:

1. Funkcja H i wartości $k \in \{0, 1\}^n$ i $b \in \{0, 1\}$ są wybierane jednostajnie.
2. Jeżeli $b = 0$, przeciwnik A otrzymuje dostęp p do wyroczni dla $F_k(\cdot) = H(k \cdot)$.
Jeśli $b = 1$, wówczas A ma dostęp p do funkcji losowej odwzorowującej n-bitowe wejścia na n-bitowe wyjścia. (Ta funkcja losowa jest niezależna od H .)
3. A wyprowadza trochę b, i powiedzie się , jeśli $b = b$.

W kroku 2 A może uzyskać dostęp p do H oprócz funkcji Oracle zapewnionej mu w eksperymencie. (Funkcja pseudolosowa w modelu losowej wyroczni musi być nie do odróżnienia od funkcji losowej niezależnej od H .)

Interesującym aspektem wszystkich powyższych twierdzeń jest to, że nie przyjmują one żadnych założeń obliczeniowych; obowiązują one nawet w przypadku przeciwników nieograniczonych obliczeniowo, o ile przeciwnicy ci ograniczają się do wykonywania wielomianowych wielu zapytań do wyroczni. Nie ma to odpowiednika w świecie rzeczywistym, gdzie, jak widzieliśmy, konieczne są założenia obliczeniowe.

5.5.2 Czy metodologia Random-Oracle jest rozsądna?

Schematy zaprojektowane w modelu losowej wyroczni są implementowane w świecie rzeczywistym poprzez utworzenie instancji H z jakąś konkretną funkcją. Mając już za sobą mechanikę modelu losowej wyroczni, przechodzimy do bardziej podstawowego pytania:

Co gwarantują dowody bezpieczeństwa w modelu losowej wyroczni, jeśli chodzi o bezpieczeństwo dowolnej instancji w świecie rzeczywistym?

Na to pytanie nie ma jednoznacznej odpowiedzi: w społeczności kryptograficznej toczy się obecnie debata na temat interpretacji dowodów w modelu losowej wyroczni, a aktywnym obszarem badań jest określenie, co dokładnie jest dowodem bezpieczeństwa w modelu losowo-wyroczniowym. model wyroczni implikuje porównanie ze światem rzeczywistym. Możemy mieć tylko nadzieję , że oddamy przedsmak obu stron debaty.

Zastrzeżenia do modelu losowej wyroczni. Punkt wyjścia argumentów przeciwko stosowaniu losowych wyroczni jest prosty: jak już zauważliśmy, nie ma formalnego ani rygorystycznego uzasadnienia, aby sądzić, że dowód bezpieczeństwa dla jakiegoś schematu Π w modelu losowej wyroczni mówi cokolwiek o bezpieczeństwie Π w prawdziwym świecie, po utworzeniu losowej instancji wyroczni H

dowolna konkretna funkcja skrótu H^* . To coś wiej cej niż tylko teoretyczny niepokój. Krótki namysł pokazuje, że żadna konkretna funkcja skrótu nie może nigdy działać jako „prawdziwa” losowa wyrocznia. Na przykład w modelu losowej wyroczni wartość $H(x)$ jest „całkowicie losowa”, jeśli x nie zostało jawnie zapytane. Odpowiednikiem byłoby wymaganie, aby $H^*(x)$ było losowe (lub pseudolosowe), jeśli H^* nie zostało jawnie oszacowane na x . Jak mamy to interpretować w realnym świecie? Nie jest nawet jasne, co to znaczy „jawnie ocenić” H^* : co się stanie, jeśli przeciwnik zna jakiś skrót do obliczenia H^* , który nie wymaga uruchomienia faktycznego kodu dla H^* ? Co wiej cej, $H^*(x)$ nie może być losowe (ani nawet pseudolosowe), ponieważ gdy przeciwnik dowie się, opis H^* na wszystkich danych wejściowych jest natychmiast określany, wartość tej funkcji

Ograniczenia modelu losowej wyroczni staną się wyraźniejsze, gdy przeanalizujemy wprowadzone wcześniej techniki dowodowe. Przypomnijmy, że jedną z technik dowodowych jest wykorzystanie faktu, że redukcja może „zobaczyć” zapytania, które przeciwnik A kieruje do losowej wyroczni. Jeśli zastąpimy losową wyrocznię konkretną funkcją skrótu H^* , oznacza to, że na początku eksperymentu musimy przedstawić przeciwnikowi opis H^* . Ale wtedy A może samodzielnie oszacować H^* , bez robienia żadnych jawnego zapytań, a zatem redukcja nie będzie już w stanie „zobaczyć” żadnych zapytań zadanych przez A. (W rzeczywistości, jak zauważono wcześniej, koncepcja A wykonującego jawnie oceny H^* mogą nie być prawdziwe i na pewno nie można ich formalnie zdefiniować.) Podobni dowody bezpieczeństwa w modelu losowej wyroczni pozwalają na redukcję w wyborze wyników H według własnego uznania, co jest wyraźnie niemożliwe, gdy określona jest konkretna funkcja używana.

Nawet jeśli jesteśmy skłonni przeoczyć powyższe obawy teoretyczne, praktycznym problemem jest to, że obecnie nie bardzo dobrze rozumiemy, co to znaczy, że konkretna funkcja skrótu jest „wystarczająco dobra” w tworzeniu instancji losowej wyroczni. Dla konkretnego założymy, że chcemy utworzyć instancję losowej wyroczni przy użyciu odpowiedniej modyfikacji SHA-1 (SHA-1 to kryptograficzna funkcja skrótu omówiona w sekcji 6.3.3). Podczas gdy w przypadku jakiegoś konkretnego schematu Π rozsądne może być założenie, że Π jest bezpieczne, gdy zostanie utworzone przy użyciu SHA-1, znacznie mniej rozsądne jest założenie, że SHA-1 może zastąpić losową wyrocznię w każdym schemacie zaprojektowanym w trybie random-model wyroczni.

Rzeczywiście, jak powiedzieliśmy wcześniej, wiemy, że SHA-1 nie jest przypadkową wyrocznią. Nie jest trudno zaprojektować schemat, który będzie bezpieczny w modelu losowej wyroczni, ale będzie niepewny, gdy losowa wyrocznia zostanie zastąpiona przez SHA-1.

Podkreślamy, że założenie w postaci „SHA-1 działa jak losowa wyrocznia” jakościowo różni się od założeń takich jak „SHA-1 jest odporny na kolizje” czy „AES jest funkcją pseudolosową”. Problem częściowo polega na tym, że nie ma zadowalającej definicji tego, co oznacza pierwsze stwierdzenie, podczas gdy mamy takie definicje dla dwóch ostatnich.

Z tego powodu wykorzystanie modelu random-oracle do udowodnienia bezpieczeństwa schematu różni się jakościowo od np. wprowadzenia nowego założenia kryptograficznego w celu udowodnienia bezpieczeństwa schematu w modelu standardowym. Dlatego dowody bezpieczeństwa w modelu losowej wyroczni są mniej satysfakcjonujące niż dowody bezpieczeństwa w modelu standardowym.

Obsługa modelu losowej wyroczni. Biorąc pod uwagę wszystkie problemy związane z modelem losowej wyroczni, po co w ogóle go używać? Mówiąc bardziej sedno: dlaczego model losowej wyroczni wywarł tak duży wpływ na rozwój współczesnej kryptografii (zwłaszcza na obecne praktyczne zastosowanie kryptografii) i dlaczego nadal jest tak szeroko stosowany? Jak zobaczymy, model losowej wyroczni umożliwia projektowanie znacznie wydajniejszych schematów niż te, które umiemy konstruować w modelu standardowym. W związku z tym obecnie używanych jest niewiele (jeśli w ogóle) kryptosystemów z kluczem publicznym posiadających dowody bezpieczeństwa w modelu standardowym, podczas gdy istnieje wiele wdrożonych schematów posiadających dowody bezpieczeństwa w modelu random-oracle. Ponadto dowody w modelu losowej wyroczni są niemal powszechnie uznawane za potwierdzające bezpieczeństwo systemów rozważanych do standaryzacji.

Podstawową przyczyną tego jest przekonanie, że:

Dowód bezpieczeństwa w modelu random-oracle jest znacznie lepszy niż żaden dowód.

Chociaż niektórzy się z tym nie zgadzają, na poparcie tego twierdzenia oferujemy następujące argumenty:

- Dowód bezpieczeństwa schematu w modelu losowej wyroczni wskazuje, że projekt schematu jest „solidny” w tym sensie, że jedynymi możliwymi atakami na instancję schematu w świecie rzeczywistym są te, które powstają z powodu słabości funkcja skrótu używana do tworzenia losowej wyroczni. Zatem, jeśli do utworzenia losowej wyroczni zostanie użyta „wystarczająco dobra” funkcja skrótu, powinniśmy mieć pewność co do bezpieczeństwa schematu.
Co więcej, jeśli dana instancja schematu zostanie pomyślnie zaatakowana, możemy po prostu zastąpić używaną funkcję skrótu „lepszą”.
- Co ważne, nie doszło do udanych ataków w świecie rzeczywistym na schematy sprawdzone w modelu losowej wyroczni, gdy instancja losowej wyroczni została prawidłowo utworzona. (Nie uwzględniamy tutaj ataków na „wymyślone” schematy, ale zauważamy, że należy zachować szczególną ostrożność przy tworzeniu losowej wyroczni, jak wskazano w ćwiczeniu 5.10.) Daje to dowód na użyteczność modelu losowej wyroczni w projektowaniu praktycznych schematów.

Niemniej jednak powyższe ostatecznie reprezentuje jedynie intuicyjne spekulacje co do przydatności dowodów w modelu losowej wyroczni i – przy założeniu, że wszystkie inne czynniki są równe – preferowane są dowody bez przypadkowych wyroczni.

Tworzenie instancji losowej wyroczni

Prawidłowe utworzenie losowej wyroczni jest subtelną sprawą, a pełne omówienie wykracza poza zakres tej książki. W tym miejscu jedynie ostrzegamy czytelnika, że używanie „gotowej” kryptograficznej funkcji skrótu bez modyfikacji nie jest, ogólnie rzecz biorąc, rozsądnym podejściem. Po pierwsze, wiele kryptograficznych funkcji skrótu jest zbudowana przy użyciu paradygmatu Merkle'a – Damgård (por. sekcja 5.2), który można łatwo odróżnić od losowej wyroczni, gdy dane wejściowe mają zmienną długość

są dozwolone. (Patrz ćwiczenie 5.10.) Ponadto w niektórych konstrukcjach konieczne jest, aby wynik losowej wyroczni mieścić się w pewnym przedziale (np. wyrocznia powinna wyprowadzać elementy jakiejś grupy), co powoduje dodatkowe komplikacje.

5.6 Dodatkowe zastosowania funkcji skrótu

Zakończymy ten rozdział krótkim omówieniem niektórych dodatkowych zastosowań kryptograficznych funkcji skrótu w kryptografii i bezpieczeństwie komputerowym.

5.6.1 Odciski palców i deduplikacja

Podczas korzystania z odpornej na kolizje funkcji skrótu H, skrót (lub skrót) pliku służy jako unikalny identyfikator tego pliku. (Jeśli okaże się, że jakikolwiek inny plik ma ten sam identyfikator, oznacza to kolizję w H). Hash H(x) pliku x jest jak odcisk palca i można sprawdzić, czy dwa pliki są równe, porównując ich skróty. Ten prosty pomysł ma wiele zastosowań.

- **Odciski palców wirusów:** Skanery antywirusowe identyfikują wirusy i blokują je lub poddają kwarantannie. Jednym z najbardziej podstawowych kroków w tym kierunku jest przechowywanie bazy danych zawierającej skróty znanych wirusów, a następnie sprawdzenie w tej bazie danych skrótu pobranej aplikacji lub załącznika do wiadomości e-mail.
Ponieważ dla każdego wirusa wystarczy zarejestrować (i/lub rozprowadzić) tylko krótki ciąg znaków, możliwe jest poniesienie związanych z tym kosztów ogólnych.
- **Deduplikacja:** Deduplikacja danych służy do eliminacji duplikatów danych, szczególnie w kontekście przechowywania danych w chmurze, gdzie wielu użytkowników korzysta z jednej usługi w chmurze do przechowywania swoich danych. Należy zauważyć, że jeśli wielu użytkowników chce przechowywać ten sam plik (np. popularny film), wówczas plik wystarczy zapisać tylko raz i nie trzeba go przesyłać osobno przez każdego użytkownika. Deduplikację można osiągnąć, prosząc użytkownika o przesłanie skrótu nowego pliku, który chce przechowywać; jeśli plik z tym skrótem jest już przechowywany w chmurze, dostawca usługi przechowywania w chmurze może po prostu dodać wskaźnik do istniejącego pliku, aby wskazać, że ten konkretny użytkownik również zapisał ten plik. Oszczędza to zarówno komunikację, jak i pamięć, a solidność metodologii wynika z odporności funkcji skrótu na kolizje.
- Udoskonalenie plików w trybie peer-to-peer (P2P): w systemach udostępniania plików P2P serwery przechowują tabele w celu zapewnienia usługi wyszukiwania plików. Tabele te zawierają skróty dostępne plików, ponownie zapewniając unikalny identyfikator bez użycia dużej ilości pamięci.

Może być zaskakujące, że małe podsumowanie może jednoznacznie zidentyfikować każdy plik na świecie. Ale taką gwarancję zapewniają odporne na kolizje funkcje skrótu, co czyni je przydatnymi w powyższych ustawieniach.

5.6.2 Drzewa Merkle'a

Rozważmy klienta, który przesyła plik x na serwer. Gdy klient później pobierze x , chce mieć pewność, że serwer zwróci oryginalny, niezmodyfikowany plik. Klient mógłby po prostu zapisać x i sprawdzić, czy pobrany plik jest równy x , ale to w ogóle mija się z celem korzystania z serwera.

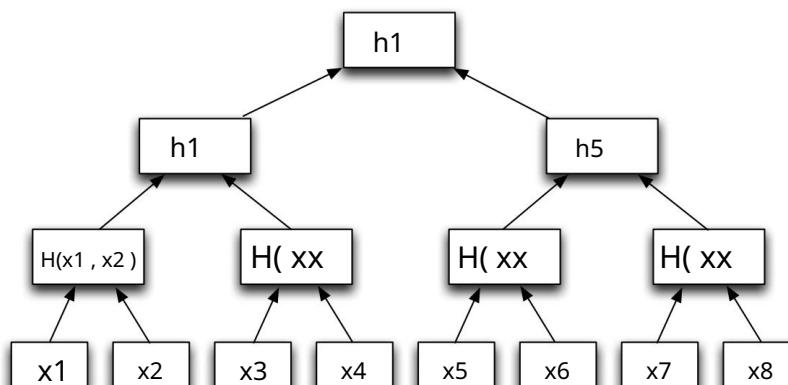
Poszukujemy rozwiązań, w którym magazyn klienta jest niewielki.

Naturalnym rozwiązaniem jest zastosowanie opisanej powyżej metody „odcisku palca”. Klient może lokalnie przechowywać krótki skrót $h := H(x)$; gdy serwer zwróci aktu kandydata x , klient musi jedynie sprawdzić, czy $H(x) \stackrel{?}{=} h$.

Co się stanie, jeśli będziemy chcieli rozszerzyć to rozwiązanie na wiele plików x_1, \dots, x_t ? Można to zrobić na dwa oczywiste sposoby. Jednym z nich jest po prostu niezależne hashowanie każdego pliku; klient będzie lokalnie przechowywać podsumowania h_1, \dots, h_t i zweryfikować pobrane pliki jak poprzednio. Ma to tę wadę, że pamięć klienta rośnie liniowo w t . Inną możliwością jest zmieszanie wszystkich plików razem. Oznacza to, że klient może obliczyć $h := H(x_1, \dots, x_t)$ i zapisać tylko h . Wadą jest to, że gdy klient chce pobrać i sprawdzić poprawność i -tego pliku x_i

, musi pobrać wszystkie pliki, aby ponownie obliczyć podsumowanie.

Drzewa Merkle, wprowadzone przez Ralphego Merkle'a, stanowią kompromis między tymi skrajnościami. Drzewo Merkle'a obliczone na podstawie wartości wejściowych x_1, \dots, x_t jest po prostu drzewem binarnym o głębokości $\log t$, w którym dane wejściowe są umieszczone na liściach, a wartość każdego węzła wewnętrzne trznego jest skrótem wartości jego dwóch dzieci; patrz rysunek 5.5. (Zakładamy, że t jest potęgą gęstości 2; jeśli nie, możemy ustawić niektóre wartości wejściowe na zero lub użyć niekompletnego drzewa binarnego, w zależności od aplikacji.)



RYSUNEK 5.5: Drzewo Merkle.

Naprawiając jakąś funkcję skrótu H , oznaczamy przez MT_t funkcja, która przyjmuje t wartości wejściowe x_1, \dots, x_t , oblicza wynikowe drzewo Merkle i wyprowadza wartość korzenia drzewa. (Kluczowana funkcja skrótu daje kluczowaną funkcję MT_t w oczywisty sposób.) Mamy:

TWIERDZENIE 5.11 Niech $(GenH, H)$ bę dzie odporne na kolizje. Wtedy $(GenH, MT_t)$ jest również odporne na kolizje dla dowolnego ustalonego t .

Drzewa Merkle'a stanowią zatem alternatywę dla transformaty Merkle'a-Damgård w celu uzyskania rozszerzenia domeny dla funkcji skrótu odpornych na kolizje. (Jednakże, jak opisano, drzewa Merkle'a nie są odporne na kolizje, jeśli liczba wartości wejściowych t może się zmieniać.)

Drzewa Merkle'a zapewniają skuteczne rozwiązywanie naszego pierwotnego problemu, ponieważ umożliwiają weryfikację dowolnego z pierwotnych danych wejściowych t przy użyciu komunikacji $O(\log t)$. Klient oblicza $h := MT_t(x_1, \dots, x_t)$, przesyła x_1, \dots, x_t na serwer i przechowuje h (wraz z liczbą plików t) lokalnie. Gdy klient pobierze i-ty plik, serwer wysyła x_i wraz z „dowodem” t_i , że jest to poprawna wartość. Dowód ten składa się z wartości węzłów drzewa Merkle'a sąsiadujących ze ścieżką od x_i do korzenia. Na podstawie tych wartości klient może ponownie obliczyć wartość pierwiastka i sprawdzić, czy jest ona równa zapisanej wartości h . Jako przykład rozważmy drzewo Merkle'a na rysunku 5.5. Klient oblicza $h_{1\dots 8} := MT_8(x_1, \dots, x_8)$, przesyła x_1, \dots, x_8 na serwer i przechowuje lokalnie $h_{1\dots 8}$. Gdy klient pobiera x_3 , serwer wysyła x_3 wraz z $x_4, h_{1\dots 2} = H(x_1, x_2)$ i $h_{5\dots 8} = H(H(x_5, x_6), H(x_7, x_8))$. (Jeśli pliki są duże, możemy chcieć uniknąć wysyłania innego pliku niż ten, którego zażądał klient. Można to łatwo zrobić, jeśli zdefiniujemy drzewo Merkle na podstawie skrótów plików, a nie samych plików. Pomijamy szczegóły.) Klient oblicza $h := H(h_{1\dots 2}, H(x_3, x_4))$ i $h := H(h_{1\dots 4}, h_{5\dots 8})$, a następnie sprawdza, czy $h_{1\dots 8} = h$.

1...4

1...8

1...8.

Jeśli H jest odporny na kolizje, serwer nie może wysłać niepoprawnego pliku (i żadnego dowodu), który spowodowałby pomyślną weryfikację. Dzięki temu podejściu lokalna pamięć klienta jest stała (niezależna od liczby plików t), a komunikacja między serwerem a klientem jest proporcjonalna do $\log t$.

5.6.3 Mieszanie hasła

Jednym z najczęstszych i najważniejszych zastosowań funkcji skrótu w bezpieczeństwie komputera jest ochrona hasłem. Rozważmy sytuację, w której użytkownik wpisuje hasło przed użyciem laptopa. Aby uwierzytelnić użytkownika, jaką formą hasła użytkownika musi być przechowywana gdzieś na jego laptopie. Jeśli hasło użytkownika jest przechowywane w postaci jawniej, przeciwnik kradnący laptopa może odczytać hasło użytkownika z dysku twardego, a następnie zalogować się jako ten użytkownik. (Ukrywanie hasła przed osobą atakującą, która może już odczytać zawartość dysku twardego, może wydawać się bezcelowe. Jednakże pliki na dysku twardym mogą zostać

zaszyfrowane kluczem pochodzącym z hasła użytkownika, a zatem byłby dostępu nie dopiero po wprowadzeniu hasła. Ponadto użytkownik prawdopodobnie użyje tego samego hasła w innych witrynach.)

Ryzyko to można ograniczyć, przechowując skrót hasła zamiast samego hasła. Oznacza to, że dysk twardy przechowuje wartość $h(pw) = H(pw)$ w pliku haseł; później, gdy użytkownik wprowadzi swoje hasło pw, system operacyjny przed udzieleniem dostępu sprawdzi, czy $H(pw) \neq h(pw)$. To samo podstawowe podejście jest również stosowane w przypadku uwierzytelniania opartego na hasłach w Internecie. Teraz, jeśli osoba atakująca ukradnie dysk twardy (lub włame się na serwer WWW), uzyska jedynie skrót hasła, a nie samo hasło.

Jeśli hasło zostanie wybrane ze stosunkowo małej przestrzeni D możliwości (np. D może być słownikiem angielskich słów, w tym przypadku $|D| = 80\,000$), atakujący może wyliczyć wszystkie możliwe hasła $pw_1, pw_2, \dots, pw_{|D|}$ i sprawdzić, czy $H(pw_i) = h(pw)$. Chcielibyśmy nic lepszego. Zapewnić każdego kandydata pw, że atakujący nie może zrobić (Zapewniłoby to również, że przeciwnik nie mógłby poznać hasła żadnego użytkownika, który wybrałby silne hasło z dużej przestrzeni.) Niestety, opór obrazu wstępu nie (tj. jednokierunkowość) H nie jest wystarczający, aby sugerować to, czego chcemy. Po pierwsze, odporność na obraz wstępu nie mówi tylko, że $H(x)$ jest trudne do odwrócenia, gdy x jest wybrane równomiernie z dużej dziedziny, takiej jak $\{0, 1\}^n$. Nie mówi nic o twardości odwracania H, gdy x jest wybrane z innej przestrzeni lub gdy x jest wybrane według innego rozkładu. Co więcej, opór przedobrazu nie mówi nic o konkretnej ilości czasu niezbędnego na znalezienie przedobrazu. Na przykład skrót o danym $H(x)$ wymaga czasu $2n/2$ funkcji H, dla której obliczenie $x \in \{0, 1\}^n$ może nadal kwalifikować się jako odporność na obraz bitowe hasło można odzyskać tylko w 215 raz. Wstępu nie, ale oznaczałoby to, że 30-

Jeśli modelujemy H jako losową wyrocznię, możemy formalnie udowodnić, że chcemy bezpieczeństwa, a mianowicie odzyskanie pw z h(pw) (zakładając, że pw jest wybrane jednolicie z D) wymaga średnio $|D|/2$ ocen H.

W powyższym omówieniu założono, że atakujący nie przeprowadza żadnego wstępnie przetwarzania. Jak jednak widzieliśmy w podrozdziale 5.4.3, przetwarzanie wstępu nie można wykorzystać do wygenerowania dużych tabel, które umożliwiają inwersję (nawet funkcji losowej!) szybciej niż wyszukiwanie wyczerpujące. Jest to poważny problem w praktyce: nawet jeśli użytkownik wybierze swoje hasło jako losową kombinację 8 znaków alfanumerycznych – co daje przestrzeń na hasło o rozmiarze $N = 628 = 2^{47,6}$ – następne przestrzeń $N/2/3 = 2^{32}$ atak wykorzystujący czas, który będzie bardzo skuteczny. Tabele wystarczy wygenerować tylko raz i można je wykorzystać do złamania setek tysięcy hasłów w przypadku naruszenia bezpieczeństwa serwera. Takie ataki są rutynowo przeprowadzane w praktyce.

Łagodzenie. Pokrótko opisujemy dwa mechanizmy stosowane w celu ograniczenia zagrożenia złamaniem hasła; dalsze dyskusje można znaleźć w tekstu na temat bezpieczeństwa komputerowego. Jedną z technik jest użycie „powolnych” funkcji skrótu lub spowolnienie istniejących funkcji skrótu poprzez zastosowanie wielokrotnych iteracji (tj. obliczenie $H(I)$ (pw, dla mnie) 1). Powoduje to spowolnienie legalnych użytkowników o współczynnik I, co nie stanowi problemu, jeśli parametr I jest ustawiony na jakąś „umiarkowaną” wartość (np. 1000).

Z drugiej strony ma to znaczący wpływ na przeciwnika próbującego złamać tysiące haseł na raz.

Drugi mechanizm polega na wprowadzeniu soli. Kiedy użytkownik zarejestruje swoje hasło, laptop/serwer wygeneruje długą losową wartość s („sól”), unikalną dla tego użytkownika i zapisze (s, hpw = H(s, pw)) zamiast po prostu przechowywać H (pw) jak poprzednio. Ponieważ atakujący nie zna z góry adresu s, przetwarzanie wstępne jest nieskuteczne i najlepsze, co może zrobić atakujący, to poczekać, aż uzyska plik z hasłami, a nastepnie przeprowadzić wyczerpujące przeszukiwanie domeny D w czasie liniowym, jak omówiono wcześniej. Należy również pamiętać, że ponieważ dla każdego przechowywanego hasła używana jest inna sól, do odzyskania każdego hasła potrzebne jest osobne wyszukiwanie metodą brut

5.6.4 Wyprowadzenie klucza

Wszystkie kryptosystemy z kluczem symetrycznym, które widzieliśmy, wymagają równomiernie rozłożonego ciągu bitów dla tajnego klucza. Często jednak dla obu stron wygodniej jest polegać na wspólnych informacjach, takich jak hasło lub dane biometryczne, które nie są równomiernie rozmieszczone. (Przechodząc dalej, w rozdziale 10 zobaczymy, jak strony mogą wchodzić w interakcję, aby wygenerować wspólny sekret o wysokiej entropii, który nie jest równomiernie rozłożony.) Strony mogą próbować wykorzystać swoje wspólne informacje bezpośrednio jako tajny klucz, ale ogólnie jest to nie bezpieczny (ponieważ np. wszystkie schematy klucza prywatnego zakładają klucz równomiernie rozproszony). Co więcej, udostępniane dane mogą nawet nie mieć odpowiedniego formatu, aby można było ich użyć jako tajnego klucza (mogą być na przykład za długie).

Obciążenie wspólnego sekretu lub mapowanie go w inny doraźny sposób na串 o prawidłowej długości może spowodować utratę znacznej ilości entropii. (Poniżej bardziej formalnie definiujemy jedno pojęcie entropii, ale na razie można myśleć o entropii jako logarytmie przestrzeni możliwych wspólnych tajemnic. Założymy na przykład, że dwie strony dzielą się hasłem złożonym z 28 losowych wielkich liter i chcesz używać kryptosystemu z kluczem 128-bitowym. Ponieważ dla każdego znaku istnieje 26 możliwości, istnieje $26^{28} > 2^{130}$ możliwości haseł. Jeśli hasło jest udostępniane w formacie ASCII, każdy znak jest przechowywany przy użyciu 8 bitów, dlatego całkowita długość hasła wynosi 224 bity. Jeśli strony skrócią swoje hasło do pierwszych 128 bitów, użyją tylko pierwszych 16 znaków swojego hasła. Nie bez powodu to jednak równomiernie rozłożony串 128-bitowy! W rzeczywistości reprezentacje ASCII liter A-Z mieszczą się w przedziale od 0x41 do 0x5A; w szczególności pierwsze 3 bity każdego bajtu to zawsze 010. Oznacza to, że 37,5% bitów wynikowego klucza zostanie ustalonych, a 128-bitowy klucz, który strony wybiorą, będzie miał tylko około 75 bitów entropii (tj. istnieje tylko około 275 możliwości dla klucza).

Potrzebujemy ogólnego rozwiązania do wyprowadzenia klucza ze wspólnego sekretu o wysokiej entropii (ale niekoniecznie jednolitego). Zanim przejdziemy dalej, zdefiniujemy pojęcie entropii, które tutaj rozważamy.

DEFINICJA 5.12 Rozkład prawdopodobieństwa X ma m bitów minimalnej entropii , jeśli dla każdej ustalonej wartości x przyjmuje się , że $\Pr[X = x] \geq 2^{-m}$. Oznacza to, że nawet najbardziej prawdopodobny wynik występuje z prawdopodobieństwem co najwyżej 2^{-m} .

Równomierny rozkład na zbiorze o rozmiarze S ma log minimalnej entropii S . Rozkład, w którym jeden element występuje z prawdopodobieństwem $1/10$, a 90 elementów występuje z prawdopodobieństwem $1/100$, ma log minimalnej entropii $10 \approx 3,3$. Minimalna entropia rozkładu mierzy prawdopodobieństwo, jakim atakujący może odgadnąć wartość pobraną z tego rozkładu; najlepszą strategią atakującego jest odgadnięcie najbardziej prawdopodobnej wartości, więc jeśli rozkład ma minimalną entropię m , atakujący odgadnie poprawnie z prawdopodobieństwem co najwyżej 2^{-m} . To wyjaśnia, dlaczego minimalna entropia (a nie inne pojęcia entropii) jest przydatna w naszym kontekście.

Rozszerzenie minimalnej entropii, zwane obliczeniową min-entropią, jest zdefiniowane jak powyżej, z tym wyjątkiem, że wymaga się jedynie, aby rozkład był nie do odróżnienia obliczeniowego od rozkładu o danej minimalnej entropii. (Pojęcie nieroróżnialności obliczeniowej zostało formalnie zdefiniowane w rozdziale 7.8.)

Funkcja wyprowadzania klucza umożliwia uzyskanie równomiernie rozłożonego łańcucha z dowolnej dystrybucji o wysokiej (obliczeniowej) minimalnej entropii. Nietrudno zauważyć, że jeśli modelujemy funkcję skrótu H jako losową wyrocznię , wówczas H służy jako dobra funkcja wyprowadzania klucza. Rozważmy niepewność atakującego co do $H(X)$, gdzie X jest próbkiem z rozkładu o minimalnej entropii m (ze względów technicznych wymagamy, aby rozkład był niezależny od H). Każde zapytanie atakującego skierowane do H można postrzegać jako „przypuszczenie” wartości X ; przy założeniu minimalnej entropii rozkładu, atakujący wykonujący q zapytań do H , zapyta $H(X)$ z prawdopodobieństwem co najwyżej $q \cdot 2^{-m}$. Jeśli atakujący nie zapyta X do H , wówczas $H(X)$ będzie ciągiem jednolitym.

Możliwe jest również zaprojektowanie funkcji wyprowadzania klucza bez polegania na modelu losowej Oracle, przy użyciu kluczowych funkcji skrótu zwanych (silnymi) ekstraktorami. Klucz do ekstraktora musi być jednolity, ale nie musi być utrzymywany w tajemnicy. Jeden ze standardów nazywa się HKDF; zobacz odniesienia na końcu rozdziału.

5.6.5 Schematy zobowiązania

Schemat zaangażowania pozwala jednej ze stron „zaangażować się ” w wiadomość m poprzez wysłanie wartości zobowiązania com , uzyskując jednocześnie następujące pozorne sprzeczne właściwości:

- **Ukrywanie:** zobowiązanie nie ujawnia niczego na temat m .
- **Wiązanie:** nie jest możliwe, aby osoba zatwierdzająca wygenerowała zobowiązanie com , które mogłyby później „otworzyć” jako dwie różne wiadomości m, m . (W tym sensie com naprawdę „zobowiązuje” osobę wykonującą jakąś dobrze określona wartość.)

Schemat zobowiązań można postrzegać jako kopertę cyfrową: zapieczętowanie wiadomości w kopercie i przekazanie jej innej osobie zapewnia prywatność (do czasu

koperta jest otwarta) i wiążące (ponieważ koperta jest zaklejona).

Formalnie (nieinteraktywny) schemat zobowiązania jest definiowany przez losowy algorytm Gen , który generuje parametry parametrów publicznych oraz algorytm Com , który pobiera parametry i komunikat $m \in \{0, 1\}^n$ i wysyła komunikat com ; sprawimy, że losowość używana przez Com będzie jawną i oznaczymy ją przez r . Nadawca zobowiązuje się do m , wybierając uniform r , obliczając $\text{com} := \text{Com}(\text{params}, m; r)$ i wysyłając go do odbiorcy. Nadawca może później anulować com i ujawnić m , wysyłając m, r do odbiorcy; odbiorca weryfikuje to sprawdzając, że $\text{Com}(\text{params}, m; r) = \text{com}$.

Ukrywanie się nieformalnie oznacza, że com nie ujawnia niczego na mój temat; wiązanie oznacza, że niemożliwe jest wygenerowanie zobowiązania com , które można otworzyć na dwa różne sposoby. Definiujemy teraz te właściwości formalnie.

Eksperyment ukrywania zaangażowania $\text{HidingA}, \text{Com}(n)$:

1. Generowane są parametry param $\text{Gen}(1^n)$.
2. Przeciwnik A otrzymuje parametry wejściowe i wysyła parę komunikatów $m_0, m_1 \in \{0, 1\}^n$.
3. Wybieramy uniform $b \in \{0, 1\}$ i obliczamy $\text{com} = \text{Com}(\text{params}, m_b; r)$.
4. Przeciwnik A otrzymuje com i wysyła bit b .
5. Wynik eksperymentu wynosi 1 wtedy i tylko wtedy, gdy $b = b$.

Eksperyment wiązania zobowiązania $\text{BindingA}, \text{Com}(n)$:

1. Generowane są parametry param $\text{Gen}(1^n)$.
 2. A ma dane wejściowe i wyjściowe ($\text{com}, m, r, m_3, \dots, R$).
- Wynik eksperymentu definiuje się jako 1 wtedy i tylko wtedy, gdy $m = m_3$ i $\text{Com}(\text{params}, m; r) = \text{com} = \text{Com}(\text{params}, m_3; r)$.

DEFINICJA 5.13 Schemat zobowiązania Com jest bezpieczny, jeśli dla wszystkich przeciwników ppt A istnieje pomijalna funkcja negl taka, że

$$\Pr_{I \sim \mathcal{I}} [\text{UkrywanieA}, \text{Com}(n)] = 1 - \frac{1}{2^{n/2}} + \text{negl}(n)$$

$$\Pr_{I \sim \mathcal{I}} [\text{WiązanieA}, \text{Com}(n)] = 1 - \text{negl}(n).$$

Łatwo jest skonstruować bezpieczny schemat zobowiązania na podstawie losowej wyroczni H . Aby zatwierdzić wiadomość m , nadawca wybiera uniform $r \in \{0, 1\}^n$ i wyprowadza $\text{com} := H(mr)$. (W modelu losowej wyroczni Gen i parametry nie są potrzebne, ponieważ w efekcie H służy jako publiczne parametry schematu.) Intuicyjnie ukrywanie wynika z faktu, że przeciwnik pyta $H(r)$.

tylko znikome prawdopodobieństwo (ponieważ r jest jednolitym ciągiem n -bitowym); jeśli nigdy nie zadaje zapytania w tej formie, to $H(mr)$ nie ujawnia niczego na temat m . Wiązanie wynika z faktu, że H jest odporna na kolizje.

Schematy zobowiązania można konstruować bez przypadkowych wyroczni (właściwie z funkcji jednokierunkowych), ale szczegółowe wykraczają poza zakres tej książki.

Referencje i dodatkowe lektury

Odporne na kolizje funkcje skrótu zostały formalnie zdefiniowane przez Damgård [52]. Dodatkową dyskusję dotyczącą pojęcia bezpieczeństwa funkcji skrótu poza odpornością na kolizje można znaleźć w [120, 150]. Transformatę Merkle'a-Damgård'a wprowadzili niezależnie Damgård i Merkle [53, 123].

HMAC został wprowadzony przez Bellare i in. [14] i później znormalizowane [131].

Atak urodzinowy na małą przestrzeń opisany w sekcji 5.4.2 opiera się na algorytmie Floyda znajdującym cykl. Powiązane algorytmy i wyniki opisano na stronie http://en.wikipedia.org/wiki/Cycle_detection. Pomysł znalezienia znaczących kolizji za pomocą ataku na małą przestrzeń jest autorstwa Yuvala [180]. Możliwość równoległych ataków mających na celu znalezienie kolizji, które w praktyce mogą zapewnić znaczne przyspieszenie, omówiono w [170]. Kompromisy czasu i przestrzeni w celu inwersji funkcji zostały wprowadzone przez Hellmana [87], a praktyczne ulepszenia – nie omawiane tutaj – zostały zaproponowane przez Rivesta (niepublikowane) i Oechslina [134].

Pierwsze formalne podejście do modelu losowej wyroczni przedstawili Bellare i Rogaway [21], chociaż pomysł wykorzystania funkcji „losowo wyglądającej” w zastosowaniach kryptograficznych sugerowali już wcześniej, w szczególności Fiat i Shamir [65]. Prawidłowe tworzenie losowej wyroczni w oparciu o konkretne kryptograficzne funkcje skrótu omówiono w [21, 22, 23, 48]. Przełomowym negatywnym wynikiem dotyczącym modelu losowej wyroczni jest wynik Canettiego i in. [41], którzy pokazują (wymyślone) schematy, które są bezpieczne w modelu losowej wyroczni, ale są niepewne w przypadku jakiejkolwiek konkretnej instancji losowej wyroczni.

Drzewa Merkle zostały wprowadzone w [121]. W praktyce stosowane funkcje wprowadzania kluczy obejmują HKDF, PBKDF2 i bcrypt. Zobacz [109], aby zapoznać się z formalnym podejściem do problemu i analizą HKDF.

Ćwiczenia

- 5.1 Podaj formalne definicje rezystancji drugiego obrazu wstępnego i oporu obrazu wstępnego. Udosadnij, że każda funkcja mieszająca odporna na kolizje jest odporna na drugi obraz wstępny, a każda funkcja skrótu odporna na drugi obraz wstępny jest odporna na obraz wstępny.

5.2 Niech $(\text{Gen}_1, \text{H}_1)$ i $(\text{Gen}_2, \text{H}_2)$ będą dą dwiema funkcjami skrótu. Zdefiniuj (Gen, H) tak, aby Gen uruchamiał Gen_1 i Gen_2 w celu uzyskania odpowiednio kluczy s_1 i s_2 .
 Następnie zdefiniuj $\text{Hs}_1, s_2(x) = \text{H}_1^{s_1}(x)\text{H}_2^{s_2}(x)$.

- (a) Udowodnić, że jeśli przynajmniej jedno z $(\text{Gen}_1, \text{H}_1)$ i $(\text{Gen}_2, \text{H}_2)$ jest kolizją odporny, wówczas (Gen, H) jest odporny na kolizje.
- (b) Ustal, czy analogiczne twierdzenie dotyczy odpowiednio drugiej odporności na przedobraz i odporności na przedobraz. W każdym przypadku uzasadnij swoją odpowiedź.

5.3 Niech (Gen, H) będzie odporną na kolizje funkcją skrótu. Czy $(\text{Gen}, \text{H}^\wedge)$ jest zdefiniowane przez $\text{H}^\wedge(s(x)) = \overset{\text{def}}{=} \text{Hs}(\text{Hs}(x))$ koniecznie odporne na kolizje?

5.4 Przedstaw formalny dowód Twierdzenia 5.4 (tj. opisz redukcję).

5.5 Uogólnić transformatę Merkle'a-Damgårdę (konstrukcja 5.3) dla przypadku, gdy funkcja mieszająca o stałej długości ma długość wejściową $n+k$ (przy $k > 0$) i długość wyjściową n , a długość wejściowa H powinna być zakodowane jako wartość -bitowa (jak omówiono w sekcji 5.3.2). Udowodnić odporność na kolizje (Gen, H) , zakładając odporność na kolizje (Gen, h) .

5.6 Dla każdej z poniższych modyfikacji transformaty Merkle'a - Damgårdę (Konstrukcja 5.3) należy określić, czy wynik jest odporny na kolizje. Jeżeli tak, proszę przedstawić dowód; jeśli nie, zademonstruj atak.

- (a) Zmodyfikuj konstrukcję tak, aby długość wejściowa w ogóle nie była uwzględniona (tj. wyjście zB , a nie $zB+1 = h s(zBL)$). Założmy, że wynikowa funkcja skrótu jest zdefiniowana tylko dla danych wejściowych, których długość jest całkowitą wielokrotnością długości bloku.)
- (b) Zmodyfikuj konstrukcję tak, aby zamiast wyprowadzać $z = h s(zBL)$, algorytm wyprowadzał zBL .
- (c) Zamiast używać IV, po prostu rozpoczęź obliczenia od x_1 . To znaczy zdefiniuj $z1 := x_1$, a następnie oblicz $zi := h s(zi-1xi)$ dla $i = B + 1$ i wyprowadź $zB+1$ jak poprzednio.
 $2, \dots,$
- (d) Zamiast używać ustalonego zbioru IV $z0 := L'$ i następnie obliczać $zi := h s(zi-1xi)$ dla $i = 1, \dots, B$ i wyjście zB .

5.7 Założymy, że istnieją odporne na kolizje funkcje mieszające. Pokaż konstrukcję funkcji skrótu o stałej długości (Gen, h) , która nie jest odporna na kolizje, ale taka, że funkcja mieszająca (Gen, H) uzyskana z Merkle'a-Damgårdą przekształca się w (Gen, h) , jak w konstrukcji 5.3 to odporność na kolizje tandemny.

5.8 Udowodnić lub obalić: jeśli (Gen, h) jest odporne na obraz wstępny, to taka sama jest funkcja mieszająca (Gen, H) otrzymywana poprzez zastosowanie transformaty Merkle'a - Damgårdę do (Gen, h) , jak w Konstrukcji 5.3.

5.9 Udowodnić lub obalić: jeśli (Gen, h) jest odporne na drugi obraz wstę pny, to funkcja skrótu (Gen, H) jest również oporna otrzymana poprzez zastosowanie transformaty Merkle'a - Damgård do (Gen, h) , jak w Konstrukcji 5.3.

5.10 Przed wprowadzeniem HMAC powszechnie było definiowanie MAC dla wiadomości o dowolnej długości na komputerach $\text{Mac}, k(m) = Hs(km)$, gdzie H jest odporną na kolizje funkcją skrótu.

(a) Pokaż, że nigdy nie jest to bezpieczny MAC, gdy H jest konstruowane za pomocą transformaty Merkle'a - Damgård. (Załóżmy, że atakujący zna klucz mieszający s , a jedynie k jest utrzymywane w tajemnicy.) (b) Udowodnij, że jest to bezpieczny adres MAC, jeśli H zamodeluje się jako losową wyrocznię .

5.11 Udowodnić, że konstrukcja funkcji pseudolosowej podana w rozdz. wersja 5.5.1 jest bezpieczna w modelu random-oracle.

5.12 Dowód twierdzenia 5.11.

5.13 Pokaż, jak znaleźć kolizję w konstrukcji drzewa Merkle'a, jeśli t nie jest ustalone. W szczególności pokaż, jak znaleźć dwa zestawy wejść x_1, \dots, x_t i $x_1, \dots, 2t$ taki, że $\text{MT } t(x_1, \dots, x_t) = \text{MT } 2t(x_1, \dots, 2t)$.

5.14 Rozważmy scenariusz przedstawiony w podrozdziale 5.6.2, w którym klient przechowuje pliki na serwerze i chce sprawdzić, czy zwarcane pliki są niezmodyfikowane.

(a) Podaj formalną definicję bezpieczeństwa dla tego ustawienia. (b) Sformalizuj konstrukcję w oparciu o drzewo Merkle, jak omówiono w Sekcji 5.6.2.

(c) Udowodnić, że konstrukcja jest bezpieczna w stosunku do definicji, przy założeniu, że (GenH, H) jest odporna na kolizje.

5.15 Udowodnić, że schemat zobowiązań omówiony w podrozdziale 5.6.5 jest bezpieczny w modelu losowej wyroczni.

Machine Translated by Google

Rozdział 6

Praktyczne konstrukcje

Elementy pierwotne z kluczem symetrycznym

W poprzednich rozdziałach pokazaliśmy, jak bezpieczne schematy szyfrowania i kody uwierzytelniania wiadomości można skonstruować na podstawie prymitywów kryptograficznych, takich jak generatory pseudolosowe, permutacje pseudolosowe i funkcje skrótu. Jednakże jednym z pytań, na które jeszcze nie odpowiedzieliśmy, jest przede wszystkim to, jak zbudowane są te prymitywy kryptograficzne, a nawet czy w ogóle istnieją! W następym rozdziale przeanalizujemy to pytanie z teoretycznego punktu widzenia i pokażemy konstrukcje generatorów pseudolosowych i permutacji pseudolosowych w oparciu o dość słabe założenia. (Okazuje się, że funkcje mieszające są trudniejsze do skonstruowania i wydają się wymagać silniejszych założeń. W sekcji 8.4.2 zobaczymy możliwą do udowodnienia konstrukcję funkcji skrótu.) W tym rozdziale skupimy się na heurystyce porównawczej, ale znacznie wydajniejsze konstrukcje tych prymitywów, które są szeroko stosowane w praktyce.

Jak już wspomniano, konstrukcje, które omówimy w tym rozdziale, są heurystyczne w tym sensie, że nie można udowodnić ich bezpieczeństwa w oparciu o żadne słabsze założenie. Konstrukcje te opierają się jednak na szeregu zasad projektowania, z których część można uzasadnić analizą teoretyczną. Co być może ważniejsze, wiele z tych konstrukcji przetrwało lata publicznej kontroli i prób kryptoanalizy, a biorąc to pod uwagę, całkiem rozsądne jest założenie, że te konstrukcje są bezpieczne.

W pewnym sensie nie ma zasadniczej różnicy między założeniem, powiedzmy, że rozkład na czynniki jest trudny, a założeniem, że AES (szynfr blokowy, który szczegółowo przeanalizujemy w dalszej części tego rozdziału) jest permutacją pseudolosową. Istnieje jednak znacząca różnica jakościowa pomiędzy tymi założeniami.¹ Podstawowa różnica polega na tym, że pierwsze założenie jest bardziej wiarygodne, ponieważ pozornie odnosi się do słabszego wymogu: założenie, że duże liczby całkowite są trudne do rozłożenia na czynniki, jest prawdopodobnie bardziej naturalne niż założenie, że AES z jednolitym kluczem jest nie do odróżnienia od losowej permutacji. Inne istotne różnice między założeniami polegają na tym, że faktoring był badany znacznie dłużej niż problem odróżnienia

¹ Powinno być jasne, że dyskusja w tym akapicie ma charakter nieformalny, ponieważ nie możemy formalnie dyskutować na ten temat, skoro nie możemy nawet udowodnić, że faktoring jest trudny!

i uznano go za trudny problem na długo przed pojawiением się opartych na nim schematów kryptograficznych.

Podsumowując, rozsądne jest założenie, że zalecane konstrukcje opisane w tym rozdziale są bezpieczne i ludzie mogą w praktyce polegać na takich założeniach. Mimo to lepiej byłoby oprzeć bezpieczeństwo prymitywów kryptograficznych na słabszych i bardziej długotrwałych założeniach.

Jak zobaczymy w rozdziale 7, jest to (w zasadzie) możliwe; niestety konstrukcje, które tam zobaczymy, są o rząd wielkości mniej wydajne od konstrukcji tu opisanych i jako takie nie sprawdzają się w praktyce.

Cel tego rozdziału

Głównymi celami tego rozdziału są (1) przedstawienie niektórych zasad projektowania stosowanych w konstrukcji współczesnych prymitywów kryptograficznych oraz (2) zapoznanie czytelnika z niektórymi popularnymi konstrukcjami szeroko stosowanymi w świecie rzeczywistym. Przestrzegamy, że:

- Celem tego rozdziału nie jest nauczenie czytelników, jak projektować nowe prymitywy kryptograficzne. Wręcz przeciwnie, wierzymy, że projektowanie nowych prymitywów wymaga znacznej wiedzy i wysiłku i nie można go podjąć lekko. Osobom zainteresowanym zdobyciem dodatkowej wiedzy specjalistycznej w tej dziedzinie zaleca się zapoznanie się z bardziej zaawansowanymi źródłami zawartymi na końcu rozdziału.

- Nie jest naszą intencją prezentowanie wszystkich niskopoziomowych szczegółów różnych prymitywów, które tu omawiamy i nie należy polegać na naszych opisach przy implementacji. W rzeczywistości nasze opisy są czasami celowo niedokładne, ponieważ pomijamy pewne szczegóły, które nie są istotne dla szerszego punktu pojęcia, który staramy się podkreślić.

6.1 Szyfry strumieniowe

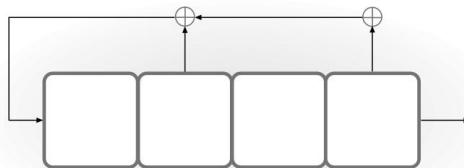
Przypomnijmy z sekcji 3.3.1, że szyfr strumieniowy jest definiowany przez dwa algorytmy deterministyczne (Init, GetBits). Algorytm Init przyjmuje jako dane wejściowe klucz k oraz (opcjonalny) wektor inicjujący IV i zwraca pewien stan początkowy st. Za pomocą algorytmu GetBits można wygenerować nieskończony strumień bitów y₁, y₂, ... w oparciu o ul. Głównym wymaganiem szyfru strumieniowego jest to, że powinien zachowywać się jak generator pseudolosowy, a mianowicie, gdy k jest wybrane jednolicie losowo, wynikowa sekwencja y₁, y₂, ... powinien być nie do odróżnienia od sekwencji jednolitych i niezależnych bitów przez dowolnego atakującego ograniczonego obliczeniowo.

(W sekcji 3.6.1 zauważliśmy, że szyfry strumieniowe muszą czasami spełniać surowsze wymagania bezpieczeństwa. Nie zajmujemy się tym tutaj wyraźnie).

Wskazaliśmy już (patrz koniec sekcji 3.5.1), że szyfry strumieniowe można łatwo skonstruować z szyfrów blokowych, które są silniejszym prymitywem. Główną motywacją do stosowania dedykowanych konstrukcji szyfru strumieniowego przedstawionych w tej sekcji jest wydajność, szczególnie w środowiskach o ograniczonych zasobach (np. w sprzęcie, w którym może wystąpić potrzeba utrzymania małej liczby bramek). Wykazano jednak ataki na kilka najnowszych konstrukcji szyfrów strumieniowych, a ich bezpieczeństwo wydaje się znacznie słabsze niż w przypadku szyfrów blokowych. Dlatego zalecamy, jeśli to możliwe, stosowanie szyfrów blokowych (prawdopodobnie w trybie szyfru strumieniowego).

6.1.1 Rejestry przesunięcia ze sprzężeniem zwrotnym liniowym

Zaczniemy od omówienia rejestrów przesuwnych ze sprzężeniem zwrotnym liniowym (LFSR). Były one używane w przeszłości do generowania liczb pseudolosowych, ponieważ są niezwykle wydajne w implementacji sprzętowej i generują dane wyjściowe o dobrych właściwościach statystycznych. Same jednak nie dają silnych kryptograficznie generatorów pseudolosowych i tak naprawdę pokażemy łatwy atak polegający na odzyskiwaniu klucza na LFSR. Niemniej jednak LFSR można wykorzystać jako składnik w budowaniu szyfrów strumieniowych o większym bezpieczeństwie.



RYSUNEK 6.1: Rejestr przesuwny ze sprzężeniem zwrotnym liniowym.

LFSR składa się z tablicy n rejestrów s_{n-1}, \dots, s_0 wraz z posuwem c_0 . (Patrz pętla Fig-back zbiór w współczynników sprzężenia zwrotnego c_{n-1}, \dots, c_0). Rozmiar tablicy określona przez nazywany jest stopniem LFSR. Każdy rejestr przechowuje pojedynczy bit, a stan LFSR w dowolnym momencie jest po prostu zbiorem bitów zawartych w rejestrach. Stan LFSR jest aktualizowany w każdym z serii „taktów zegara” poprzez przesunięcie wartości we wszystkich rejestrach w prawo i ustawienie nowej wartości lewego rejestru równej XOR pewnego podzbioru bieżącego rejestrów, których podzbiór jest określony przez $(t) \leq n-1, \dots, 0$,

współczynniki sprzężenia zwrotnego. Oznacza to, że jeśli stan w pewnym momencie t wynosi $s(t+1)$, to stan po następny takcie zegara wynosi $s^{(t+1)} = s_0, \dots, s_{n-1}$

$$\begin{aligned}
 s_{ja}^{(t+1)} &= s_i^{(t)}, & ja = 0, \dots, n-2 \\
 s_{n-1}^{(t+1)} &:= \sum_{i=0}^{n-1} c_i s_i^{(t)}.
 \end{aligned}$$

Rysunek 6.1 przedstawia LFSR stopnia 4, gdzie $c_0 = c_2 = 1$ i $c_1 = c_3 = 0$.

Przy każdym taktowaniu zegara LFSR wyprowadza wartość skrajnego prawego rejestru s0. Jeśli stan początkowy LFSR to s $\overset{(0)}{s_0} \ 1, \dots, \overset{(0)}{s_n}, \dots, 0$, pierwszych n bitów wyniku strumień są dokładnie s $\overset{(0)}{s_0} \ 0, \dots, \overset{(0)}{s_{n-1}}$. Nastę pny bit wyjściowy to s $(i-1) \overset{(1)}{s_n} \ 1 = \overset{n-1}{\underset{i=0}{\underset{\text{ci s i}}{\underset{\text{Nastę pnie}}{\underset{|}{\underset{j=0}{\underset{\text{tak =}}{\underset{j}{\underset{|}{\underset{c_j \ s_i \ n+j \ 1}}{\underset{|}{\underset{ja > n.}}}}}}}}}$.

Jeśli oznaczymy bity wyjściowe przez y_1, y_2, \dots , gdzie $y_i = s_i$,

$$\begin{array}{ll} y_i = s_i \quad \overset{(0)}{1}, & ja = 1, \dots, \quad N \\ & n-1 \\ \text{tak = } & c_j y_i \quad n+j \quad 1 \\ & j=0 & ja > n. \end{array}$$

Jako przykład wykorzystując LFSR z rysunku 6.1, jeśli stan początkowy wynosi (0, 0, 1, 1), to stany dla pierwszych kilku okresów czasu są następujące:

$$\begin{array}{l} (0, 0, 1, 1) \\ (1, 0, 0, 1) \\ (1, 1, 0, 0) \\ (1, 1, 1, 0) \\ (1, 1, 1, 1) \end{array}$$

a wyjściem (które można odczytać z prawej kolumny powyższego) jest strumień bitów 1, 1, 0, 0, 1,

Stan LFSR składa się z n bitów; w ten sposób LFSR może przejść przez co najwyżej 2^n możliwych stanów przed powtórzeniem. Kiedy stany się powtarzają, bity wyjściowe się powtarzają, co oznacza, że sekwencja wyjściowa zacznie się powtarzać po wygenerowaniu co najwyżej 2^n bitów wyjściowych. LFSR o maksymalnej długości przechodzi przez wszystkie $2^n - 1$ stany niezerowe przed powtórzeniem. (Zauważ, że jeśli kiedykolwiek zostanie zrealizowany stan zerowy, wówczas LFSR pozostanie w tym stanie na zawsze, dlatego go wykluczamy.) To, czy LFSR ma maksymalną długość, czy nie, zależy tylko od współczynników sprzężenia zwrotnego; jeśli jest to maksymalna długość, to po zainicjowaniu w dowolnym stanie niezerowym przejdzie przez wszystkie $2^n - 1$ stanów niezerowych. Dobre wiadomo, jak ustawić współczynniki sprzężenia zwrotnego, aby uzyskać LFSR o maksymalnej długości, chociaż szczegóły wykraczają poza zakres tej książki.

Ataki rekonstrukcyjne. Wynik LFSR o maksymalnej długości stopnia n ma dobre właściwości statystyczne; na przykład każdy n-bitowy ciąg występuje z mniej więcej tej równą częstotliwością w strumieniu wyjściowym LFSR. Niemniej jednak generatory LFSR nie są dobrymi generatorami pseudolosowymi do celów kryptograficznych, ponieważ ich dane wyjściowe są przewidywalne. Wynika to z faktu, że osoba atakująca może zrekonstruować cały stan LFSR stopnia n po zaobserwowaniu co najwyżej $2n$ bitów wyjściowych. Aby to zobaczyć, załóżmy, że zarówno stan początkowy, jak i współczynniki sprzężenia zwrotnego niektórych LFSR są nieznane. Pierwsze n bitów wyjściowych y_1, \dots, y_n LFSR dokładnie ujawniają stan początkowy. Biorąc pod uwagę kolejnych n bitów wyjściowych y_{n+1}, \dots, y_{2n} , atakujący może ustawić układ n równań liniowych w n

niewiadome c_0, \dots, c_{n-1} :

$$\begin{aligned} y_{n+1} &= c_n \quad 1 \quad y_n \quad \cdots \quad c_0 \quad y_1 \\ &\vdots \\ y_{2n} &= c_n \quad 1 \quad y_{2n-1} \quad \cdots \quad c_0 \quad y_n. \end{aligned}$$

Można wykazać, że powyższe równania są liniowo niezależne (modulo 2) dla LFSR o maksymalnej długości, a zatem jednoznacznie określają współczynniki sprzężenia zwrotnego. (Rozwiązywanie można efektywnie znaleźć za pomocą algebry liniowej.) Znając współczynniki sprzężenia zwrotnego, można łatwo obliczyć wszystkie kolejne bity wyjściowe LFSR.

6.1.2 Dodawanie nieliniowości

Liniowe relacje między bitami wyjściowymi LFSR są dokładnie tym, co umożliwia łatwy atak. Aby udaremnić takie ataki, musimy wprowadzić pewną nieliniowość, czyli pewne operacje inne niż XOR. Można to zrobić na kilka różnych sposobów, a my omówimy tutaj tylko niektóre z nich.

Nieliniowe sprzężenie zwrotne. Jednym z oczywistych sposobów modyfikacji LFSR jest uczynienie pełnego sprzężenia zwrotnego nieliniową. Rejestr przesuwny z nieliniowym sprzężeniem zwrotnym (FSR) będzie dalej ponownie składał się z tablicy rejestrów, z których każdy będzie zawierał pojedynczy bit. Tak jak poprzednio, stan FSR jest aktualizowany w każdym z serii taktów zegara poprzez przesunięcie wartości we wszystkich rejestrach w prawo; teraz jednak nowa wartość lewego rejestru jest nieliniową funkcją bieżących rejestrów. Innymi słowy (t) , jeśli stan w pewnym momencie t jest s , to stan po $(t+1)$ ($t+1$) taktie zegara jest $s \oplus g(s_0, \dots, s_{n-1})$,

$$\begin{aligned} s_{ja}^{(t+1)} &= s_{i+1}^{(t)}, & ja = 0, \dots, n-2 \\ s_n^{(t+1)} &= g(s_0, \dots, s_{n-1})^{(t)} \end{aligned}$$

dla jakiejś funkcji nieliniowej g . Tak jak poprzednio, FSR wprowadza wartość skrajnego prawa rejestru s_0 przy każdym taktowaniu zegara.

Możliwe jest zaprojektowanie nieliniowego FSR o maksymalnej długości i takiego, aby sygnał wyjściowy miał dobre właściwości statystyczne.

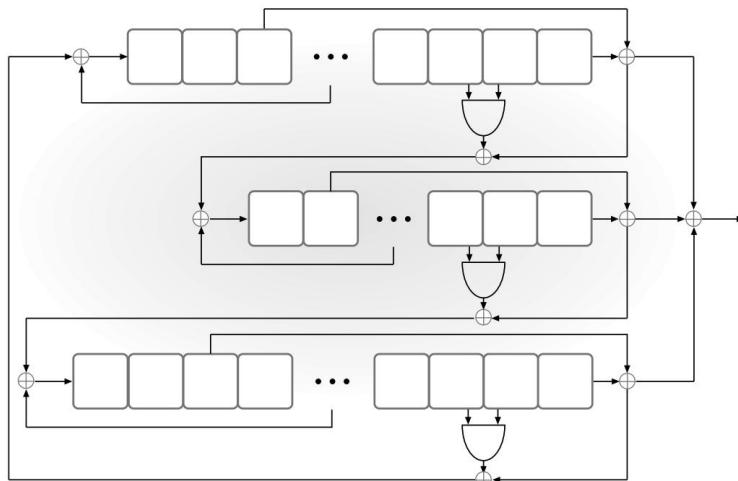
Generatory kombinacji nieliniowych. Innym podejściem jest wprowadzenie nieliniowości w sekwencji wyjściowej. W najbardziej podstawowym przypadku moglibyśmy mieć LFSR jak poprzednio (gdzie nowa wartość lewego rejestru jest ponownie obliczana jako funkcja liniowa rejestrów bieżących), ale gdzie sygnał wyjściowy przy każdym taktowaniu zegara jest funkcją nieliniową g wszystkich obecnych rejestrów, a nie tylko rejestr znajdujący się najbardziej na prawo. Ważne jest tutaj, aby g było zrównoważone w tym sensie, że $\text{Pr}[g(s_0, \dots, s_{n-1}) = 1] = 1/2$ (gdzie prawdopodobieństwo zależy od jednolitego wyboru s_0, \dots, s_{n-1}); w przeciwnym razie, chociaż może to być trudne

zrekonstruuje cały stan LFSR na podstawie sygnału wyjściowego, strumień wyjściowy będzie obciążony, a zatem będzie łatwy do odróżnienia od jednolitego.

Wariant powyższego polega na użyciu kilku LFSR (z każdym indywidualnym strumieniem wyjściowym obliczanym, jak poprzednio, po prostu poprzez pobranie wartości prawego rejestru każdego LFSR) i wygenerowaniu rzeczywistego strumienia wyjściowego poprzez połączenie wyjścia poszczególnych LFSR w jakiś nieliniowy sposób. Daje to tak zwany (nieliniowy) generator kombinacji. Poszczególne LFSR nie muszą mieć tego samego stopnia i w rzeczywistości długość cyklu generatora kombinowanego zostanie zmaksymalizowana, jeśli nie będą miały tego samego stopnia. W tym przypadku należy zadbać o to, aby strumień wyjściowy generatora kombinowanego nie był zbyt silnie skorelowany z żadnym ze strumieni wyjściowych poszczególnych LFSR; wysoka korelacja może prowadzić do ataków na poszczególne LFSR, tym samym mijając się z celem stosowania kilku LFSR w konstrukcji.

6.1.3 Trywium

Aby zilustrować pomysły z poprzedniej sekcji, krótko opisujemy szyfr strumieniowy Trivium. Ten szyfr strumieniowy został wybrany jako część projektu eSTREAM – europejskiego projektu zakończonego w 2008 roku, którego celem była identyfikacja nowych szyfrów strumieniowych. Trivium zostało zaprojektowane tak, aby mieć prosty opis i umożliwiać kompaktową implementację sprzętową.



RYSUNEK 6.2: Schematyczna ilustracja Trivium z (od góry do dołu) trzema sprzężonymi, nieliniowymi FSR A, B i C.

Trivium wykorzystuje trzy sprzężone, nieliniowe FSR oznaczone jako A, B i C i mające odpowiednio stopień 93, 84 i 111. (Patrz rysunek 6.2.) Stan Trivium to po prostu 288 bitów zawierających wartości we wszystkich rejestrach

tych FSR. Algorytm GetBits dla Trivium działa w następujący sposób: Przy każdym taktowaniu zegara wyjściem każdego FSR jest XOR jego skrajnego prawnego rejestru i jednego dodatkowego rejestru; wyjściem Trivium jest XOR bitów wyjściowych trzech FSR. Rejestry FSR są sprzężone: przy każdym taktie zegara nowa wartość lewego rejestru każdego FSR jest obliczana jako funkcja jednego z rejestrów w tym samym FSR i podzbioru rejestrów z drugiego FSR.

(Funkcja sprzężenia zwrotnego dla A zależy od jednego rejestru A i czterech rejestrów C; funkcja sprzężenia zwrotnego dla B zależy od jednego rejestru B i czterech rejestrów A; funkcja sprzężenia zwrotnego dla C zależy od jednego rejestru C i czterech rejestrów B.) Funkcja sprzężenia zwrotnego jest w każdym przypadku nieliniowa.

Algorytm Init Trivium akceptuje 80-bitowy klucz i 80-bitowy IV. Klucz jest ładowany do 80 skrajnych lewych rejestrów A, a IV jest ładowany do 80 skrajnych lewych rejestrów B. Pozostałe rejesty są ustawione na 0, z wyjątkiem trzech skrajnych prawnych rejestrów C, które są ustawione na 1. Następnie GetBits jest uruchamiane $4 \cdot 288$ razy (z odrzuconymi danymi wyjściowymi), a wynikowy stan jest przyjmowany jako st0.

Do chwili obecnej nie są znane żadne ataki kryptoanalityczne lepsze niż wyczerpujące wyszukiwanie kluczy przeciwko pełnemu szyfrowi Trivium.

6.1.4 RC4

LFSR są wydajne, gdy są zaimplementowane sprzężeniem, ale mają słabą wydajność w oprogramowaniu. Z tego powodu zbadano alternatywne projekty szyfrów strumieniowych. Wybitnym przykładem jest RC4, który został zaprojektowany przez Rona Rivesta w 1987 roku. RC4 jest niezwykły ze względu na swoją szybkość i prostotę i przez kilka lat opierał się na poważnym ataku. Jest ono dziś szeroko stosowane i dlatego omawiamy je z tego powodu; ostrzegamy jednak czytelnika, że ostatnie ataki wykazały poważne słabości kryptograficzne w RC4 i nie należy go już używać.

ALGORYTM 6.1 Algorytm inicjalizacji dla RC4 Wejście: 16-bajtowy klucz k Wyjście: Stan początkowy (S, i, j) (Uwaga: całe dodawanie odbywa się modulo 256) dla i = 0 do 255: S[i] := ik[i] := k[i mod 16] j := 0 dla i = 0 do 255: jot := jot + S[i] + k[i] Zamień S[i] i S[j] i := 0, j := 0 return (S, i, j)

Stan RC4 to 256-bajtowa tablica S, która zawsze zawiera permutację elementów $0, \dots, 255$, wraz z dwiema wartościami i, j $\in \{0, \dots, 255\}$. Init _

algorytm dla RC4 przedstawiono jako Algorytm 6.1. Dla uproszczenia zakładamy 16-bajtowy (128-bitowy) klucz k , chociaż algorytm może obsługiwać klucze o długości od 1 bajtu do 256 bajtów. Indeksujemy bajty S jako $S[0], \dots, S[255]$, a bajty klucza jako $k[0], \dots, k[15]$.

Podczas inicjalizacji, S jest najpierw ustawiane na permutację tożsamości (tj. $S[i] = i$ dla wszystkich i), a k jest rozszerzane do 256 bajtów przez powtarzanie. Następnie każdy wpis S jest zamieniany co najmniej raz z innym wpisem S w „pseudolosowym” miejscu. Indeksy i, j są ustawione na 0, a (S, i, j) jest wyprowadzane jako stan początkowy.

Stan jest następnie używany do generowania sekwencji bitów wyjściowych, jak pokazano w Algorytmie 6.2. Indeks i jest po prostu zwiększały (modulo 256), a j zmieniane jest w jakiś „pseudolosowy” sposób. Wpisy $S[i]$ i $S[j]$ są zamieniane i wyprowadzana jest wartość S na pozycji $S[i] + S[j]$ (ponownie obliczona modulo 256). Należy zauważać, że każdy wpis S jest zamieniany z innym wpisem S (prawdopodobnie z samym S) co najmniej raz na 256 iteracji, zapewniając dobry „wymieszanie” permutacji S .

ALGORYTM 6.2 Algorytm

GetBits dla RC4

Wejście: stan bieżący (S, i, j)

Wyjście: bajt wyjściowy y ; stan zaktualizowany (S, i, j)

(Uwaga: całe dodawanie odbywa się modulo 256)

$ja := ja + 1$

$j := j + S[i]$

Zamień $S[i]$ i $S[j]$ $t :=$

$S[i] + S[j]$ $y := S[t]$

powrót $(S,$

$i, j), y$

RC4 nie został zaprojektowany do przyjmowania IV jako wejścia; jednak w praktyce IV jest częstołączany przez proste połączenie go z rzeczywistym kluczem k przed inicjalizacją. Oznacza to, że wybierany jest losowy IV o żądanej długości, k jest ustawiane jako równe połączeniu IV i k (można to zrobić dodając lub dodając IV), a następnie uruchamia się Init jak w algorytmie 6.1 w celu wygenerowania stanu początkowego. Bity wyjściowe są następnie tworzone przy użyciu algorytmu 6.2 dokładnie tak, jak poprzednio. Zakładając, że RC4 jest używany w trybie niezsynchonizowanym (patrz sekcja 3.6.1), IV zostanie następnie przesyłany w postaci czystej do odbiornika – który prawdopodobnie ma już rzeczywisty klucz k – umożliwiając mu w ten sposób wygenerowanie tego samego stanu początkowego, a tym samym ten sam strumień wyjściowy. Ta metodałączenia IV jest stosowana w standardzie szyfrowania Wired Equivalent Privacy (WEP) do ochrony komunikacji w sieciach bezprzewodowych 802.11.

Należy zaniepokoić się tym stosunkowo ad hoc sposobem modyfikowania RC4 w celu zaakceptowania IV. Nawet gdyby RC4 był bezpiecznym szyfrem strumieniowym przy użyciu (tylko) klucza w oryginale zaprojektowanym, nie ma powodu sądzić, że powinno tak być

bezpieczne po modyfikacji w celu użycia kroplówki w ten sposób. Rzeczywiście, w przeciwnieństwie do klucza, IV jest ujawniany atakującemu (ponieważ jest wysyłany w sposób jawnym); co więcej, użycie różnych IV z tym samym stałym kluczem k —tak jak byłoby to zrobione przy użyciu RC4 w trybie niezsynchonizowanym —oznacza, że powiązane wartości k są używane do inicjacji stanu RC4. Jak zobaczymy poniżej, oba te problemy prowadzą do ataków, gdy RC4 jest używany w ten sposób.

Ataki na RC4. Chociaż RC4 jest wszechobecny w nowoczesnych systemach, od kilku lat znane są różne ataki na RC4. Z tego powodu nie należy już używać RC4; zamiast tego należy użyć bardziej nowoczesnego szyfru strumieniowego lub szyfru blokowego.

Zaczynamy od zademonstrowania prostego ataku statystycznego na RC4, który nie polega na tym, że uczciwe strony używają IV. Atak wykorzystuje fakt, że drugi bajt wyjściowy RC4 jest (nieznacznie) przesunięty w stronę 0. Niech S0 oznacza stan tablicy S po iteracjach GetBits, gdzie S0 oznacza stan początkowy. Traktując S0 (heurystycznie) jako jednolitą permutację $\{0, \dots, 255\}$, z prawdopodobieństwem $1/256 \cdot (1 - 1/255)$

$\underset{\text{def}}{1/256}$, przyjmuje się, że $S0[2] = 0$ i $X = S0[1] = 2$. Założymy na chwilę, że tak jest. W pierwszej iteracji GetBits wartość i jest zwięzla kiszana do 1, a j jest ustawiane jako równe $S0[i] = S0[1] = X$. Następnie $S0[1]$ i $S0[X]$ są zamieniane, tak że przy na końcu iteracji mamy $S1[X] = S0[1] = X$. W drugiej iteracji i jest zwięzla kiszane do 2, a j otrzymuje wartość

$$j + S1[i] = X + S1[2] = X + S0[2] = X,$$

ponieważ $S0[2] = 0$. Następnie S1[2] i S1[X] zamieniamy miejscami, tak że $S2[X] = S1[2] = S0[2] = 0$ i $S2[2] = S1[X] = X$. Na końcu wykonywana jest wartość S2 w pozycji $S2[i] + S2[j] = S2[2] + S2[X] = X$; jest to dokładnie wartość $S2[X] = 0$.

Gdy $S0[2] = 0$, drugi bajt wyjściowy jest równomierne rozłożony. Ogólnie, wówczas prawdopodobieństwo, że drugi bajt wyjściowy wynosi 0, wynosi

$$\begin{aligned} \Pr[S0[2] = 0 \text{ i } S0[1] = 2] &= \frac{1}{256} \cdot 1 \cdot \Pr[S0[2] = 0 \text{ i } S0[1] = 2] \cdot \frac{256}{256} \\ &= \frac{1}{256} \cdot \frac{1}{256} \cdot \frac{1}{256} \cdot \frac{256}{256} = \frac{1}{256}, \end{aligned}$$

lub dwa razy więcej, niż można się spodziewać w przypadku jednolitej wartości.

Samo powyższe nie może być postrzegane jako szczególnie poważny atak, chociaż wydaje się wskazywać na podstawowe problemy strukturalne RC4.

Poważniejszy atak na RC4 jest możliwy, gdy IV zostanie dołączony do klawisza. Atak ten może posłużyć do odzyskania klucza, niezależnie od jego długości, dlatego jest poważniejszy niż atak wyróżniający, taki jak opisany powyżej. Co ważne, atak ten może zostać wykorzystany do całkowitego złamania wspomnianego wcześniej standardu szyfrowania WEP i miał wpływ na wymianę tego standardu.

Istotą ataku jest sposób na rozszerzenie wiedzy o pierwszych n bajtach k do wiedzy o pierwszych $(n + 1)$ bajtach k. Należy pamiętać, że gdy IV jest

dodanych do rzeczywistego klucza k (więc $c = IV \oplus k$), kilka pierwszych bajtów k jest przekazywanych atakującemu za darmo! Jeśli IV ma długość n bajtów, wówczas przeciwnik może użyć tego ataku, aby najpierw odzyskać $(n+1)$ -szy bajt k (który jest pierwszym bajtem prawdziwego klucza k), następnie pnie następny bajt k i tak włączony, dopóki nie wydedukuje całego klucza.

Załóżmy, że IV ma długość 3 bajtów, tak jak w przypadku WEP. Atakujący czeka, aż pierwsze dwa bajty IV będą miały określona formę. Atak można przeprowadzić na kilka możliwości dla pierwszych dwóch bajtów IV , ale spójrzmy na przypadek, w którym IV przyjmuje postać $IV = (3, 255, X)$ dla X dowolnego bajtu. Oznacza to oczywiście, że $k[0] = 3$, $k[1] = 255$ i $k[2] = X$ w algorytmie 6.1. Można sprawdzić, że po pierwszych czterech iteracjach drugiej pętli Init mamy

$$S[0] = 3, S[1] = 0, S[3] = X + 6 + k[3]. \quad (6.1)$$

W następnych 252 iteracjach algorytmu Init jest zawsze więcej znaków niż 3. Zatem wartości $S[0]$, $S[1]$ i $S[3]$ nie są później modyfikowane, o ile ją nigdy nie przyjmuje wartości 0, 1 lub 3. Jeśli (heurystycznie) traktujemy ją jako przyjmujące jednakową wartość w każdej iteracji, oznacza to, że $S[0]$, $S[1]$ i $S[3]$ nie są później modyfikowane z prawdopodobieństwem $(253/256)^{252} \approx 0.05$, czyli 5% czasu. Zakładając, że tak jest, pierwszym bajtem wyprodukowanym przez GetBits będzie $S[3] = X + 6 + k[3]$; ponieważ X jest znane, ujawnia to $k[3]$.

Zatem atakujący wie, że w 5% przypadków pierwszy bajt wyniku jest skorelowany z $k[3]$, jak opisano powyżej. (Jest to znacznie lepsze niż losowe zgadywanie, które jest poprawne w $1/256 = 0.4\%$ przypadków). Zatem zbierając wystarczająco dużo próbek pierwszego bajtu wyniku – dla kilku IV mających poprawną formę – atakujący otrzymuje oszacowanie o wysokiej pewności dla $k[3]$.

6.2 Szyfry blokowe

Przypomnijmy sobie z sekcji 3.5.1, że szyfr blokowy jest wydajną permutacją z kluczem $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$. Oznacza to funkcję F_k zdefiniowaną przez $F_k(x) \stackrel{\text{def}}{=} F(k, x)$ jest bijekcją (tj. permutacją), a ponadto F_k i jego odwrotność F^{-1} są efektywnie obliczalne, biorąc pod uwagę k . Oznosimy się do n jako długości klucza k i jako długości bloku F i tutaj wyraźnie pozwalamy, aby się one różniły. Długość klucza i długość bloku są teraz stałymi stałymi, podczas gdy w rozdziale 3 były one postrzegane jako funkcje parametru bezpieczeństwa. To stawia nas w sytuacji bezpieczeństwa konkretnego, a nie asymptotycznego.² Konkretnie

² Chociaż szyfr blokowy ze stałą długością klucza nie ma „parametru bezpieczeństwa”, o którym można by mówić, nadal postrzegamy bezpieczeństwo jako zależne od długości klucza i dlatego oznaczamy tę wartość przez n .

wymagania bezpieczeństwa dotyczące szyfrów blokowych są dość rygorystyczne, a szyfr blokowy jest ogólnie uważany za „dobry” tylko wtedy, gdy najbardziej znany atak (bez wstępnego przetwarzania) ma złożoność czasową mniej więcej tej równą wyszukiwaniu klucza metodą brute-force. Zatem, jeśli szyfr o długości klucza $n = 256$ można złamać w czasie 2^{128} , szyfr jest (ogólnie), uważany za niepewny, mimo że atak trwający 2^{128} razy jest nadal niewykonalny. Natomiast w ustawieniu asymptotycznym atak o złożoności $2^{n/2}$ nie jest uważany za skuteczny, ponieważ wymaga czasu wykładniczego (a zatem szyfr, w którym taki atak jest możliwy, może nadal spełniać definicję bicia permutacją pseudolosową). Jednak w konkretnym kontekście musimy martwić się rzeczywistą złożonością ataku (a nie jego asymptotycznym zachowaniem). Ponadto istnieje obawa, że istnienie takiego ataku może wskazywać na bardziej fundamentalną słabość w konstrukcji szyfru.

Szyfry blokowe zaprojektowano tak, aby zachowywały się co najmniej jako (silne) permutacje pseudolosowe; patrz Definicja 3.28. Modelowanie szyfrów blokowych jako permutacji pseudolosowych pozwala na udowodnienie bezpieczeństwa konstrukcji opartych na szyfrach blokowych, a także wyjaśnia niezbędne wymagania szyfra blokowego. Solidne zrozumienie celów, jakie mają osiągnąć szyfry blokowe, ma kluczowe znaczenie w ich projektowaniu. Pogląd, że szyfry blokowe powinny być modelowane jako permutacje pseudolosowe, przynajmniej w niedawnej przeszłości, wywarł główny wpływ na ich konstrukcję. Przykładowo w zaproszeniu do składania wniosków dotyczących najnowszego zaawansowanego standardu szyfrowania (AES), z którym spotkamy się w dalszej części tego rozdziału, określono następujące kryterium oceny:

Bezpieczeństwo zapewniane przez algorytm jest najważniejszym czynnikiem. . . .

Algorytmy będą oceniane na podstawie następujących czynników. . . .

- Stopień, w jakim wynik algorytmu jest nieroźróżnialny z losowej permutacji. . .

Nowoczesne szyfry blokowe nadają się do wszystkich konstrukcji wykorzystujących permutacje pseudolosowe (lub funkcje pseudolosowe), które widzieliśmy w tej książce.

Często szyfry blokowe są projektowane (i zakładane) tak, aby spełniały jeszcze silniejsze właściwości bezpieczeństwa, co pokróćce omówimy w sekcji 6.3.1.

Niezależnie od faktu, że szyfry blokowe same w sobie nie są szyfrowaniem, schematowa terminologia dotycząca ataków na szyfr blokowy F jest następująca:

- W ataku ze znanym tekstem jawnym atakujący otrzymuje pary wejść/wyjść $\{(x_i, F_k(x_i))\}$ (dla nieznanego klucza k), przy czym $\{x_i\}$ jest poza kontrolą atakującego.
- W ataku z wybranym tekstem jawnym atakujący otrzymuje $\{F_k(x_i)\}$ (ponownie dla nieznanego klucza k) dla serii danych wejściowych $\{x_i\}$ wybranych przez atakującego. •

W ataku wybranym szyfrogramem atakujący otrzymuje $\{F_k(x_i)\}$ za $\{x_i\}$
wybrany przez atakującego, a także $\{F_k^{-1}(y_i)\}$ dla wybranego $\{y_i\}$.

Postrzeganie długości klucza jako parametru ma sens przy porównywaniu szyfrów blokowych o różnych długościach kluczy lub podczas używania szyfru blokowego obsługującego klucze o różnych długościach.

Oprócz wykorzystania powyższego do odróżnienia Fk od jednolitej permutacji, bę dziemy również zainteresowani atakami polegającymi na odzyskiwaniu klucza, w których atakujący jest w stanie odzyskać klucz k po interakcji z Fk. (To jest silniejsze niż umiejętnośc odróżnienia Fk od munduru.)

W odniesieniu do tej taksonomii permutacji pseudolosowej nie można odróżnić od permutacji jednolitej w przypadku ataku z wybranym tekstem jawnym, podczas gdy silnej permutacji pseudolosowej nie można odróżnić nawet w przypadku ataku z wybranym tekstem zaszyfrowanym.

6.2.1 Sieci substytucyjne-permutacyjne

Szyfr blokowy musi zachować się jak losowa permutacja. Są 2^n permutacje na ciągach n -bitowych, więc c reprezentowanie dowolnej permutacji z n -bitową długością bloku wymaga $\log(2^n) = n \cdot 2$ bitów. Jest to niepraktyczne dla $n > 20$ i niewykonalne dla $n > 50$. (Patrząc w przyszłość, współczesne szyfry blokowe mają długość bloków $n = 128$). Wyzwaniem podczas projektowania szyfru blokowego jest skonstruowanie zestawu permutacji ze związkim opisem (tj. krótkim kluczem), która zachowuje się jak losowa permutacja. W szczególności, tak jak ocena losowej permutacji na dwóch wejściach, które różnią się tylko jednym bitem, powinna dać dwa (prawie) niezależne wyniki (nie są one całkowicie niezależne, ponieważ nie mogą być równe), tak też zmiana jednego bitu wejścia na Fk (ϕ), gdzie ϕ jest jednolite i nieznane atakującemu, powinno dać (prawie) niezależny wynik. Oznacza to, że jednobitowa zmiana na wejściu powinna „wpływać” na każdy bit sygnału wyjściowego. (Zauważ, że nie oznacza to, że wszystkie bity wyjściowe zostaną zmienione — byłoby to inne zachowanie, niż można by się spodziewać w przypadku losowej permutacji. Raczej nieformalnie mamy na myśli, że każdy bit wyniku zmienia się z prawdopodobieństwem mniej więcej o połowę więcej niż tym).

Paradygmat zamieszania i dyfuzji. Oprócz swojej pracy nad doskonałą tajemnicą Shannon wprowadził także podstawowy paradygmat konstruowania związków, losowo wyglądających permutacji. Podstawową ideą jest skonstruowanie losowo wyglądającej permutacji F o dużej długości bloku z wielu mniejszych losowych (lub losowo wyglądających) permutacji $\{\phi_i\}$ o małej długości bloku. Zobaczmy, jak to działa na najbardziej podstawowym poziomie. Założymy, że chcemy, aby F miał długość bloku 128 bitów. Możemy zdefiniować F w następujący sposób: klucz k dla F bę dzie określał 16 permutacji ϕ_1, \dots, ϕ_{16} , z których każdy ma 8-bitową (1-bajtową) długość bloku.³ Biorąc pod uwagę dane wejściowe $x \in \{0, 1\}^{128}$, analizujemy je jako 16 bajtów $x_1 \cdots x_{16}$ i następnie ustaviamy

$$F_k(x) = \phi_1(x_1) \cdots \phi_{16}(x_{16}). \quad (6.2)$$

Mówiąc się, że te okrągłe funkcje $\{\phi_i\}$ wprowadzają zamieszanie do F.

³Dowolną permutację na 8 bitach można przedstawić za pomocą $\log(2^8) = 1600$ bitów, więc c dugość klucza dla F wynosi około $16 \cdot 1600$ bitów, czyli 3 kilobajty. To znacznie mniej niż 128 $\approx 128^2$ bitów, które byłyby wymagane do określenia dowolnej permutacji na 128 bitach.

Jednak od razu powinno być jasne, że F zdefiniowane powyżej nie będą działać pseudolosowe. W szczególności, jeśli x i x' różnią się tylko pierwszym bajtem, wówczas $F_k(x)$ i $F_k(x')$ będą dążyć różnić się tylko pierwszym bajtem (niezależnie od klucza k).

W przeciwnieństwie do tego, gdyby F było naprawdę losową permutacją, wówczas można by oczekwać, że zmiana pierwszego bitu danych wejściowych wpłynie na wszystkie bajty danych wyjściowych.

Z tego powodu wprowadza się etap dyfuzji, podczas którego bity wyjściowe są permutowane, czyli „mieszane”, przy użyciu permutacji mieszącej. Ma to wpływ na rozproszenie lokalnej zmiany (np. zmiany w pierwszym bajcie) na cały blok. Etapy zamieszania/dyfuzji – zwane łącznie rundą – powtarzane są wielokrotnie. Pomaga to zapewnić, że zmiana pojedynczego bitu na wejściu wpłynie na wszystkie bity na wyjściu.

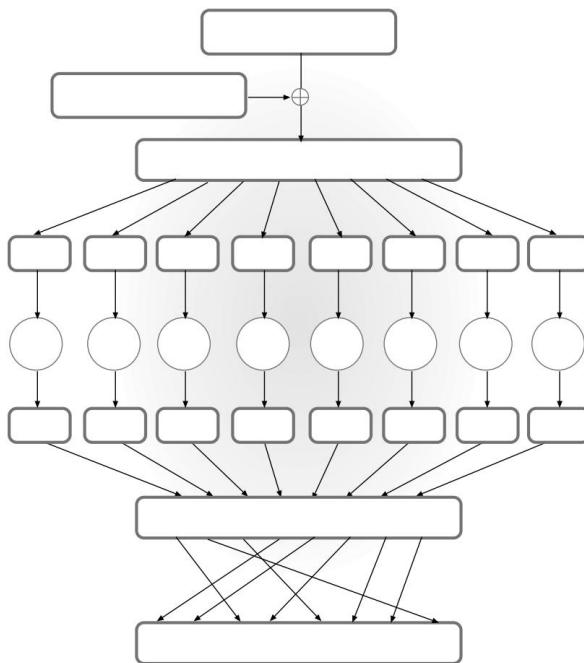
Na przykład dwurundowy szyfr blokowy zgodny z tym podejściem działały w następujący sposób. Po pierwsze, zamieszanie wprowadza się obliczając wynik pośredni $f_1(x_1) \dots f_{16}(x_{16})$ jak w równaniu (6.2). Bity wyniku są następnie „tasowane”, czyli ponownie porządkowane, aby otrzymać x . Wtedy $f_1 \dots f_{16}(x_{16})$ to $\dots x_{16}$, obliczone (gdzie $x = x$ 1 bity) korzystając z możliwie różnych funkcji f_i , oraz wyniku są permutowane, aby dać wynik x . $\{f_i\}$, $\{f\}$ permutacje miksujące mogą być losowe i zależne od klucza, jak opisaliśmy powyżej. W praktyce jednakże są one specjalnie zaprojektowane i zamocowane, a klucz jest wbudowany w inny sposób, jak opiszemy poniżej.

Sieci podstawieniowo-permutacyjne. Sieć substytucyjno-permutacyjną (SPN) można postrzegać jako bezpośrednią implementację paradygmatu zamieszania i dyfuzji. Różnica polega na tym, że teraz funkcje okrągłe mają określona postać, a nie są wybierane ze zbioru wszystkich możliwych permutacji w jakiejś dziedzinie. W szczególności, zamiast mieć (czyć) klucza k określającą dowolną permutację f , zamiast tego ustalamy publiczną „funkcję podstawienia” (tj. permutację) S zwaną S-boxem, a następnie pozwalamy k zdefiniować funkcję f dane przez $f(x) = S(k \oplus x)$.

Aby zobaczyć, jak to działa konkretnie, rozważ nazwę SPN z 64-bitową długością bloku opartą na zbiorze 8-bitowych (1-bajtowych) S-boxów S_1, \dots, S_8 . (Patrz rysunek 6.3.) Ocena szyfru przebiega w serii rund, przy czym w każdej rundzie stosujemy następującą sekwencję operacji na 64-bitowym wejściu x tej rundy (wejście do pierwszej rundy jest tylko wejściem do szyfru):

1. Mieszanie kluczy: Ustaw $x := x \oplus k$, gdzie k jest podkluczem bieżącej rundy;
2. Podstawienie: Ustaw $x := S_1(x_1) \dots S_8(x_8)$, gdzie x_i jest i -tym bajtem x ;
3. Permutacja: permutuj bity x , aby otrzymać wynik rundy.

Wyniki każdej rundy są podawane jako dane wejściowe do następnej rundy. Po ostatniej rundzie następuje ostatni etap miksuowania kluczy, a wynikiem jest wynik szyfru. (Zgodnie z zasadą Kerckhoffa zakładamy, że S-boxy i permutacje miksuowania są publiczne i znane każdemu atakującemu. Oznacza to, że bez końcowego etapu miksuowania kluczy ostatnie kroki podstawienia i permutacji nie zapewniły żadnych dodatkowych bezpieczeństwa, ponieważ nie zależą one od klucza.) Rysunek 6.4 pokazuje



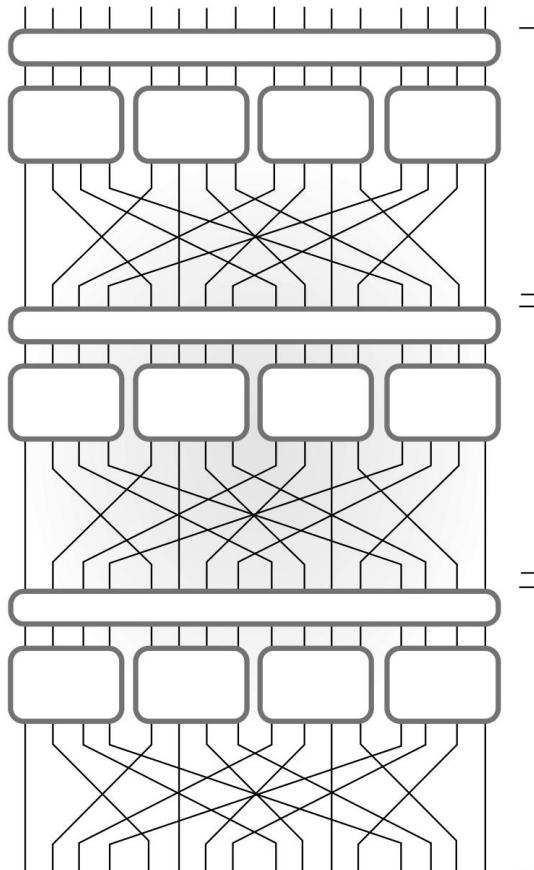
RYSUNEK 6.3: Pojedyncza runda sieci podstawieniowo-permutacyjnej.

struktura wysokiego poziomu nazwy SPN z 16-bitową długością bloku i innym zestawem 4-bitowych S-boxów używanych w każdej rundzie.

W każdej rundzie używane są różne podklucze (lub klawisze okrągłe). Rzeczywisty klucz szyfru blokowego jest czasami nazywany kluczem głównym. Okrągłe podklucze są wyprowadzane z klucza głównego zgodnie z harmonogramem kluczy. Harmonogram kluczy jest często prosty i może działać po prostu poprzez pobranie różnych podzbiorów bitów klucza głównego, chociaż można również zdefiniować bardziej złożone harmonogramy. R-okrągła nazwa SPN obejmuje r (pełne) rundy miksuowania klawiszy, podstawienie S-boxu i zastosowanie permutacji miksuowania, po czym następuje końcowy etap miksuowania kluczy. (Oznacza to, że w okrągłej nazwie SPN $r + 1$ używane są podklucze.)

Dowolną nazwę SPN można odwrócić (biorąc pod uwagę klucz). Aby to zobaczyć, pokazujemy, że biorąc pod uwagę wynik SPN i klucz, możliwe jest odzyskanie danych wejściowych. Wystarczy pokazać, że pojedynczą rundę można odwrócić; oznacza to, że cały SPN można odwrócić, przechodząc od ostatniej rundy z powrotem do początku. Ale odwrócenie pojedynczej rundy jest łatwe: permutację miksuowania można łatwo odwrócić, ponieważ jest to po prostu zmiana kolejności bitów. Ponieważ S-boxy są permutacjami (tj. jeden do jednego), one również mogą zostać odwrócone. Wynik można następnie poddać XOR za pomocą odpowiedniego podklucza, aby uzyskać oryginalne dane wejściowe. Dlatego:

TWIERDZENIE 6.3 Niech F będzie funkcją z kluczem zdefiniowaną przez SPN, w której wszystkie S-boxy są permutacjami. Wtedy niezależnie od kluczowego harmonogramu i liczby rund, F_k jest permutacją dowolnego k .



RYSUNEK 6.4: Sieć podstawieniowo-permutacyjna.

Liczba rund, wraz z dokładnym wyborem S-boxów, permutacji mikowania i harmonogramu kluczy, ostatecznie określają, czy dany szyfr blokowy jest łatwy do złamania lub wysoce bezpieczny. Omówimy teraz podstawową zasadę stojącą za projektowaniem S-boxów i permutacji mieszania.

Efekt lawiny. Jak wielokrotnie zauważono, ważną właściwością każdego szyfru blokowego jest to, że niewielka zmiana na wejściu musi „wypływać” na każdy bit wyniku. Nazywamy to efektem lawinowym. Jednym ze sposobów wywołania efektu lawinowego w sieci podstawień-permutacji jest zapewnienie, że spełnione są następujące dwie właściwości (i zostanie użyta wystarczająca liczba rund):

1. S-boxy są zaprojektowane w taki sposób, że zmiana jednego bitu wejściowego na S-box powoduje zmianę co najmniej dwóch bitów na wyjściu S-boxa.
2. Permutacje miksujące są zaprojektowane w taki sposób, że bity wyjściowe dowolnego danego S-boxa są używane jako dane wejściowe dla wielu S-boxów w następnej rundzie.

Aby zobaczyć, jak daje to efekt lawinowy, przynajmniej heurystycznie, założmy, że wszystkie S-boxy są takie, że zmiana pojedynczego bitu na wejściu S-boxa powoduje zmianę dokładnie dwóch bitów na wyjściu S-boxa i że permutacje mieszania zostały wybrane zgodnie z powyższymi wymaganiami. Dla konkretności założymy, że S-boxy mają rozmiar wejścia/wyjścia 8 bitów i że długość bloku szyfru wynosi 128 bitów. Zastanówmy się teraz, co się stanie, gdy szyfr blokowy zostanie zastosowany do dwóch wejść różniących się jednym bitem:

1. Po pierwszej rundzie wartości pośrednie różnią się dokładnie w dwóch pozycjach bitów. Dzieje się tak, ponieważ XORowanie bieżącego podklucza utrzymuje 1-bitową różnicę w wartościach pośrednich, a zatem wejścia do wszystkich S-boxów z wyjątkiem jednego są identyczne. W jednym S-boxie, w którym wejścia się różnią, sygnał wyjściowy S-boxa powoduje różnicę 2-bitową. Permutacja mieszania zastosowana do wyników zmienia położenie tych różnic, ale utrzymuje różnicę 2-bitową.
2. Permutacja mieszania zastosowana na końcu pierwszej rundy rozkłada dwie pozycje bitów, w których wyniki pośrednie różnią się, na dwa różne S-boxy w drugiej rundzie. Pozostaje to prawda nawet po wykonaniu XOR odpowiedniego podklucza z wynikiem poprzedniej rundy. Zatem w drugiej rundzie istnieją teraz dwa S-boxy, które otrzymują dane wejściowe różniące się o jeden bit. Zatem na koniec drugiej rundy wartości pośrednie różnią się o 4 bity.
3. Kontynuując ten sam argument, oczekujemy, że po 3. rundzie będzie to miało wpływ na 8 wartości pośrednich, po 4. rundzie będzie to miało wpływ na 16 bitów, a na koniec 7. rundy będzie to miało wpływ na wszystkie 128 bitów wartości wyjściowej.

Ostatni punkt nie jest do końca precyzyjny i z pewnością jest możliwe, że na koniec jakiejś rundy będzie mniej różnic, niż oczekiwano. (W rzeczywistości musi tak być, ponieważ dane wyjściowe również nie powinny różnić się wszystkimi bitami.) Z tego powodu zwyczajowo używa się znacznie więcej niż 7 rund. Jednakże powyższa analiza podaje dolną granicę liczby rund: jeśli używanych jest mniej niż 7 rund, musi istnieć pewien zestaw bitów wyjściowych, na które nie ma wpływu jednobitowa zmiana na wejściu, co oznacza, że będzie to możliwe jest odróżnienie szyfru od losowej permutacji.

Można się spodziewać, że „najlepszym” sposobem zaprojektowania S-boxów byłoby wybranie ich losowo (z zastrzeżeniem, że są to permutacje). Co ciekawe, okazuje się, że tak nie jest, przynajmniej jeśli chcemy spełnić wspomniane wcześniej kryterium konstrukcyjne. Rozważmy przypadek S-boxa działającego na wejściach 4-bitowych i niech x i x' będą dwiema różnymi wartościami. Niech $y = S(x)$, a teraz rozważmy wybór jednolitego $y' = y$ jako wartości $S(x')$. Istnieją 4 ciągi znaków, które różnią się od y tylko 1 bitem, więc z prawdopodobieństwem $4/15$ wybierzemy y' , które nie różni się od y dwoma lub więcej bitami. Problem komplikuje się, gdy weźmiemy pod uwagę wszystkie pary wejść, które różnią się jednym bitem.

Na podstawie tego przykładu dochodzimy do wniosku, że co do zasady skrzynki S-box należy projektować ostrożnie, a nie wybierać je na ślepo i losowo. Losowe S-boxy również nie nadają się do obrony przed atakami, takimi jak te, które pokażemy w sekcji 6.2.6.

Jeśli szyfr blokowy powinien być również silnie pseudolosowy, wówczas efekt lawinowy musi dotyczyć również jego odwrotności. Oznacza to, że zmiana pojedynczego bitu sygnału wyjściowego powinna wpłynąć na każdy bit sygnału wejściowego. W tym celu przydatne jest, jeśli S-boxy są zaprojektowane w taki sposób, że zmiana pojedynczego bitu na wyjściu S-boxa powoduje zmianę co najmniej dwóch bitów wejścia do S-boxa. Uzyskanie efektu lawiny w obu kierunkach to kolejny powód do dalszego zwięzania liczby strzałów.

Atakowanie SPN o zmniejszonej rundzie

Doświadczanie oraz wieloletnie wysiłki kryptoanalityczne wskazują, że sieci podstawieniowo-permutacyjne są dobrym wyborem do konstruowania permutacji pseudolosowych, pod warunkiem, że zachowią się ostrożność przy wyborze S-boxów, permutacji mieszania i harmonogramu kluczów. Zaawansowany standard szyfrowania, opisany w sekcji 6.2.5, ma podobną strukturę do opisanej powyżej sieci podstawieniowo-permutacyjnej i powszechnie uważa się, że jest silną permutacją pseudolosową.

Siła tak skonstruowanego szyfru F zależy w dużej mierze od liczby rund. Aby uzyskać lepszy wgląd w sieci substytucyjno-permutacyjne, zademonstrujemy ataki na nazwy SPN posiadające bardzo małą liczbę rund. Ataki te są proste, ale warto je zobaczyć, ponieważ niezbicie pokazują, dlaczego potrzebna jest duża liczba rund.

Banalny przypadek. Najpierw rozważmy trywialny przypadek, w którym F składa się z jednej pełnej rundy i nie ma końcowego etapu mieszania klawiszy. Pokazujemy, że przeciwnik mając tylko jedną parę wejście/wyjście (x, y) może łatwo poznać tajny klucz k , dla którego $y = F_k(x)$. Przeciwnik zaczyna od wartości wyjściowej y , a następnie odwraca permutację mikowania i S-boxy. Może to zrobić, jak wspomniano wcześniej, ponieważ pełna specyfikacja permutacji mikowania i S-boxów jest publiczna. Wartość pośrednia, którą oblicza przeciwnik, wynosi dokładnie $x \oplus k$ (zakładając, bez utraty ogólności, że klucz główny jest używany jako podklucz w jedynym rundzie sieci). Ponieważ przeciwnik zna również dane wejściowe x , może natychmiast wyprowadzić tajny klucz k . Jest to zatem całkowita przerwa.

Chociaż jest to trywialny atak, pokazuje, że w dowolnej sieci podstawieniowo-permutacyjnej nie można zapewnić bezpieczeństwa poprzez wykonanie podstawienia S-box lub zastosowanie permutacji mikującej po ostatecznym zmieszaniu podklucza.

Atakowanie jednorundowego SPN. Teraz mamy jedną pełną rundę, po której następuje etap mikowania kluczów. Dla konkretności zakładamy 64-bitową długość bloku i S-boxy o 8-bitowej (1-bajtowej) długości wejścia/wyjścia. Zakładamy, że w dwóch etapach mieszania kluczów używane są niezależne 64-bitowe podklucze k_1, k_2 , zatem klucz główny $k_1 k_2$ nazwy SPN ma długość 128 bitów.

Pierwszą obserwacją jest to, że możemy rozszerzyć atak z trywialnego przypadku powyżej, aby zapewnić tutaj atak polegający na odzyskaniu klucza, zużywając znacznie mniej niż 2128 pracy. Pomyśl jest następny: biorąc pod uwagę pojedynczą parę wejście/wyjście (x, y) , jak poprzednio, atakujący wylicza wszystkie możliwe wartości dla podklucza k_2 drugiej rundy. Dla każdej takiej wartości atakujący może odwrócić końcowy etap mikowania kluczy, aby uzyskać kandydującą wartość pośrednią y . Widzieliśmy powyżej, że mając dane wejściowe x i wyjściowe y (pełnej) rundy SPN, można łatwo zidentyfikować unikalny możliwy podklucz k_1 . Zatem dla każdego możliwego wyboru k_2 atakujący wyprowadza unikalne odpowiadające mu k_1 , dla którego k_1k_2 może być kluczem głównym. W ten sposób atakujący uzyskuje (w czasie 264) listę 264 możliwości klucza głównego. Można je zawęzić za pomocą dodatkowych par wejście/wyjście w około 64 2 dodatkowym czasie; patrz także poniżej.

Lepszy atak jest możliwy poprzez zauważenie, że poszczególne bity wyjściowe zależą tylko od części klucza głównego. Napraw daną parę wejście/wyjście (x, y) jak poprzednio. Teraz przeciwnik wyliczy wszystkie możliwe wartości pierwszego bajtu k_2 . Może wykonać XOR każdej takiej wartości z pierwszym bajtem y , aby otrzymać wartość kandydującą na wyjście pierwszego S-boxa. Odwracając ten S-box, atakujący poznaje wartość kandydującą do wprowadzenia tego S-boxa. Ponieważ wejściem do tego S-boxu jest XOR 8 bitów x i 8 bitów k_1 (gdzie pozycje tych bitów zależą od permutacji mikowania pierwszej rundy i są znane atakującemu), daje to wartość kandydującą dla 8 bitów k_1 .

Podsumowując: dla każdej wartości kandydującej pierwszego bajtu k_2 istnieje unikalna możliwa wartość odpowiadająca około 8 bitom k_1 . Inaczej mówiąc, oznacza to, że dla około 16 bitów klucza głównego atakujący zmniejszył liczbę możliwych wartości tych bitów z 216 do 28. Osoba atakująca może zestawić wszystkie możliwe wartości w 28 czasie. Można to powtórzyć dla każdego bajtu k_2 , uzyskując 8 list – każda zawierająca 28 wartości – które łącznie charakteryzują możliwe wartości całego klucza głównego. W ten sposób atakujący zmniejszył = 264, tak jak we wcześniejszym ataku. Liczba możliwych kluczy głównych do $(28) = 211$, co stanowi czas na wykonanie tej czynności wynosi teraz $8 \cdot 8$ zdecydowaną poprawę. Jednakże całkowity par wejścia/wyjścia, aby jeszcze bardziej zmniejszyć Osoba atakująca może użyć dodatkowych

ilość miejsc na możliwe klucze. Rozważ listę 28 możliwych wartości dla pewnego zestawu 16 bitów klucza głównego. Osoba atakująca wie, że prawidłowa wartość z tej listy musi być zgodna z dodatkowymi parami wejścia/wyjścia, których dowiaduje się atakujący. Z heurystycznego punktu widzenia każda niepoprawna wartość z listy jest zgodna z dodatkową parą wejście/wyjście (x, y) z prawdopodobieństwem nie większym niż losowe zgadywanie; ponieważ każda 16-bitowa wartość z tabeli może zostać użyta do obliczenia 1 bajtu wyniku, biorąc pod uwagę dane wejściowe x , spodziewamy się, że niepoprawna wartość będzie zgodna z rzeczywistym wynikiem z prawdopodobieństwem 2-8. Zatem niewielka liczba dodatkowych par wejście/wyjście wystarczy, aby zawęzić wszystkie tabele do jednej wartości w każdej, w którym to momencie znany jest cały klucz główny.

Jest tu ważna lekcja do odrobienia. Atak jest możliwy, ponieważ różne części klucza można odizolować od innych części. Zatem konieczna jest dalsza dyfuzja, aby mieć pewność, że wszystkie bity klucza wpływają na wszystkie bity wyniku. Aby tak się stało, potrzeba wielu rund.

Atakowanie dwurundowego SPN. Możliwe jest rozszerzenie powyższych pomysłów, aby zapewnić atak lepszy niż brutalna siła na dwurundową nazwę SPN przy użyciu niezależnych podkluczy w każdej rundzie; zostawiamy to jako ćwiczenie.

Zamiast tego po prostu zauważamy, że dwuokrągły SPN nie bę dzie dobrą permutacją pseudolosową. Tutaj opieramy się na wspomnianym wcześniej fakcie, że efekt lawinowy nie następuje już po dwóch rundach (oczywiście zależy to od długości bloku szyfru i długości wejścia/wyjścia S-boxów, ale przy rozsądnych parametrach tak właśnie bę dzie). Osoba atakująca może odróżnić dwurundową nazwę SPN od jednolitej permutacji, jeśli pozna wynik oceny nazwy SPN na dwóch wejściach, które różnią się jednym bitem: w dwurundowej nazwie SPN wiele bitów z dwóch wyjść bę dzie takich samych, coś, czego nie oczekuje się w przypadku losowej permutacji.)

6.2.2 Sieci Feistela

Sieci Feistela oferują inne podejście do konstruowania szyfrów blokowych. Zaletą sieci Feistela w porównaniu z sieciami podstawieniowo-permutacyjnymi jest to, że podstawowe funkcje używane w sieci Feistela —w przeciwieństwie do S-boxów używanych w nazwach SPN —nie muszą być odwracalne. Sieć Feistela umożliwia zatem skonstruowanie funkcji odwracalnej ze składowych nieodwracalnych. Jest to ważne, ponieważ dobry szyfr blokowy powinien zachowywać się „nieuporządkowany” (aby wyglądał losowo), jednak wymaganie, aby wszystkie elementy konstrukcji były odwracalne, z natury wprowadza strukturę. Wymaganie odwracalności wprowadza również dodatkowe ograniczenia dla S-boxów, czyniąc je trudniejszymi do zaprojektowania.

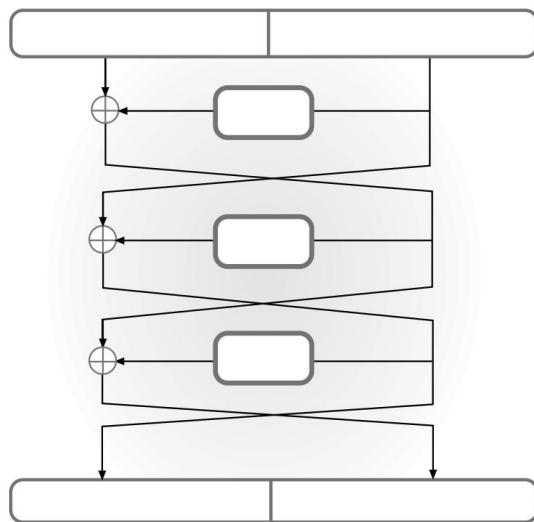
Sieć Feistela działa w serii rund. W każdej rundzie stosowana jest funkcja rundy z kluczem w sposób opisany poniżej. Funkcje okrągłe nie muszą być odwracalne. Zwykle bę dą zbudowane z komponentów takich jak S-boxy i permutacje miksujące, ale sieć Feistela może obsługiwać dowolne okrągłe funkcje, niezależnie od ich konstrukcji.

W zrównoważonej sieci Feistela (jedyny typ, który rozważymy) i-ta funkcja okrągła \hat{f}_i przyjmuje jako dane wejściowe podklucz k_i i ciąg /2-bitowy, a na wyjściu otrzymuje ciąg /2-bitowy. Podobnie jak w przypadku nazw SPN, klucz główny k służy do wyprowadzania podkluczy dla każdej rundy. Gdy zostanie wybrany jakiś klucz główny, wyznaczając w ten poprzez $f_i(R)$ każdy, definiujemy $f_i : \{0, 1\} /2 \xrightarrow{\text{def}} \hat{f}_i(k_i, R) : \{0, 1\} /2$ sposób = podklucz k_i . Należy zauważać, że funkcje rundy \hat{f}_i są stałe i publicznie znane, ale f_i zależą od klucza głównego i dlatego nie są znane atakującemu.

I-ta runda sieci Feistela działa w następujący sposób. Dane wejściowe rundy są podzielone na dwie połowy oznaczone L_{i-1} i R_{i-1} (odpowiednio „lewa” i „prawa” połowa). Jeśli długość bloku szyfru wynosi b bitów, wówczas L_{i-1} i R_{i-1} mają długość /2. Wynik (L_i, R_i) rundy wynosi

$$L_i := R_{i-1} \quad \text{i} \quad R_i := L_{i-1} \quad \hat{f}_i(R_{i-1}). \quad (6.3)$$

W r-okrągłej sieci Feistela, -bit wejściowy do sieci jest analizowany jako (L_0, R_0), a na wyjściu jest wartość -bitowa (L_r, R_r) uzyskana po zastosowaniu wszystkich r rund. Na rysunku 6.5 pokazano trójkątną sieć Feistela.



RYSUNEK 6.5: Trójkątna sieć Feistela.

Odwracanie sieci Feistela. Sieć Feistela jest odwracalna niezależnie od $\{f_i\}$ (a zatem niezależnie od funkcji okrągłych $\{\hat{f}_i\}$). Aby to pokazać, wystarczy pokazać, że każdą rundę sieci można odwrócić, jeśli znane są $\{f_i\}$. Mając wynik (L_i, R_i) i-tego rundy, możemy obliczyć (L_{i-1}, R_{i-1}) w następujący sposób: pierwszy zbiór $R_{i-1} := L_i$. Następnie oblicz

$$L_{i-1} := R_i \oplus f_i(R_{i-1}).$$

Daje to wartość (L_{i-1}, R_{i-1}) , która była wejściową tą rundą (tzn. oblicza odwrotność równania (6.3)). Należy zauważyć, że f_i jest oceniane tylko w kierunku do przodu, więc c nie musi być odwracalne. Mamy zatem:

TWIERDZENIE 6.4 Niech F będzie funkcją kluczową zdefiniowaną przez sieć Feistela. Wtedy niezależnie od funkcji okrągłych $\{\hat{f}_i\}$ i liczby rund, F_k jest efektywnie odwracalną permutacją dla wszystkich k .

Podobnie jak w przypadku sieci substytucyjno-permutacyjnych, ataki na sieci Feistela są możliwe przy zbyt małej liczbie rund. Zobaczmy takie ataki, gdy będą dziedziny omawiać DES w następnej sekcji. Teoretyczne wyniki dotyczące bezpieczeństwa sieci Feistela omówiono w rozdziale 7.6.

6.2.3 DES – Standard szyfrowania danych

Standard szyfrowania danych (DES) został opracowany w latach 70 IBM (przy pomocy Agencji Bezpieczeństwa Narodowego) i przyjęty w 1977 r. jako

federalny standard przetwarzania informacji dla Stanów Zjednoczonych. W swojej podstawowej formie DES nie jest już uważany za bezpieczny ze względu na krótką długość klucza wynoszącą 56 bitów, co czyni go podatnym na ataki typu brute-force. Niemniej jednak pozostaje on dziś w powszechnym użyciu we wzmacnionej formie późniejszego DES, opisanej w sekcji 6.2.4.

DES ma ogromne znaczenie historyczne. Został poddany intensywnej analizie w społeczności kryptograficznej, prawdopodobnie bardziej niż jakikolwiek inny algorytm kryptograficzny w historii. Powszechnie panuje opinia, że poza długością klucza DES jest wyjątkowo dobrze zaprojektowany szyfrem. Rzeczywiście, nawet po wielu latach najbardziej znanym w praktyce atakiem na DES jest wyczerpujące przeszukanie wszystkich 256 możliwych kluczy. (Jak zobaczymy, istnieją ważne teoretyczne ataki na DES, które wymagają mniej obliczeń, jednakże ataki te zakładają pewne warunki, które wydają się trudne do zrealizowania w praktyce.)

W tej sekcji zapewniamy ogólny przegląd głównych komponentów DES. Zaznaczamy, że nie podamy pełnej specyfikacji, która będzie poprawna w każdym szczególe, a niektóre elementy projektu zostaną pominięte w naszym opisie.

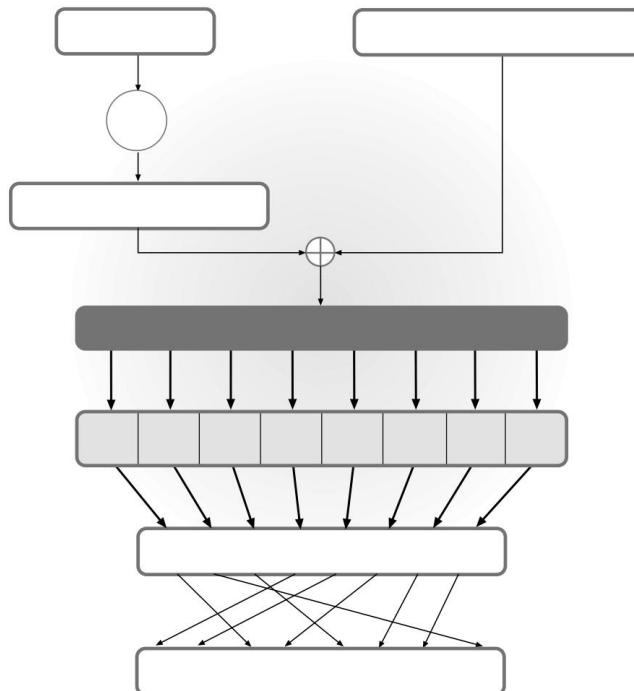
Naszym celem jest przedstawienie podstawowych idei leżących u podstaw konstrukcji DES, a nie wszystkich szczegółów niskiego poziomu; Czytelnik zainteresowany takimi szczegółami może zapoznać się z odnośnikami na końcu tego rozdziału.

Projekt DES

Szyfr blokowy DES to 16-okrągła sieć Feistela o długości bloku 64 bitów i długości klucza 56 bitów. Ta sama funkcja okrągła \hat{f} jest używana w każdej z 16 rund. Funkcja round przyjmuje 48-bitowy podklucz i , zgodnie z oczekiwaniemi dla (zbalansowanej) sieci Feistel, 32-bitowe wejście (mianowicie pół bloku). Schemat kluczy DES jest używany do wyprowadzenia sekwencji 48-bitowych podkluczy k_1, \dots, k_{16} z 56-bitowego klucza głównego. Schemat kluczy DES jest stosunkowo prosty, przy czym każdy podklucz k_i jest permutowanym podzbiorem 48 bitów klucza głównego.

Dla naszych celów wystarczy zauważyć, że 56 bitów klucza głównego jest podzielonych na dwie połówki – „lewą połowę” i „prawą połowę” – każda zawierająca 28 bitów. (Podział ten następuje po zastosowaniu początkowej permutacji klucza, ale ignorujemy to w naszym opisie.) W każdej rundzie 24 bity podklucza znajdujące się najbliżej na lewo są traktowane jako podzbiór 28 bitów po lewej stronie połowa klucza głównego i 24 bity znajdujące się najbliżej na prawo od okrągłego podklucza są traktowane jako podzbiór 28 bitów prawej połowy klucza głównego. Podkreślamy, że cały schemat klucza (w tym sposób podziału bitów na lewą i prawą połowę oraz które bity są wykorzystywane do tworzenia każdego podklucza k_i) jest stały i publiczny, a jedyną tajemnicą jest sam klucz główny.

Funkcja okrągła DES. Okrągła funkcja DES —często nazywana funkcją manglera DES —jest skonstruowana przy użyciu paradygmatu, który wcześniej analizowaliśmy: jest to (w zasadzie) po prostu sieć podstawień-permutacji! Bardziej szczegółowo, obliczenie $\hat{f}(k_i, R)$ z $k_i \in \{0, 1\}^{32}$ wygląda następująco: najpierw R jest rozszerzane do 48-bitowej $R' \in \{0, 1\}^{48}$ i $R' = E(R)$ gdzie



RYSUNEK 6.6: Funkcja manglera DES.

E nazywa się funkcją rozwinięcia cia. Następnie obliczenia przebiegają dokładnie tak, jak we wcześniejszym omówieniu nazw SPN: rozszerzona wartość R jest poddawana operacji XOR z k, która również ma 48 bitów, a wynikowa wartość jest dzielona na 8 bloków, z których każdy ma długość 6 bitów. Każdy blok przechodzi przez (inny) S-box, który przyjmuje 6-bitowe wejście i daje 4-bitowe wyjście; połączenie danych wyjściowych z 8 S-boxów daje wynik 32-bitowy. Następnie do bitów tego wyniku stosuje się permutację mieszania, aby uzyskać końcowy wynik. Patrz rysunek 6.6.

Jedna różnica w porównaniu z naszą pierwotną dyskusją na temat SPN polega na tym, że tutaj S-boxy nie są odwracalne; w istocie nie można ich odwrócić, ponieważ ich dane wejściowe są dłuższe niż dane wyjściowe. Dalsze omówienie szczegółów konstrukcyjnych S-boxów podano poniżej.

Jeszcze raz podkreślamy, że wszystko w powyższym opisie (łącznie z samymi S-boxami i permutacją mikowania) jest publicznie znane.

Jedynym sekretem jest klucz główny, z którego powstają wszystkie podklucze.

S-boxy i permutacja mieszająca. Osiem S-boxów tworzących „rdzeń” f jest kluczowym elementem konstrukcji DES i zostało bardzo starannie zaprojektowanych. Badania DES wykazały, że gdyby S-boxy zostały nieco zmodyfikowane, DES byłby znacznie bardziej podatny na atak.

Powinno to służyć jako ostrzeżenie dla każdego, kto chce zaprojektować szyfr blokowy: pozorne arbitralne wybory wcale nie są arbitralne, a jeśli nie zostaną wykonane prawidłowo, mogą sprawić, że cała konstrukcja będzie niepewna.

Przypomnijmy, że każdy S-box mapuje 6-bitowe wejście na 4-bitowe wyjście. Każde S-box można postrzegać jako tabelę z 4 wierszami i 16 kolumnami, gdzie każda komórka tabeli zawiera 4-bitowy wpis. Wejście 6-bitowe można postrzegać jako indeksowanie jednej z $2^6 = 64 = 4 \times 16$ komórek tabeli w następujący sposób: Pierwszy i ostatni bit wejściowy służą do wyboru wiersza tabeli, a bity 2–5 służą do aby wybrać kolumnę tabeli. 4-bitowy wpis w jakiejś pozycji tabeli reprezentuje wartość wyjściową dla wejścia powiązanego z tą pozycją.

S-boxy DES posiadają m.in. następujące właściwości:

1. Każdy S-box jest funkcją 4 do 1. (Oznacza to, że na każde możliwe wyjście odwzorowane są dokładnie 4 wejścia). Wynika to z poniższych właściwości.
2. Każdy wiersz tabeli zawiera każdy z 16 możliwych ciągów 4-bitowych dokładnie raz.
3. Zmiana jednego bitu dowolnego wejścia na S-box zawsze zmienia co najmniej dwa bity wyjścia.

Permutację mieszania również zaprojektowano starannie. W szczególności ma tę właściwość, że cztery bity wyjściowe z dowolnego S-boxa będą miały wpływ na wejście do sześciu S-boxów w następnej rundzie. (Jest to możliwe dzięki funkcji rozwinięcia, która jest stosowana w następnej rundzie przed obliczeniem S-boxów.)

Efekt lawinowy DES. Konstrukcja funkcji manglera zapewnia, że DES wykazuje silny efekt lawinowy. Aby to zobaczyć, prześledźmy różnicę między wartościami pośrednimi w obliczeniach DES dwóch wejść, które różnią się tylko jednym bitem. Oznaczmy dwa wejścia do szyfru przez (L_0, R_0) i $(L, \text{ gdzie } L \neq L_0)$. W obliczeniach DES występuje lewej połowie wejść (pomocne może być odniesienie się do równania (6.3) i rysunku 6.6 w dalszej części). Po pierwszej rundzie wartości pośrednie (L_1, R_1) i (L'_1, R'_1) , gdzie różnią się tylko jeden bit, R, choć teraz różnica ta jest w prawej połowie. W drugiej rundzie DES, prawa połowa każdego wejścia przechodzi przez f. Zakładając, że bit, w którym R1 się różni, nie jest powielany w kroku rozszerzania, wartości pośrednie przed zastosowaniem S-boxów nadal różnią się tylko o jeden bit. Zgodnie z właściwością 3 S-boxów, wartości pośrednie po obliczeniu S-boxów różnią się co najmniej dwoma bitami. W rezultacie wartości pośrednie (L_2, R_2) i (L'_2, R'_2) różnią się trzema bitami: istnieje 1-bitowa różnica pomiędzy L2 i L (przeniesiona z różnicą pomiędzy R1 i R1) i 2-bitowej różnicą pomiędzy R2 i R.

L_2, R_2

2

2.

Permutacja mieszania rozkłada dwubitową różnicę między R2 i R tak, że w następnej rundzie każdy z dwóch bitów jest używany jako dane wejściowe do innego S-boxa, co skutkuje różnicą co najmniej 4 bitów w prawych połówkach wartości pośrednich. (Jeśli jeden lub oba bity, w których R2 i R

2

różnią się są powielane przez E, różnica może być jeszcze więcej.) Istnieje teraz również 2-bitowa różnica w lewych połówkach. Podobnie jak w przypadku sieci podstawieniowo-permutacyjnej, mamy efekt wykładniczy, więc c po 7 rundach spodziewamy się, że bę będzie miało wpływ na wszystkie 32 bity w prawej połowie (a po 8 rundach bę będzie miało wpływ na wszystkie 32 bity w lewej połowie).

DES składa się z 16 nabojuów, zatem efekt lawinowy pojawia się na bardzo wcześniejszym etapie obliczeń. Zapewnia to, że obliczenie DES na podobnych danych wejściowych daje niezależnie wyglądające wyniki.

Ataki na DES o zmniejszonej rundzie

Przydatnym ćwiczeniem pozwalającym lepiej zrozumieć konstrukcję DES i jego bezpieczeństwo jest przyjrzenie się zachowaniu DES przy zaledwie kilku rundach. Pokażemy ataki na jedno-, dwu- i trzyrundowe warianty DES (przypomnijmy, że prawdziwy DES ma 16 rund). DES z trzema lub mniejszą liczbą rund nie może być funkcją pseudolosową, ponieważ trzy rundy nie wystarczą, aby wystąpił efekt lawinowy. Dlatego bęędziemy zainteresowani zademonstrowaniem bardziej skomplikowanych (i bardziej szkodliwych) ataków polegających na odzyskiwaniu klucza, podczas których obliczany jest klucz k przy użyciu tylko stosunkowo małej liczby par wejście/wyjście obliczonych przy użyciu tego klucza. Niektóre ataki są podobne do tych, które widzieliśmy w kontekście sieci podstawień-permutacji; tutaj jednak zobaczymy, jak można je zastosować do konkretnego szyfru blokowego, a nie do abstrakcyjnego projektu.

Poniższe ataki będą atakami ze znanym tekstem jawnym, w których przeciwnik zna pewne pary tekst jawnego/zaszyfrowany $\{(x_i, y_i)\}$ z $y_i = DES_k(x_i)$ dla jakiegoś tajnego klucza k . Opisując ataki, skoncentrujemy się na konkretnej parze wejście/wyjście (x, y) i opiszemy informację o kluczu, jaki przeciwnik może z tej pary wyprowadzić. Kontynuując notację opracowaną wcześniej, oznaczamy lewą i prawą połowę wejścia x odpowiednio jako L_0 i R_0 , a $L_i R_i$ oznaczamy lewą i prawą połowę po i -tym okrążeniu. Przypomnijmy, że E oznacza funkcję rozwiniętą dla DES, która oznacza podklucz użyty w rundzie i , a $f_i(R) = \hat{f}(k_i, R)$ oznacza rzeczywistą funkcję zastosowaną w sieci Feistela w i -tej rundzie.

Jednorundowy DES. Założymy, że mamy parę wejście/wyjście (x, y) . W jednorundowym DES mamy $y = (L_1, R_1)$, gdzie $L_1 = R_0$ i $R_1 = L_0 \oplus f_1(R_0)$.

Dlatego znamy parę wejście/wyjście dla f_1 ; konkretnie wiemy, że $f_1(R_0) = R_1 - L_0$. Stosując odwrotność permutacji miksującej do wyjścia $R_1 - L_0$, otrzymujemy wartość pośrednią składającą się z wyjść z wszystkich S-boxów, gdzie pierwsze 4 bity są wyjściem z pierwszego S-boxa, kolejne 4 bity są wyjściem z drugiego S-boxa i tak dalej.

Rozważmy (znane) 4-bitowe wyjście pierwszego S-boxa. Ponieważ każdy S-box jest funkcją 4 do 1, oznacza to, że istnieją dokładnie cztery możliwe wejścia do tego S-boxa, które dadzą dany wynik i podobnie dla wszystkich pozostałych S-boxów; każde takie wejście ma długość 6 bitów. Dane wejściowe do S-boxów to po prostu XOR $E(R_0)$ z podkluczem k_1 . Ponieważ R_0 , a co za tym idzie $E(R_0)$, jest znane, możemy to zrobić

obliczyć zbiór czterech możliwych wartości dla każdej 6-bitowej części k1. Oznacza to, że zmniejszyliśmy liczbę możliwych kluczy k1 z 248 do $448/6 = 48 = 216$ (ponieważ istnieją cztery możliwości dla każdej z ośmiu 6-bitowych części k1).

To już niewielka liczba, więc możemy po prostu wypróbować wszystkie możliwości na innej parze wejście/wyjście (x, y), aby znaleźć właściwy klucz. W ten sposób uzyskujemy klucz, używając tylko dwóch znanych tekstów jawnych w czasie mniej więcej tej samej 216.

Dwurundowy DES. W dwurundowym DES wynik y jest równy (L2, R2) gdzie

$$\begin{aligned} L_1 &= R_0 \\ R_1 &= L_0 \quad f_1(R_0) \\ L_2 &= R_1 = L_0 \quad f_1(R_0) \\ R_2 &= L_1 \quad f_2(R_1). \end{aligned}$$

L0, R0, L2 i R2 są znane z danej pary wejście/wyjście (x, y), a zatem wiemy również, że $L_1 = R_0$ i $R_1 = L_2$. Oznacza to, że znamy dane wejściowe/wyjściowe zarówno f_1 , jak i f_2 , zatem tę samą metodę zastosowaną w ataku na jednorundowy DES można tutaj zastosować do określenia zarówno k_1 , jak i k_2 w czasie w przybliżeniu $2 \cdot 2^{16}$. Ten atak działa nawet jeśli k_1 i k_2 są całkowicie niezależnymi kluczami, chociaż w rzeczywistości harmonogram kluczy DES zapewnia, że wiele bitów k_1 i k_2 jest równych (co można wykorzystać do dalszego przyspieszenia ataku).

Trzyrundowy DES. Odnosząc się do rysunku 6.5, wartość wyjściowa y jest teraz równa (L3, R3). Ponieważ $L_1 = R_0$ i $R_2 = L_3$, jedynymi nieznanymi wartościami na rysunku są R1 i L2 (które są równe).

Teraz nie mamy już wejścia/wyjścia żadnej funkcji okrągłej f_i . Na przykład wartość wyjściowa f_2 jest równa $L_1 - R_2$, gdzie obie te wartości są znane. Nie znamy jednak wartości R_1 wprowadzanej do f_2 .

Podobnie możemy określić dane wejściowe f_1 i f_3 , ale nie dane wyjściowe tych funkcji. Zatem atak, którego użyliśmy do przełamania jednorundowego i dwurundowego DES, nie sprawdzi się tutaj.

Zamiast polegać na pełnej wiedzy o wejściu i wyjściu jednej z funkcji okrągłych, wykorzystamy wiedzę o pewnej relacji pomiędzy wejściami i wyjściami f_1 i f_3 . Zauważ, że wynik f_1 jest równy $L_0 - R_1 = L_0 - L_2$, a wynik f_3 jest równy $L_2 - R_3$. Dlatego,

$$f_1(R_0) - f_3(R_2) = (L_0 - L_2) - (L_2 - R_3) = L_0 - R_3,$$

gdzie znane są zarówno L_0 , jak i R_3 . Oznacza to, że znany jest XOR wyjść f_1 i f_3 . Ponadto wejściem do f_1 jest R_0 , a wejściem do f_3 jest L_3 , przy czym oba są znane. Dochodzimy do wniosku, że możemy określić dane wejściowe f_1 i f_3 oraz XOR ich wyjść. Opiszemy teraz atak, który odnajduje tajny klucz na podstawie tych informacji.

Przypomnijmy, że harmonogram kluczy DES ma tę właściwość, że klucz główny jest podzielony na „lewą połowę”, którą oznaczamy k_L , i „prawą połowę” k_R , z których każda zawiera 28 bitów. Co więcej, 24 skrajnie lewe bity podklucza używanego w

każda runda jest pobierana tylko z kL, a 24 najbardziej na prawo bity każdego podklucza są pobierane tylko z kR. Oznacza to, że kL wpływa tylko na wejścia do pierwszych czterech S-boxów w dowolnej rundzie, podczas gdy kR wpływa tylko na wejścia do ostatnich czterech S-boxów. Ponieważ znana jest permutacja mieszania, wiemy również, które bity wyniku każdej funkcji okrągłej pochodzą z każdego S-boxa.

Ideą ataku jest raczej osobne przemierzanie kluczowej przestrzeni dla każdej 28 połowy klucza głównego, co daje atak o złożoności w przybliżeniu $2 \cdot 2$ niż złożoność 256. Taki atak będzie możliwy, jeśli uda nam się zweryfikować odgadnięcie połowy klucza głównego, a teraz pokażemy, jak można to zrobić. Założymy, że zgadujemy pewną wartość kL, lewej połowy klucza głównego. Znamy wartość wejściową R0 f1, więc korzystając z przypuszczenia kL, możemy obliczyć wartość wejściową dla pierwszych czterech S-boxów. Oznacza to, że możemy obliczyć połowę bitów wyjściowych f1 (permutacja mieszająca rozkładana znane nam bity, ale ponieważ permutacja mieszająca jest znana, wiemy dokładnie, które to bity). Podobnie możemy obliczyć te same lokalizacje na wyjściu f3, używając znanych danych wejściowych L3 do f3 i tego samego przypuszczenia dla kL. Na koniec możemy obliczyć XOR tych wartości wyjściowych i sprawdzić, czy odpowiadają one odpowiednim bitom znanej wartości XOR wyjść f1 i f3. Jeżeli nie są one równe, wówczas nasze przypuszczenie dotyczące kL jest błędne. Prawidłowe przypuszczenie dla kL zawsze przejdzie ten test, więc jeśli zostanie wyeliminowane, ale oczekuje się, że nieprawidłowe przypuszczenie przejdzie ten test tylko z prawdopodobieństwem w przybliżeniu 2-16 (ponieważ sprawdzamy równość 16 bitów w dwóch obliczonych wartościach). Istnieje 228 możliwych wartości kL, więc jeśli każda niepoprawna wartość pozostaje realnym kandydatem z prawdopodobieństwem 2-16, to spodziewamy się, że po powyższym pozostałe nam $2^{28} - 2 = 212$ możliwości dla kL.

Wykonując powyższe dla każdej połowy klucza głównego, otrzymujemy w czasie $28 \cdot 2$ około 212 kandydatów na lewą połowę i 212 kandydatów na prawą połowę. Ponieważ możliwa jest każda kombinacja lewej i prawej połowy, mamy w sumie 224 klucze kandydujące i możemy przeprowadzić wyszukiwanie brute-force w tym zestawie, używając dodatkowej pary wejście/wyjście (x, y). (Alternatywne podejście, bardziej efektywne, polega po prostu na powtórzeniu poprzedniego ataku przy użyciu 212 pozostałych kandydatów dla każdej połowy klucza). Złożoność czasowa ataku wynosi w przybliżeniu $2 \cdot 2^{28} + 224 < 2^{30}$, czyli znacznie mniej niż 256.

Bezpieczeństwo DES

Po prawie 30 latach intensywnych badań najbardziej znanym praktycznym atakiem na DES jest nadal wyczerpujące przeszukiwanie jego kluczowej przestrzeni. (Kilka ważnych teoretycznych ataków omawiamy w sekcji 6.2.6. Ataki te wymagają dużej liczby par wejście/wyjście, co byłoby trudne do uzyskania w ataku na jakikolwiek system w świecie rzeczywistym wykorzystującym DES.) Niestety, 56-bitowy długość klucza DES jest na tyle krótka, że możliwe jest teraz wyczerpujące przeszukanie wszystkich 256 możliwych kluczy. Już pod koniec lat 70. XX w. pojawiły się mocne zastrzeżenia co do wyboru tak krótkiego klucza dla DES. Wtedy sprzeciw był teoretyczny, ponieważ moc obliczeniowa wymagała przeszukania tak wielu obiektów

Klucze były ogólnie niedostępne.⁴ Jednakże praktyczność ataku brute-force na DES została wykazano w 1997 r., kiedy pierwsze z zestawu wyzwań DES postawionych przez RSA Security zostało rozwiązane w ramach projektu DESCHALL przy użyciu tysięcy komputerów skoordynowanych w całym Internet; obliczenia zajęły 96 dni. Drugie wyzwanie zostało pokonane w następny rok w ciągu zaledwie 41 dni przez projekt distribution.net. Znaczący przełom nastąpił w 1998 r., kiedy trzecie wyzwanie zostało rozwiązane w zaledwie 56 godzin. Tego imponującego wyczynu dokonano dzięki specjalnej maszynie do łamania DES o nazwie Deep Crack, która została zbudowana przez Electronic Frontier Foundation za 250 000 dolarów. W 1999 r. wyzwanie DES zostało rozwiązane w nieco ponad 22 godziny dzięki połączonemu wysiłkowi Deep Crack i Distributed.net. Najnowocześniejszym rozwiązaniem jest cracker DES firmy PICO Computing, który wykorzystuje 48 układów FPGA i potrafi znaleźć klucz DES w ciągu około 23 godzin.

Kompromisy czasoprzestrzenne omówione w sekcji 5.4.3 pokazują, że wyczerpujące ataki polegające na poszukiwaniu klucza można przyspieszyć, korzystając z obliczeń wstępnych i dodatkowej pamięci. Ze względu na krótką długość klucza DES, kompromisy czas/przestrzeń mogą być szczególnie skuteczne. W szczególności, stosując przetwarzanie wstępne, możliwe jest wygenerowanie tabeli o wielkości kilku terabajtów, która następnie umożliwia odzyskanie klucza DES z dużym prawdopodobieństwem z pojedynczej pary wejście/wyjście przy użyciu około 238 ocen DES (które można obliczyć w ciągu zaledwie minut). Najważniejsze jest to, że DES ma klucz, który jest zdecydowanie za krótki i nie można go obecnie uznać za bezpieczny w przypadku żadnej poważnej aplikacji.

Drugim powodem do niepokoju jest stosunkowo krótka długość bloku DES. Mała długość bloku jest problematyczna, ponieważ konkretne bezpieczeństwo wielu konstrukcji opartych na szyfrach blokowych zależy od długości bloku – nawet jeśli zastosowany szyfr jest „idealny”. Na przykład dowód bezpieczeństwa dla trybu CTR (por. Twierdzenie 3.32) pokazuje, że nawet w przypadku użycia całkowicie losowej funkcji atakujący może złamać bezpieczeństwo tego schematu szyfrowania z prawdopodobieństwem $2^{q/2}$, jeśli uzyska q zwykły tekst/pary szyfrogramów. W przypadku DES, gdzie $q = 64$, oznacza to, że jeśli atakujący uzyska tylko $q = 230$ par tekstu jawnego/tekstu zaszyfrowanego, bezpieczeństwo jest zagrożone z dużym prawdopodobieństwem. Uzyskanie par tekstu jawnego/zaszyfrowanego jest stosunkowo łatwe, jeśli przeciwnik podsłuchuje szyfrowanie wiadomości zawierających znane nagłówki, nadmiarowościami itp.

Niepewność DES nie ma nic wspólnego z jego konstrukcją jako taką, ale raczej wynika z krótkiej długości klucza (i, w mniejszym stopniu, krótkiej długości bloku).

Jest to wielki hołd dla projektantów DES, którym udało się skonstruować niemal „idealny” szyfr blokowy (pomijając zbyt krótki klucz).

Ponieważ wydaje się, że sam DES nie ma znaczących słabości strukturalnych, sensowne jest użycie DES jako elementu konstrukcyjnego do konstruowania szyfrów blokowych z dłuższymi kluczami. Omówimy to szerzej w sekcji 6.2.4.

⁴ W 1977 roku oszacowano, że zbudowanie komputera, który będzie w stanie złamać DES w jeden dzień, będzie kosztować 20 milionów dolarów.

Zastąpienie DES —zaawansowany standard szyfrowania (AES), omówiony w dalszej części tego rozdziału —zostało wyraźnie zaprojektowane, aby rozwiązać obawy dotyczące krótkiej długości klucza i długości bloku DES. AES obsługuje klucze 128-, 192- lub 256-bitowe i długość bloku 128 bitów.

Ataki na DES lepsze niż brutalna siła zostały po raz pierwszy zaprezentowane na początku lat 90. XX wieku przez Biham i Shamira, którzy opracowali technikę zwaną różnicową kryptoanalizą. Ich atak zajmuje 237 czasu i wymaga 247 wybranych tekstów jawnego. Chociaż atak był przełomowy z teoretycznego punktu widzenia, nie wydaje się budzić wiele kłopotów w praktyce, ponieważ trudno sobie wyobrazić realistyczny scenariusz, w którym przeciwnik może uzyskać tak wiele szyfrowań wybranych tekstów jawnego.

Co ciekawe, prace Biham i Shamira wykazały, że S-boxy DES zostały specjalnie zaprojektowane tak, aby były odporne na różnicową kryptoanalizę, co sugeruje, że technika kryptoanalizy różnicowej była znana (ale nie ujawniona publicznie) przez projektantów DES. Po ogłoszeniu wyniku przez Biham i Shamira podejrzenie to potwierdziło się.

Kryptanaliza liniowa została opracowana przez Matsui w połowie lat 90. XX wieku i została z powodzeniem zastosowana również w DES. Zaletą tego ataku jest to, że wykorzystuje znane teksty jawnego, a nie wybrane teksty jawnego. Niemniej jednak liczba wymaganych par tekstu jawnego/zaszyfrowanego – około 243 – jest nadal ogromna.

Krótko opisujemy kryptoanalizę różnicową i liniową w rozdziale 6.2.6.

6.2.4 3DES: Zwiększenie długości klucza szyfru blokowego

Główną słabością DES jest jego krótki klucz. Dlatego sensowne jest zaprojektowanie szyfru blokowego o większej długości klucza, wykorzystując DES jako element konstrukcyjny. Niektóre podejścia do tego omówiono w tej sekcji. Chociaż w całej dyskusji często odwołujemy się do DES, a DES jest najważniejszym szyfrem blokowym, do którego zastosowano te techniki, wszystko, co tutaj powiemy, odnosi się ogólnie do dowolnego szyfru blokowego.

Modyfikacje wewnętrzne a konstrukcje „czarnej skrzynki”. Istnieją dwa ogólne podejścia, które można zastosować do skonstruowania kolejnego szyfru opartego na DES. Pierwszym podejściem byłoby zmodyfikowanie w jakiś sposób wewnętrznej struktury DES przy jednoczesnym zwiększeniu długości klucza. Na przykład można pozostawić funkcję rundy nikt nie tą i po prostu użyć 128-bitowego klucza głównego z innym harmonogramem kluczy (nadal wybierając 48-bitowy podklucz w każdej rundzie). Można też zmienić same S-boxy i używać większego podklucza w każdej rundzie. Wadą takich podejść jest to, że modyfikując DES – nawet w najmniejszym stopniu – tracimy zaufanie, jakie do niego zdobyliśmy dzięki temu, że przez tyle lat pozostawał on odporny na ataki. Konstrukcje kryptograficzne są bardzo wrażliwe i nawet łagodne, pozornie nieistotne zmiany mogą sprawić, że konstrukcja będzie całkowicie niepewna.⁵ Dlatego nie zaleca się zmianiania elementów wewnętrznych szyfru blokowego.

⁵ W rzeczywistości wykazano różne wyniki w tym zakresie dla DES; np. zmiana S-boxów lub permutacji mikowania może sprawić, że DES będzie znacznie bardziej podatny na atak.

Alternatywnym podejściem, w którym nie występuje powyższy problem, jest wykorzystanie DES jako „czarnej skrzynki” i w ogóle nie dotykanie jej wewnętrznej struktury.

W tym podejściu traktujemy DES jako „idealny” szyfr blokowy z 56-bitowym kluczem i konstruujemy nowy szyfr blokowy, który odwołuje się jedynie do oryginalnego, niezmodyfikowanego DES. Ponieważ sam DES nie jest modyfikowany, jest to znacznie ostrożniejsze podejście i to właśnie ono będzie dzielić tutaj stosować.

Podwójne szyfrowanie

Niech F będzie dziesiętnym blokowym szyfrem o n -bitowej długości klucza i -bitowej długości bloku. Następnie można zdefiniować nowy szyfr blokowy F z kluczem o długości $2n$

$$F_{k1, k2}(X) \stackrel{\text{def}}{=} F_{k2}(F_{k1}(X)),$$

gdzie $k1$ i $k2$ są kluczami niezależnymi. W przypadku, gdy F jest DES, otrzymujemy szyfr F o nazwie 2DES, który przyjmuje klucz 112-bitowy; gdyby wyczerpujące wyszukiwanie klucza było najlepszym dostępnym atakiem, wystarczyłaby długość klucza wynosząca 112 bitów, ponieważ atak wymagający czasu 2^{112} jest całkowicie poza zasięgiem. Niestety, teraz pokazujemy atak na F , który przebiega w czasie około $2n$, znacznie krótszym niż czas 2^{2n} , jaki byłby niezbędny do przeprowadzenia wyczerpującego wyszukiwania $2n$ -bitowego klucza. Oznacza to, że nowy szyfr blokowy w zasadzie nie jest lepszy od starego, mimo że ma dwukrotnie dłuższy klucz.⁶

Atak nazywany jest „atakiem spotkania w środku” z powodów, które wkrótce staną się jasne. Założymy, że przeciwnik otrzymuje pojedynczą parę wejście/wyjście (x, y) , gdzie $y = F(x)$ dla nieznanego k . Moga zdarzyć się 2^n możliwych kluczy w $k \in \{0, 1\}^n$. Przeciwnik 1, następny pujący sposób:

1. Dla każdego $k1 \in \{0, 1\}^n$ oblicz $z := F_{k1}(x)$ i zapisz $(z, k1)$ na liście L .

2. Dla każdego $k2 \in \{0, 1\}^n$ oblicz $z := F_{k2}^{-1}(y)$ i zapisz $(z, k2)$ na liście L .

3. Wpisz $(z1, k1) \in L$ i $(z2, k2) \in L$ są zgodne, jeśli $z1 = z2$. Dla każdego takiego dopasowania dodaj $(k1, k2)$ do zbioru S . (Dopasowania można łatwo znaleźć po posortowaniu L i L według ich pierwszych składników).

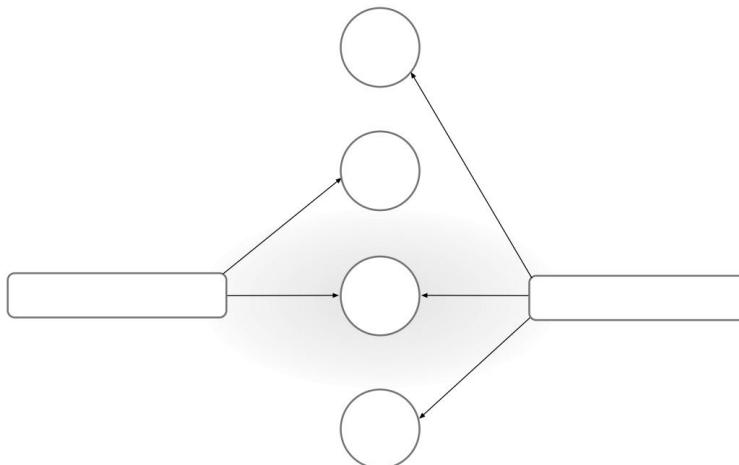
Graficzne przedstawienie ataku przedstawiono na rysunku 6.7.

Atak zajmuje czas $O(n \cdot 2^n)$ i wymaga przestrzeni $O((n + 1) \cdot 2^n)$. Zbiór S wyjścia tego algorytmu zawiera dokładnie te wartości $(k1, k2)$, dla których

$$F_{k1}(x) = F_{k2}^{-1}(y) \quad (6.4)$$

lub równoważnie, dla którego $y = F_{k1, k2}(x)$. W szczególności $(k1, k2) \in S$. Na k. z drugiej strony para $(k1, k2) = (k_1, k_2)$ oczekuje się (heurystycznie), że spełni $y = F_{k1}(x)$. Równanie (6.4) z prawdopodobieństwem 2, jeśli potraktujemy $F_{k1}(x)$ i $F_{k2}^{-1}(y)$ jako jednorodny

⁶ Nie jest to do końca prawdziwe, gdyż atak brute-force na F może zostać przeprowadzony w czasie 2^n i stałe pamięć ci, podczas gdy atak, który pokazujemy na F wymaga $2n$ czasu i 2. Niemniej jednak atak N^n pamięć č. pokazuje, że F nie osiąga pożądanego poziomu bezpieczeństwa.



RYSUNEK 6.7: Atak typu „spotkanie w środku”.

-bitowe ciągi znaków, zatem oczekiwany rozmiar S wynosi $22n \cdot 2^{2n} = 22n^{2n}$. Używając kilku kolejnych par wejście/wyjście i biorąc przecięcie uzyskanych zbiorów, poprawny (k, 1, $\frac{1}{2^n}$) można zidentyfikować z bardzo dużym prawdopodobieństwem.

Potrójne szyfrowanie

Oczywistym uogólnieniem powyższego podejścia jest zastosowanie szyfru blokowego trzy razy z rzędu. Powszechnie są dwa warianty tego podejścia:

Wariant 1: trzy klucze. Wybierz trzy niezależne klucze k_1, k_2, k_3 i zdefiniuj

$$F_{k_1, k_2, k_3}(X) \stackrel{\text{def}}{=} F_{k_3}(F_{k_2}^{-1}(F_{k_1}(X))).$$

Wariant 2: dwa klucze. Wybierz dwa niezależne klucze k_1, k_2 i następnie zdefiniuj

$$F_{k_1, k_2}(X) \stackrel{\text{def}}{=} F_{k_1}(F_{k_2}^{-1}(F_{k_1}(X))).$$

Przed porównaniem bezpieczeństwa obu alternatyw zauważamy, że środkowe wywołanie F jest odwrócone. Jeśli F jest wystarczająco dobrym szyfrem, nie ma to znaczenia, jeśli chodzi o bezpieczeństwo, ponieważ jeśli F jest silną permutacją pseudolosową, to F też musi być. Powodem odwrócenia drugiego zastosowania F jest uzyskanie kompatybilności wstępnej: jeśli ustawiemy $k_1 = k_2 = k_3$, wynikowa funkcja będzie równoważna pojedynczemu wywołaniu F przy użyciu klucza k_1 .

Bezpieczeństwo pierwszego wariantu. Długość klucza pierwszego wariantu wynosi $3n$, więc możemy mieć nadzieję, że najlepszy atak na ten szyfr wymagałby czasu $23n$.

Jednak szyfr jest podatny na atak typu „spotkaj się w środku”, podobnie jak w przypadku podwójnego szyfrowania, chociaż atak zajmuje teraz czas $22n$. To najbardziej znany atak. Zatem chociaż ten wariant nie jest tak bezpieczny, jak można by się spodziewać, zapewnia on wystarczające bezpieczeństwo ze wszystkich praktycznych powodów nawet dla $n = 56$ (zakładając oczywiście, że oryginalny szyfr F nie ma słabych punktów).

Bezpieczeństwo drugiego wariantu. Długość klucza w tym wariantie wynosi $2n$, zatem najlepsze, na co możemy liczyć, to zabezpieczenie przed atakami trwającymi w czasie 2^{2n} . Nie jest znany atak o wiele kszej złożoności czasowej, w którym przeciwnik otrzymuje tylko niewielką liczbę par wejście/wyjście. (Zobacz ćwiczenie 6.13, aby zapoznać się z atakiem wykorzystującym $2n$ wybranych tekstów jawnych.) Zatem potrójne szyfrowanie dwukluczowe jest rozsądny wyborem w praktyce.

Potrójny DES (3DES). Triple-DES (lub 3DES) opiera się na potrójnym wywołaniu DES przy użyciu dwóch lub trzech klawiszy, jak opisano powyżej. 3DES został ustandaryzowany w 1999 roku i jest obecnie szeroko stosowany. Jego głównymi wadami są stosunkowo mała długość bloku i fakt, że jest stosunkowo powolny, ponieważ wymaga 3 pełnych operacji szyfrowania blokowego. Ponieważ minimalna zalecana obecnie długość klucza wynosi 128 bitów, 2-klawiszowy 3DES nie jest już zalecany (ze względu na długość klucza wynoszącą tylko 112 bitów). Te wady doprowadziły do zastąpienia DES/potrójnego DES przez zaawansowany standard szyfrowania, przedstawiony w następnej sekcji.

6.2.5 AES – Zaawansowany standard szyfrowania

W styczniu 1997 roku Narodowy Instytut Standardów i Technologii Stanów Zjednoczonych (NIST) ogłosił, że zorganizuje konkurs na wybór nowego szyfru blokowego —zwanego Advanced Encryption Standard (AES) —który zastąpi DES. Konkurs rozpoczął się od otwartego zaproszenia dla zespołów do przesyłania do oceny kandydatów na szyfry blokowe. W sumie nadesłano 15 różnych algorytmów z całego świata, w tym wkład wielu najlepszych kryptologów i kryptoanalityków. Szyfr kandydujący na każdy zespół był intensywnie analizowany przez członków NIST, opinię publiczną i (zwłaszcza) inne zespoły. Zorganizowano dwa warsztaty, jeden w 1998 r. i jeden w 1999 r., w celu omówienia i przeanalizowania różnych zgłoszeń. Po drugim warsztacie NIST zawiązało grono do 5 „finalistów” i rozpoczęło się druga runda konkursu. Trzecie warsztaty AES odbyły się w kwietniu 2000 r. i zaowocowały dodatkową analizą pięciu finalistów. W październiku 2000 roku NIST ogłosił, że zwycięskim algorytmem został Rijndael (szyfr blokowy zaprojektowany przez belgijskich kryptografów Vincenta Rijmena i Joan Daemen), chociaż NIST przyznał, że każdy z pięciu finalistów dokonałby doskonałego wyboru.

W szczególności u żadnego z 5 finalistów nie stwierdzono żadnych poważnych luk w zabezpieczeniach, a wybór „zwycięzcy” opierał się częściowo na takich cechach, jak wydajność, wydajność sprzętu, elastyczność itp.

Proces wyboru AES był genialny, ponieważ każda grupa, która przesłała algorytm i w związku z tym była zainteresowana przyjęciem jego algorytmu, miała silną motywację do znalezienia ataków na inne zgłoszenia. W ten sposób najlepsi kryptoanalitycy świata skupili swoją uwagę na znalezieniu nawet najmniejszych słabości w zgłoszonych do konkursu szyfrach-kandydatach. Już po kilku latach każdy z kandydatów na algorytm został już poddany intensywnym badaniom, co zwiększyło naszą pewność co do bezpieczeństwa zwycięskiego algorytmu. Oczywiście im dłużej algorytm jest używany i badany bez niego

zostanie złamana, tym bardziej będę dzieś rosła nasza pewność siebie. Obecnie AES jest szeroko stosowany i nie wykryto żadnych znaczących luk w zabezpieczeniach.

Konstrukcja AES. W tej sekcji przedstawiamy strukturę wysokiego szczebla Rijndael/AES. (Technicznie rzecz biorąc, Rijndael i AES to nie to samo, ale różnice nie mają znaczenia dla naszej dyskusji.) Podobnie jak w przypadku DES, nie będę dzielić przedstawiać pełnej specyfikacji i nasz opis nie powinien być używany jako podstawa do wdrożenia. Naszym celem jest jedynie przedstawienie ogólnego pojęcia o działaniu algorytmu.

Szyfr blokowy AES ma 128-bitową długość bloku i może wykorzystywać klucze 128-, 192- lub 256-bitowe. Długość klucza wpływa na harmonogram klucza (tj. podklucz używany w każdej rundzie), a także na liczbę rund, ale nie wpływa na wysokopoziomową strukturę każdej rundy.

W przeciwieństwie do DES, który wykorzystuje strukturę Feistela, AES jest zasadniczo siecią podstawieniowo-permutacyjną. Podczas obliczania algorytmu AES tablica 4 na 4 bajtów zwana stanem jest modyfikowana w serii rund.

Stan jest początkowo ustawiony jako równy wejściu szyfro (należy pamiętać, że dane wejściowe to 128 bitów, czyli dokładnie 16 bajtów). Następnie w każdej rundzie do stanu przeprowadzane są następujące operacje w serii czterech etapów:

Etap 1 – AddRoundKey: W każdej rundzie AES z klucza głównego wyprowadzany jest 128-bitowy podklucz, który jest interpretowany jako tablica bajtów 4 na 4. Tablica stanów jest aktualizowana poprzez XORowanie jej za pomocą tego podklucza.

Etap 2 – Podbajty: Na tym etapie każdy bajt tablicy stanu jest zastąpiony innym bajtem zgodnie z pojedynczą stałą tablicą przeglądową S. Ta tablica podstawień (lub S-box) jest bijekcją przez $\{0, 1\}^8$.

Etap 3 – ShiftRows: W tym kroku bajty w każdym wierszu tablicy stanu są przesuwane w lewo w następujący sposób: pierwszy wiersz tablicy pozostaje nienaruszony, drugi wiersz jest przesuwany o jedno miejsce w lewo, trzeci wiersz jest przesunięty o dwa miejsca w lewo, a czwarty rząd przesunięty o trzy miejsca w lewo. Wszystkie przesunięcia mają charakter cykliczny, tak że np. w drugim wierszu pierwszy bajt staje się czwartym bajtem.

Etap 4 – MixColumns: Na tym etapie do czterech bajtów w każdej kolumnie stosowana jest odwracalna transformacja. (Z technicznego punktu widzenia jest to transformacja liniowa —tj. mnożenie macierzy —po odpowiednim polu.) Transformacja ta ma tę właściwość, że jeśli dwa dane wejściowe różnią się o $b > 0$ bajtami, to zastosowanie transformacji daje dwa wyjścia różniące się co najmniej 5-bajtami.

W ostatniej rundzie MixColumns zostaje zastąpiony przez AddRoundKey. Uniemożliwia to przeciwnikowi proste odwrócenie trzech ostatnich etapów, które nie zależą od klucza.

Postrzegając etapy 3 i 4 łącznie jako etap „mieszania”, widzimy, że każda runda AES ma strukturę sieci podstawień-permutacji:

Podklucz rundy jest najpierw poddawany XOR z wejściem do bieżącej rundy; następnie do „fragmentów” wynikowej wartości stosowana jest mała, odwracalna funkcja; na koniec fragmenty wyniku mieszają się w celu uzyskania dyfuzji. Jedyna różnica polega na tym, że w przeciwnieństwie do naszego poprzedniego opisu sieci podstawieniowo-permutacyjnych, tutaj etap mieszania nie polega na prostym przemieszczaniu bitów, lecz jest przeprowadzany przy użyciu przemieszczania i odwracalnej transformacji liniowej.

(Upraszczając trochę i patrząc na trywialny 3-bitowy przykład, przetasowanie bitów $x = x_1x_2x_3$ może np. odwzorować x na $x = x_2x_1x_3$. Odwracalna transformacja liniowa może odwzorować x na $x_1 \quad x_2x_2 \quad x_3x_1 \quad x_2 \quad x_3$.)

Liczba rund zależy od długości klucza. Do tego używa się dziesięciu rund klucz 128-bitowy, 12 rund dla klucza 192-bitowego i 14 rund dla klucza 256-bitowego.

Bezpieczeństwo AES. Jak wspomnieliśmy, szyfr AES był przedmiotem intensywnej kontroli podczas procesu selekcji i od tego czasu jest nadal badany. Do tej pory nie ma praktycznych ataków kryptoanalytycznych, które byłyby znacznie lepsze niż wyczerpujące wyszukiwanie klucza.

Dochodzimy do wniosku, że na dzień dzisiejszy AES stanowi doskonały wybór dla każdego schematu kryptograficznego, który wymaga (silnej) permutacji pseudolosowej. Jest bezpłatny, ustandaryzowany, wydajny i wysoce bezpieczny.

6.2.6 *Kryptoanaliza różnicowa i liniowa

Szyfry blokowe są stosunkowo skomplikowane i jako takie trudne do analizy. Niemniej jednak nie należy dać się zwieść myśleniu, że skomplikowany szyfr jest trudny do złamania. Wręcz przeciwnie, bardzo trudno jest skonstruować bezpieczny szyfr blokowy i zaskakująco łatwo znaleźć ataki na wiele kogoś konstrukcji (bez względu na to, jak bardzo wydają się skomplikowane). Powinno to służyć jako ostrzeżenie, że osoby niebędące ekspertami nie powinny próbować konstruować nowych szyfrów. Biorąc pod uwagę dostępność potrójnego DES i AES, trudno uzasadnić użycie czegokolwiek innego.

W tej sekcji opisujemy dwa narzędzia, które są obecnie standardową częścią zestawu narzędzi kryptoanalytyka. Naszym celem jest posmakowanie zaawansowanej analizy krypt, a także wzmocnienie idei, że projektowanie bezpiecznego szyfru blokowego wymaga starannego wyboru jego komponentów.

Kryptanaliza różnicowa. Technika ta, która może prowadzić do ataku wybranego tekstu jawnego na szyfr blokowy, została po raz pierwszy zaprezentowana pod koniec lat 80. XX wieku przez Bihama i Shamira, którzy użyli jej do ataku na DES w 1993 r. Podstawową ideą ataku jest zestawienie konkretnych różnic w dane wejściowe, które prowadzą do określonych różnic w wynikach z prawdopodobieństwem więcej niż można by się spodziewać w przypadku losowej permutacji. W szczególności, założmy, że różnica (x, y) występuje w pewnej kluczowanej permutacji F z prawdopodobieństwem p , jeśli dla jednolitych danych wejściowych x_1 i x_2 spełniających $x_1 - x_2 = x$ i równomierny wybór klucza k , prawdopodobieństwo, że $f(x_1) \neq f(x_2)$, jest p . Dla dowolnego ustalonego (x, y) i x_1, x_2 spełniającego $x_1 - x_2 = x$, jeśli wybierzymy funkcję jednorodną $f : \{0, 1\}^n \rightarrow \{0, 1\}$, mamy $\Pr[f(x_1) \neq f(x_2)] = p$. Jednak w słabym szyfrze blokowym może tak być

bę dą różnice, które występują ze znacznie więcej ksyym prawdopodobieństwem. Można to wykorzystać, aby zapewnić pełny atak polegający na odzyskiwaniu klucza, jak teraz pokazujemy dla nazw SPN.

Opisujemy podstawową ideę, a następnie pracujemy nad konkretnym przykładem. Niech F będzie szyfrem blokowym z-bitową długością bloku, która jest okrągłą SPN i niech $F_k(x)$ oznacza pośredni wynik obliczenia $F_k(x)$ po zastosowaniu etapu mieszania kluczy w rundzie r . (Oznacza to, że F wyklucza podstawienie S-boxa i permutację mikowania ostatniej rundy, a także końcowy etap mikowania kluczy). Założymy, że istnieje różniczka ($x_i - y_i$) w F , która występuje z prawdopodobieństwem $p/2^r$. Możliwe jest wykorzystanie tej różnicy o wysokim prawdopodobieństwie do poznania bitów końcowego podklucza mikującego k_{r+1} . Idea wysokiego poziomu jest następująca: niech $\{(x_i - y_i) \mid x_i, y_i \in \{0, 1\}^r\}$ dla wszystkich i . Stosując atak wybranym tekstem tj. z $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ jawnym, uzyskaj $y = F_k(x)$ dla wszystkich i . Teraz dla wszystkich możliwych wartości $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$, oblicz $\tilde{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} + \begin{pmatrix} k_{r+1} \\ 0 \end{pmatrix}$, zakładając, że ciągów bitów $\{1\}$, oblicz $\tilde{y} = \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix} = F_k(\begin{pmatrix} x \\ k_{r+1} \end{pmatrix})$ dla wszystkich i . Kwartylowa wartość końcowego podklucza k_{r+1} wynosi $k_{r+1} = \tilde{y}_1 \oplus \tilde{y}_2$. Odbiera się to poprzez odwrócenie ostatniego etapu mikowania klucza za pomocą k_{r+1} , a następnie odwrócenie permutacji mikowania i S-boxów okrągłego r , które nie zależą od klucza głównego. Gdy $k_{r+1} = k_{r+1}$, oczekujemy, że ułamek $p/2^r$ spełnia $y = \tilde{y}$. Z drugiej strony, gdy $k_{r+1} \neq k_{r+1}$, możemy heurystycznie oczekwać, że tylko 2 ułamek par da tę różnicę. Ustawiając odpowiednio duże L , można określić poprawną wartość końcowego podklucza k_{r+1} .

To działa, ale nie jest zbyt wydajne, ponieważ w każdym kroku wyliczamy ponad 2^{r+1} możliwe wartości. Możemy zrobić lepiej, zgadując porcję k_{r+1} na raz.

Mówiąc bardziej konkretnie, założymy, że S-boxy w F mają 1-bajtową długość wejścia/wyjścia i skupiają się na pierwszym bajcie y , który, jak zakładamy, jest różny od zera. Można sprawdzić, czy różnica utrzymuje się w tym bajcie, zgadując tylko 8 bitów k_{r+1} , a mianowicie 8 bitów, które odpowiadają (po permutacji mieszania round- r) wyjściu pierwszego S-boxa. Zatem, postępując jak powyżej, możemy nauczyć się tych 8 bitów, wyliczając wszystkie możliwe wartości tych bitów i sprawdzając, która wartość z największym prawdopodobieństwem daje żądaną różnicę w pierwszym bajcie. Nieprawidłowe domysły dla tych 8 bitów dają oczekiwanej różnicę w tym bajcie (heurystycznym) prawdopodobieństwem 2^{-8} , ale prawidłowe odgadnięcie da oczekiwanej różnicę z prawdopodobieństwem w przybliżeniu $p/2^8$; dzieje się tak dlatego, że z prawdopodobieństwem $p/2^8$ różnica obowiązuje w całym bloku (a więc w szczególności w przypadku pierwszego bajtu), a gdy tak nie jest, możemy traktować różnicę w pierwszym bajcie jako losową. Należy pamiętać, że do nauczenia się różnych części k_{r+1} mogą być potrzebne różne różnice.

W praktyce przeprowadza się różne optymalizacje, aby poprawić skuteczność powyższego testu, a dokładniej, aby zwiększyć różnicę mięź dzy prawdopodobieństwem, że błędne przypuszczenie da różnicę, a prawdopodobieństwem, że uda się dokonać prawidłowego przypuszczenia. Jedną z optymalizacji jest użycie mechanizmu różnicowego o małej wadze, w którym y ma wiele bajtów zerowych w pozycjach wchodzących do S-boxów w rundzie r . Dowolne pary y_1, y_2 spełniające taką różnicę mają równe wartości wchodzące do wielu S-boxów w rundzie r , co daje w rezultacie wartości wyjściowe y_1, y_2 , które są równe w odpowiednich pozycjach bitów (w zależności od końcowego

permutacja mieszania). Oznacza to, że wykonując opisany wcześniej test, można po prostu odrzucić dowolne pary (pozycje y (ponieważ y_1, y_2), które nie zgadzają się w tych bit-odpowiednie wartości pośrednie (y_1, y_2) nie mogą w żadnym wypadku spełniać różnicy, dla dowolnego wyboru końcowego podklucza). Znaczco poprawia to skuteczność ataku.

Gdy znane jest $kr+1$, atakujący może „odkleić” ostatni etap mikowania klucza, a także etapy permutacji mikowania i podstawienia S-boxa w rundzie r (ponieważ nie zależą one od klucza głównego), a następnie zastosuj ten sam atak — używając innego mechanizmu różnicowego — aby znaleźć podklucz kr w trzeciej rundzie i tak dalej, aż nauczy się wszystkich podkluczy (lub, równoważnie, całego klucza głównego).

Sprawdzony przykład. Pracujemy na przykładzie „zabawki”, ilustrując również, jak można znaleźć dobry mechanizm różnicowy. Używamy czterokrągłego SPN o długości bloku 16 bitów, opartego na pojedynczym S-boxie z 4-bitową długością wejścia/wyjścia. S-box jest zdefiniowany w następujący sposób (tabela pokazuje, w jaki sposób każde 4-bitowe wejście jest mapowane na 4-bitowe wyjście):

Wejście: 0000 0001 0010 0011 0100 0101 0110 0111

Wyjście: 0000 1011 0101 0001 0110 1000 1101 0100

Wejście: 1000 1001 1010 1011 1100 1101 1110 1111

Wyjście: 1111 0111 0010 1100 1001 0011 1110 1010

Permutacja mieszania pokazująca, gdzie przesuwany jest każdy bit dla każdego z nich 16 bitów w bloku wygląda następująco:

W: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Schodzi: 7 2 3 8 12 5 11 9 10 1 14 13 4 6 16 15

x	$S(x)$	$x \oplus 1111$	$S(x \oplus 1111)$	$S(x) \oplus S(x \oplus 1111)$
0000	0000	1111	1010	1010
0001	1011	1110	1110	0101
0010	0101	1101	0011	0110
0011	0001	1100	1001	1000
0100	0110	1011	1100	1010
0101	1000	1010	0010	1010
0110	1101	1001	0111	1010
0111	0100	1000	1111	1011
1000	1111	0111	0100	1011
1001	0111	0110	1101	1010
1010	0010	0101	1000	1010
1011	1100	0100	0110	1010
1100	1001	0011	0001	1000
1101	0011	0010	0101	0110
1110	1110	0001	1011	0101
1111	1010	0000	0000	1010

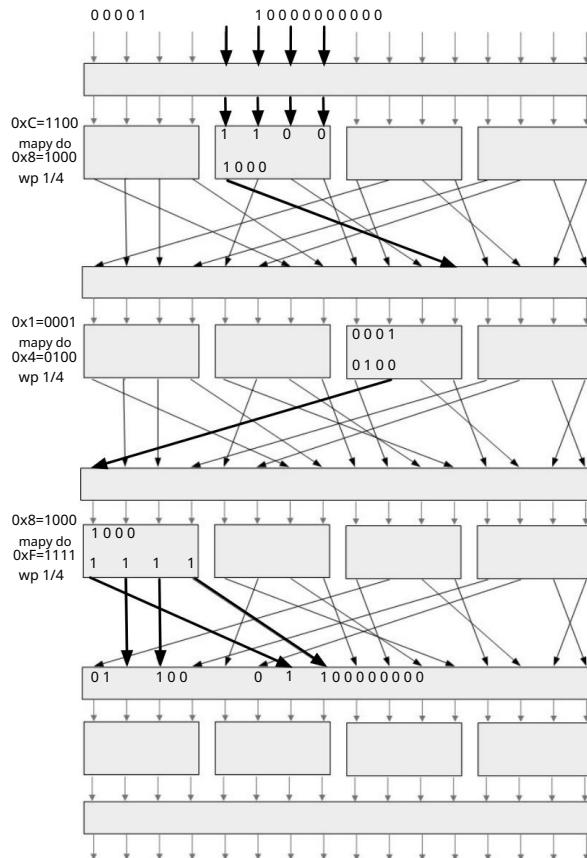
RYSUNEK 6.8: Wpływ różnic wejściowej $x = 1111$ w naszym S-boxie.

Najpierw znajdujemy różnicę w S-boxie. Niech $S(x)$ oznacza wynik funkcji S-box na wejściu x . Rozważmy różnicę $x = 1111$. Wtedy na przykład my mieć $S(0000) = 0000$ $1010 = 1010$ i tak w tym przypadku różnica Wartość 1111 na wejściach prowadzi do różnicy 1010 na wyjściach. Zobaczmy, czy ta zależność występuje często. Mamy $S(0001) = 1011$ i $S(0001 \oplus 1111) = S(1110) = 1110$, a więc taka różnica 1111 na wejściach nie ma znaczenia prowadzi do różnicy na wyjściu wynoszącej 1010. Jednak $S(0100) = 0110$ i $S(0100 \oplus 1111) = S(1011) = 1100$ i tak w tym przypadku różnica 1111 na wejściach daje różnicę 1010 na wyjściach. Na rysunku 6.8 my mają tabelaryczne wyniki dla wszystkich możliwych danych wejściowych. Widzimy to w połowie przypadków a różnica 1111 na wejściach daje różnicę 1010 na wyjściach. Zatem $(1111, 1010)$ jest różniczką w S, która występuje z prawdopodobieństwem 1/2.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Różnica wyjściowa	y
	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	000000000000	
0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	000000000000	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00400022220040	
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20202242200	
3	3	0	2	2	4	0	0	0	0	0	0	0	0	0	0	0	40000000220	
4	4	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	26020002000	
x	5	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	00040004220	
6	6	0	2	0	2	0	0	0	0	0	0	0	0	0	0	0	00002020404	
7	7	0	2	0	0	2	0	0	0	0	0	0	0	0	0	0	42202000200	
8	8	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	00200022224	
9	9	0	2	0	2	2	0	0	0	0	0	0	0	0	0	0	20402200000	
A	A	0	0	0	0	4	0	2	0	0	0	0	0	0	0	0	02420200000	
B	B	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	02002004240	
C	C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00044040004	
D	D	0	4	2	2	0	0	0	0	0	0	0	0	0	0	0	02200000004	
mi		0	2	4	2	4	0	0	0	0	0	0	0	0	0	0	0242400000002020	
F	F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22020820000	

RYSUNEK 6.9: Różnice w naszym S-boxie.

Ten sam proces można przeprowadzić dla wszystkich 24 różnic wejściowych x do obliczyć prawdopodobieństwo każdej różnicy. Mianowicie dla każdej pary (x, y) zestawiamy liczbę 4-bitowych wejść x , dla których $S(x) \oplus S(x \oplus y) = y$. Zrobiliśmy to dla naszego przykładu S-box na rysunku 6.9. (Dla zwięzłości my reprezentują (x, y) w zapisie szesnastkowym.) Tablicę należy czytać jako następ pującą: wpis (i, j) zlicza, ile wejść z różnicą mapuje na wyjście z różnicą j . Zauważ na przykład, że istnieje 8 wejść różniczących się $0xF = 1111$, które mapują na wyjście $0xA = 1010$, jak pokazaliśmy powyżej. To jest różnica o najwyższym prawdopodobieństwie (oprócz różniczki trywialnej $(0, 0)$). Ale są też inne interesujące różnice: różnica wejściowa $0x4 = 0100$ odwzorowuje różnicę wyjściową $0x6 = 0110$ z prawdopodobieństwem $6/16 = 3/8$, oraz istnieje kilka różnic z prawdopodobieństwem $4/16 = 1/4$.



RYSUNEK 6.10: Śledzenie różnic w czterorundowej nazwie SPN, która używa permutacji S-box i mieszanie podana w tekście.

Teraz rozszerzamy to, aby znaleźć dobrą różnicę dla pierwszych trzech rund SPN. Rozważ ocenę SPN na dwóch wejściach, które mają różnicę 0000 1100 0000 0000 i śledzenie różnicy mieć przy wartością pośrednią wartości na każdym etapie tej oceny. (Patrz rysunek 6.10, który pokazuje cztery pełne rundy SPN plus ostatni etap mikowania kluczy. Dla przejrzystości rysunek pomija permutację mieszania w 4. rundzie; ta permutacja mieszania powoduje po prostu przetasowanie bitów mechanizmu różnicowego i można to łatwo zrobić należącą pod uwagę w ataku.) Etap mieszania kluczy w pierwszej rundzie nie ma wpływu na różnicę, a więc wejścia do drugiego S-boxa w pierwsza runda ma różnicę 1100. Z rysunku 6.9 widzimy, że różnica wynosi 1100 0xC = 1100 na wejściach S-box daje różnicę 0x8 = 1000 wyjścia skrzynki S z prawdopodobieństwem 1/4. Zatem z prawdopodobieństwem 1/4 różnica na wyjściu drugiego S-boxa po rundzie 1 jest pojedynczym bitem

przesunięty przez permutację mieszania z pozycji 5. na pozycję 12.

(Wejścia do pozostałych S-boxów są równe, więc ich wyjścia są równe, a różnica wyjścia wynosi 0000.) Zakładając, że tak jest, różnica wejściowa do trzeciego S-boxa w drugiej rundzie wynosi 0x1 = 0001 (po raz kolejny etap mieszania kluczy w drugiej rundzie nie ma wpływu na mechanizm różnicowy); korzystając z rysunku 6.9 mamy, że z prawdopodobieństwem 1/4 różnica wejściowa z tego S-boxa wynosi 0x4 = 0100. Zatem po raz kolejny istnieje tylko jeden bit wyjściowy, który jest inny i jest on przesuwany z 10. pozycji na pierwszą położenie poprzez permutację mieszania. Na koniec, ponownie przeglądając rysunek 6.9, widzimy, że różnica wejściowa wynosząca 0x8 = 1000 w stosunku do S-boxa skutkuje różnicą wyjściową 0xF = 1111 z prawdopodobieństwem 1/4. Bitы w pozycjach 1, 2, 3 i 4 są następujące przesuwane w wyniku permutacji miksucej do pozycji 7, 2, 3 i 8.

Ogólnie rzecz biorąc, widzimy, że różnica wejściowa wynosząca $x = 0000\ 1100\ 0000\ 0000$ daje różnicę wyjściową $y = 0110\ 0011\ 0000\ 0000$ po trzech rundach za pomocą (Mnożymy prawdopodobieństwo $\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{4} = \frac{1}{64}$. prawdopodobieństwa, ponieważ przynajmniej heurystycznie zakładamy niezależność każdej rundy.) W przypadku funkcji losowej prawdopodobieństwo wystąpienia dowolnej różnicy wynosi po prostu $2-16 = 1/65536$. Zatem znaleziona przez nas różnica występuje z prawdopodobieństwem znacznie więcej niż oczekiwane w przypadku funkcji losowej. Zauważ również, że znaleźliśmy mechanizm różnicowy o niskiej masie.

Mogliśmy użyć tej różnicy, aby znaleźć pierwsze 8 bitów końcowego podklucza k5. Jak omówiono wcześniej, zaczynamy od pozwolenia (x będące $\frac{1}{L}$ obiektem L) par losowych danych wejściowych z różniczką x . Stosując atak wybranym tekstem jawnym, wówczas dla wszystkich uzyskać wartości $y = F_k(x - 1)$ i $y' = F_k(x - 2)$. Teraz, dla wszystkich możliwych wartości y dla początkowych 8 bitów k5, obliczamy początkowe 8 bitów y' wartości $\frac{1}{1}, y'^2$, pośrednich po etapie mieszania kluczy w czwartej rundzie. (Moglibyśmy to zrobić, ponieważ wystarczy odwrócić dwa skrajne na lewo S-boxy czwartej rundy, aby otrzymać te 8 bitów.) Kiedy odgadniemy poprawną wartość dla początkowych 8 bitów k5, spodziewamy się, że 8-bitowa różnica 0110 0011, która wystąpi z prawdopodobieństwem co najmniej 1/64. Z heurystycznego punktu widzenia nieprawidłowe przypuszczenie daje oczekiwana różnicę tylko z prawdopodobieństwem $2-8 = 1/256$. Ustawiając odpowiednio duże L, możemy (z dużym prawdopodobieństwem) zidentyfikować poprawną wartość.

Ataki różnicowe w praktyce. Kryptanaliza różnicowa jest bardzo potężna i została wykorzystana do atakowania prawdziwych szyfrów. Wybitnym przykładem jest FEAL-8, który został zaproponowany jako alternatywa dla DES w 1987 r. Stwierdzono różnicowy atak na FEAL-8, który wymaga zaledwie 1000 wybranych tekstów jawnych. W 1991 r. znalezienie całego klucza za pomocą tego ataku zajęło mniej niż 2 minuty. Obecnie każdy proponowany szyfr jest testowany pod kątem odporności na kryptoanalizę różnicową.

Atak różnicowy był pierwszym atakiem na DES, który wymagał mniej czasu niż prostego wyszukiwanie metodą brute-force. Choć jest to interesujący wynik teoretyczny, atak nie budzi wiele恐慌 obaw w praktyce, ponieważ wymaga 247 wybranych tekstów jawnych. To

⁷Jest to dolna granica prawdopodobieństwa różnicę, ponieważ mogą wystąpić inne różnice w wartościach pośrednich, które skutkują taką samą różnicą w wynikach.

atakującemu bardzo trudno jest uzyskać tak wiele wybranych par tekstu jawnego/zaszyfrowanego w więcej ksmości rzeczywistych aplikacji. Co ciekawe, niewielkie modyfikacje S-boxów DES sprawiają, że szyfr jest znacznie bardziej podatny na ataki różnicowe. Osobiste zeznania projektantów DES (po odkryciu ataków różnicowych w świecie zewnątrz) potwierdziły, że S-boxy DES zostały zaprojektowane specjalnie w celu udaremnenia ataków różnicowych.

Kryptanaliza liniowa. Kryptanalizę liniową opracował Matsui na początku lat 90. Opiszymy technikę tylko na wysokim poziomie. Podstawową ideą jest rozważenie liniowych zależności między danymi wejściowymi i wyjściowymi, które zachodzą w więcej ksmo prawdopodobieństwem, niż można by się spodziewać w przypadku funkcji losowej. Bardziej szczegółowo, powiedzmy, że bity ustawiają i_1, \dots, i_n i mają odchylenie liniowe ϵ if, dla jednorodnych x i k oraz $y = F_k(x)$, to zachodzi

$$\Pr[x_{i1} + \dots + x_{in} - y_i] = \frac{1}{2} = \epsilon,$$

gdzie x_i, y_i oznaczają i -te bity x i y . W przypadku funkcji losowej i dowolnego ustalonego zestawu pozycji bitów oczekujemy, że odchylenie będzie bliskie 0. Matsui pokazał, jak zastosować wystarczająco duże odchylenie w szyfrze F, aby znaleźć tajny klucz. Oprócz zapewnienia innej metody atakowania szyfrów, ważną cechą tego ataku jest to, że nie wymaga on wybranych tekstów jawnych, ale wystarczą znane teksty jawnie.

Jest to bardzo istotne, ponieważ zaszyfrowany plik może dostarczyć ogromną ilość znanego tekstu jawnego, podczas gdy zebranie szyfrów wybranych tekstów jawnych jest znacznie bardziej trudne. Matsui pokazał, że DES można złamać za pomocą zaledwie 243 znanych par tekstu jawnego/zaszyfrowanego.

Wpływ na konstrukcję szyfru blokowego. Nowoczesne szyfry blokowe są projektowane i oceniane częściowo w oparciu o ich odporność na różnicową i liniową analizę krypt. Konstruując szyfr blokowy, projektanci wybierają S-boxy i inne komponenty, aby zminimalizować prawdopodobieństwa różnicowe i odchylenia liniowe. Zauważamy, że nie jest możliwe wyeliminowanie wszystkich różnic o wysokim prawdopodobieństwie w S-boxie: w każdym S-boxie będzie pewna różnica, która występuje częściej niż inne. Mimo to odchylenia te można zminimalizować. Co więcej, zwiększenie liczby rund (i ostrożny wybór permutacji mieszania) może zarówno zmniejszyć prawdopodobieństwa różnicowe, jak i utrudnić kryptoanalystom znalezienie jakichkolwiek różnic do wykorzystania.

6.3 Funkcje mieszające

Przypomnijmy sobie z rozdziału 5, że głównym wymogiem bezpieczeństwa funkcji skrótu H jest odporność na kolizje; to znaczy, powinno być trudno znaleźć kolizję lub różne dane wejściowe x, x takie, że $H(x) = H(x)$. (Porzucamy wzmiankę

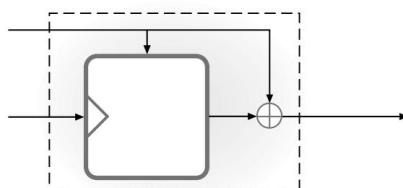
dowolnego klucza, ponieważ funkcje mieszające w świecie rzeczywistym zazwyczaj nie są kluczowane.) Jeśli funkcja mieszająca ma długość wyjściową -bitową, to najlepsze, na co możemy mieć nadzieję, to to, że znalezienie kolizji przy użyciu znacznie mniej niż $2^{2/2}$ powinno być niemożliwe. wywołania H. (Patrz sekcja 5.4.1.) Chcielibyśmy także, aby funkcja mieszająca osiągnęła (drugą) odporność na ataki poprzedzające ataki trwające w czasie znacznie krótszym niż 2, chociaż nie bierzemy pod uwagę takich ataków w naszej tutaj dyskusji.

Funkcje skrótu są zazwyczaj konstruowane w dwóch etapach. Najpierw projektowana jest funkcja kompresji (tj. funkcja mieszająca o stałej długości) h; nastepnie stosuje się pewien mechanizm do rozszerzenia h tak, aby obsłużyć dowolne długości wejściowe. W części 5.2 pokazaliśmy już jedno podejście – transformację Merkle'a – Damgarda – dla drugiego kroku. Tutaj badamy technikę projektowania podstawowej funkcji kompresji. Omówimy także niektóre funkcje skrótu stosowane w praktyce. Teoretyczna konstrukcja funkcji kompresji oparta na założeniach z teorii liczb jest podana w rozdziale 8.4.2.

6.3.1 Funkcje skrótu z szyfrów blokowych

Być może zaskakujące jest to, że możliwe jest zbudowanie odpornej na kolizje funkcji kompresji z szyfrem blokowego, który spełnia pewne dodatkowe właściwości. Można to zrobić na kilka sposobów; jedna z najczęstszych to konstrukcja Daviesa – Meyera. Niech F będzie szyfrem blokowym o n-bitowej długości klucza i -bitowej długości bloku. Możemy wtedy zdefiniować funkcję kompresji $h : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$

$$\stackrel{\text{def}}{=} F_k(x) \quad x. \quad (\text{Patrz rysunek 6.11.})$$



RYSUNEK 6.11: Konstrukcja Daviesa – Meyera.

Nie wiemy, jak udowodnić odporność na kolizje powstałej funkcji kompresji opierając się wyłącznie na założeniu, że F jest silną permutacją pseudolosową, a w rzeczywistości istnieją podstawy, aby sądzić, że taki dowód nie jest możliwy. Możemy jednak udowodnić odporność na kolizje, jeśli chcemy modelować F jako idealny szyfr. Model idealnego szyfru jest wzmacnieniem modelu losowej wyroczni (patrz sekcja 5.5), w którym zakładamy, że wszystkie strony mają dostęp p do wyroczni dla losowej permutacji z kluczem $F : \{0, 1\}^n \times \{0, 1\}^{n+1} \rightarrow \{0, 1\}^{n+1}$ oraz jego odwrotności $F^{-1}(k, F(k, x)) = x$ dla wszystkich k, x .
(tj. tak, że $F^{-1} \circ F = \text{id}$)

Można o tym pomyśleć inaczej: każdy klucz $k \in \{0, 1\}^n$ określa niezależną, jednolitą permutację $F(k, \cdot)$ na ciągach -bitowych. Podobnie jak w modelu losowej wyroczni, jedynym sposobem obliczenia F (lub F^{-1}) polega na jawnym zapytaniu wyroczni z powrotem $F(k, x)$ (lub $F^{-1}(k, x)$).

Analizowanie konstrukcji w modelu szyfru idealnego wiąże się ze wszystkimi zaletami i wadami pracy w modelu losowej wyroczni, jak omówiono szczegółowo w sekcji 5.5. Dodajemy tutaj tylko, że model szyfru idealnego implikuje brak ataków na klucz powiązany w tym sensie, że (jak właśnie powiedzieliśmy) permutacje $F(k, \cdot)$ i $F(k, \cdot)$ muszą zachowywać się niezależnie, nawet jeśli na przykład k i k różnią się tylko jednym bitem. Ponadto nie może być „słabych kluczy” k (powiedzmy klucza z wartością 0), w przypadku których $F(k, \cdot)$ można łatwo odróżnić od losowego. Oznacza to również, że $F(k, \cdot)$ powinno „zachowywać się losowo”, nawet gdy k jest znane. W przypadku dowolnego szyfru F w świecie rzeczywistym właściwości te niekoniecznie obowiązują (i nie są nawet dobrze zdefiniowane), nawet jeśli F jest silną permutacją pseudolosową, a czytelnik może zauważać, że nie omawialiśmy tych właściwości w żadnej z naszych analiz rzeczywistych konstrukcji szyfrów blokowych. (W rzeczywistości DES i potrójny DES nie spełniają tych właściwości.) Każdy szyfr blokowy używany do utworzenia instancji idealnego szyfru musi być oceniany pod kątem tych bardziej rygorystycznych wymagań.

Dowodzimy następujące twierdzenie w konkretnym kontekście, ale dowód można łatwo dostosować również do układu asymptotycznego.

Twierdzenie 6.5 Jeśli F jest modelowane jako szyfr idealny, wówczas konstrukcja Daviesa-Meyera daje funkcję kompresji odporną na kolizje. Konkretnie, każdy atakujący wykonujący zapytania $q < 2/2$ do swoich wyroczni o idealnym szifrowaniu znajduje kolizję z prawdopodobieństwem co najwyżej $q/2$.

DOWÓD Dla jasności rozpatrujemy tutaj probabilistyczny eksperyment, w którym F jest dobierane losowo (dokładniej dla każdego $k \in \{0, 1\}^n$ funkcja $F(k, \cdot) : \{0, 1\}^n \rightarrow \{0, 1\}$ jest wybierany jednolicie z zestawu Perm permutacji na ciągach bitowych), a następnie atakujący otrzymuje dostęp p do Oracle. Atakujący następnie próbuje $F \circ F^{-1}$ znaleźć kolidującą parę $(k, x), (k, x)$, czyli dla której $F(k, x) = F(k, x) = x$. Na atakującego nie nakłada się żadnych ograniczeń obliczeniowych poza ograniczeniem liczby wysyłanych przez niego zapytań Oracle. Zakładamy, że jeśli atakujący wygeneruje kolidującą parę $(k, x), (k, x)$, to wcześniej wykonał zapytania Oracle niezbędne do obliczenia wartości $F(k, x)$ i $F(k, x)$. Zakładamy również, że atakujący nigdy nie wykonuje tego samego zapytania częściej (k, y) , gdy dowie się, (i odwrotnie). Wszystkie te założenia 1) że $y = F(k, x)$ niż raz i nigdy nie zadaje pytania F pozostają bez utraty ogólności.

Rozważmy i-te zapytanie, które atakujący kieruje do swoich wyroczni. Zapytanie (k_i, x_i) do F ujawnia tylko wartość skrótu w $\stackrel{\text{def}}{=} h(k_i, x_i) = F(k_i, x_i) = x_i$; podobnie (k_i, y_i) zapytaniu do F^{-1} dając wynik $x_i = F^{-1}(k_i, y_i)$ daje tylko wartość skrótu $\stackrel{\text{def}}{=} h(k_i, x_i) = y_i$ – faktycznie, $\stackrel{\text{def}}{=} h(k_i, y_i) = F(k_i, x_i) = x_i$. Atakujący nie dochodzi do kolizji chyba że $h_i = h_j$ dla niektórych $i = j$.

Ustal i, j za pomocą $i > j$ i rozważ prawdopodobieństwo, że $h_i = h_j$. W momencie i-tego zapytania wartość h_j jest stała. Kolizja pomiędzy h_i i h_j występuje w i-tym zapytaniu tylko wtedy, gdy atakujący zada zapytanie (k_i, x_i) do F i uzyska wynik $F(k_i, x_i) = h_j = x_i$ lub zapyta (k_i, y_i) do F^{-1} i uzyska

wynik $f^1(k_i, y_i) = h_j \quad y_i$. Każde zdarzenie zachodzi z prawdopodobieństwem co najwyżej $1/2^{q-1}$, ponieważ na przykład $F(k_i, x_i)$ jest jednorodne w czasie $\{0, 1\}$, z tym wyjątkiem, że nie może być równe żadnej wartości $F(k_i, x)$ już zdefiniowane przez (co najwyżej) $i-1$ poprzednie zapytania Oracle przy użyciu klucza k atakującego. Ponieważ $i < q < 2^{q-1}$, prawdopodobieństwo, że $h_i = h_j$ wynosi co najwyżej $2/2^q$.

Biorąc związek związany ze wszystkimi $q < q/2^q$ różnych par i, j daje wynik określonymi w twierdzeniu. ■

Daviesa-Meyera i DES. Jak wspomnieliśmy powyżej, należy zachować ostrożność podczas tworzenia konstrukcji Daviesa-Meyera za pomocą dowolnego szyfru blokowego, ponieważ szyfr musi spełniać dodatkowe właściwości (poza silną permutacją pseudolosową), aby uzyskana konstrukcja była bezpieczna. W ćwiczeniu 6.21 sprawdzimy, co idzie nie tak, gdy w konstrukcji Daviesa-Meyera stosuje się DES.

Powinno to służyć jako ostrzeżenie, że dowód bezpieczeństwa konstrukcji Daviesa-Meyera w modelu idealnego szyfru niekoniecznie przekłada się na rzeczywiste bezpieczeństwo, gdy zostanie utworzony za pomocą prawdziwego szyfru. Niemniej jednak, jak opiszymy poniżej, ten paradymat został wykorzystany do skonstruowania praktycznych funkcji skrótu, które oparły się atakowi (ale szczególnie wtedy, gdy szyfr blokowy w środku konstrukcji został zaprojektowany specjalnie do tego celu).

Podsumowując, konstrukcja Daviesa-Meyera jest użytecznym paradymatem do konstruowania funkcji kompresji odpornych na kolizje. Nie należy go jednak stosować do szyfrów blokowych przeznaczonych do szyfrowania, takich jak DES i AES.

6.3.2 MD5

MD5 to funkcja skrótu o długości wyjściowej 128 bitów. Został zaprojektowany w 1991 roku i przez pewien czas uchodził za odporny na kolizje. W ciągu kilku lat w MD5 zaczęto znajdować różne słabe punkty, ale nie wydawało się, aby prowadziły one do łatwego sposobu znajdowania kolizji. Szokującym jest to, że w 2004 roku zespół chińskich kryptoanalytyków przedstawił nową metodę wyszukiwania kolizji w MD5; z łatwością byli w stanie przekonać innych, że ich podejście jest prawidłowe, demonstrując wyraźną kolizję! Od tego czasu atak został udoskonalony i dziś na komputerze stacjonarnym kolizje można znaleźć w niecałą minutę. Ponadto ataki zostały rozszerzone w taki sposób, że można znaleźć nawet „kontrolowane kolizje” (np. dwa pliki postscriptowe generujące dowolną wyświetlaną treść).

Ze względu na te ataki MD5 nie powinien być używany wszędzie tam, gdzie potrzebne jest bezpieczeństwo kryptograficzne. Wspominamy o MD5 tylko dlatego, że nadal można go znaleźć w starszym kodzie.

6.3.3 SHA-0, SHA-1 i SHA-2

Algorytm bezpiecznego skrótu (SHA) odnosi się do szeregu kryptograficznych funkcji skrótu standaryzowanych przez NIST. Być może najbardziej znanym z nich jest SHA-1, który został wprowadzony w 1995 roku. Algorytm ten ma 160-bitową długość wyjściową i zastąpił poprzednika o nazwie SHA-0, który został wycofany z powodu nieokreślonych wad wykrytych w tym algorytmie.

W chwili pisania tego tekstu w SHA-1 nie znaleziono jeszcze wyraźnej kolizji. Jednak analiza teoretyczna przeprowadzona w ciągu ostatnich kilku lat wskazuje, że kolizje w SHA-1 można znaleźć przy użyciu znacznie mniejszej liczby niż 280 ocen funkcji skrótu, które byłyby konieczne w przypadku ataku urodzinowego, i przypuszcza się, że kolizja zostanie wykryta wkrótce. Dlatego zaleca się migrację do SHA-2, który obecnie nie wydaje się mieć takich samych słabych stron. SHA-2 składa się z dwóch powiązanych funkcji: SHA-256 i SHA-512, o długości wyjściowej odpowiednio 256 i 512 bitów.

Wszystkie funkcje skrótu w rodzinie SHA są zbudowane przy użyciu tego samego podstawowego projektu, który obejmuje komponenty, które już widzieliśmy: funkcję kompresji definiującą się najpierw poprzez zastosowanie konstrukcji Daviesa-Meyera do szyfru blokowego, a następnie rozszerza ją, aby obsługiwała wprowadzanie dowolnej długości przy użyciu transformaty Merkle'a – Damgård'a. Interesującą rzeczą jest to, że szyfr blokowy w każdym przypadku został zaproektowany specjalnie do budowania funkcji kompresji. W rzeczywistości tylko z mocą wstępna wyizolowano podstawowe komponenty funkcji kompresji i przeanalizowano je jako szyfry blokowe SHACAL-1 (dla SHA-1) i SHACAL-2 (dla SHA-2). Szyfry te same w sobie są intrygujące, ponieważ mają duże długości bloków (odpowiednio 160 i 256 bitów) i 512-bitowe klucze.

6.3.4 SHA-3 (Keccak)

W następstwie ataku kolizyjnego na MD5 i teoretycznych słabości wykrytych w SHA-1, NIST ogłosił pod koniec 2007 roku publiczny konkurs na zaprojektowanie nowej kryptograficznej funkcji skrótu, która będzie nazywana SHA-3. Przesłane algorytmy musiały obsługiwać co najmniej 256- i 512-bitowe długości wyjściowe. Podobnie jak w przypadku konkursu AES sprzed mniej więcej 10 lat, konkurs miał charakter całkowicie otwarty i przejrzysty; każdy mógł zgłosić algorytm do rozpatrzenia, a społeczeństwo zostało zaproszone do przedstawienia swojej opinii na temat któregokolwiek z kandydatów. W grudniu 2008 r. zawarto żonę listę 51 kandydatów do pierwszej tury do 14, a w 2010 r. do pięciu finalistów. Pozostałowi kandydaci byli przez następne dwa lata poddawani intensywnej analizie ze strony społeczności kryptograficznej. W październiku 2012 roku NIST ogłosił wybór Keccaka na zwycięzcę konkursu. W chwili pisania tego tekstu algorytm ten przechodzi standaryzację jako następca nowej generacji SHA-2.

Keccak jest niezwykły pod kilkoma względami. (Co ciekawe, jednym z powodów wyboru Keccaka jest to, że jego struktura bardzo różni się od struktury SHA-1 i SHA-2.) W swej istocie opiera się na niekluczowanej permutacji f z dużą długością bloku wynoszącą 1600 bitów; różni się to radykalnie od np. konstrukcji Daviesa-Meyera, która opiera się na permutacji z kluczem. Co więcej, Keccak nie używa transformacji Merkle'a – Damgård'a do obsługi dowolnych długości wejściowych. Zamiast tego wykorzystuje nowsze podejście zwane konstrukcją gąbkową. Keccaka – i szerzej konstrukcję gąbki – można analizować w modelu losowej permutacji, w którym postulujemy, że strony

mieć dostęp p do wyroczni dla losowej permutacji $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ (i ewentualnie jej odwrotność). Jest to słabsze niż model szyfru idealnego; w istocie możemy łatwo uzyskać losową permutację w modelu idealnego szyfru, po prostu ustalając klucz do szyfru na dowolną stałą wartość.

Fascynujące bę die obserwowanie ewolucji nowego standardu skrótu i obserwowanie tego jak szybko programiści dostosowują się z SHA-1/SHA-2 do nowszego SHA-3.

Referencje i dodatkowe lektury

Dodatkowe informacje na temat LFSR i szyfrów strumieniowych można znaleźć w Handbook of Applied Cryptography [120] lub w nowszym tekście Paara i Pelzla [135]. Dalsze szczegóły dotyczące eSTREAM, a także szczegółową specyfikację Trivium można znaleźć na stronie <http://www.ecrypt.eu.org/stream>.

Zobacz pracę AlFardana i in. [9] na potrzeby niedawnego badania ataków na RC4.

Paradygmat zamieszania-dyfuzji i sieci podstawień-permutacji wprowadzili Shannon [154] i Feistel [64]. Więcej informacji na temat projektowania SPN można znaleźć w pracy Heysa [90]. Znane są lepsze ogólne ataki na trzyrundowe SPN niż te, które tutaj pokazaliśmy [31]. Miles i Vi-ola [126] podają teoretyczną analizę SPN.

Sieci Feistela zostały po raz pierwszy opisane w [64]. Teoretyczna analiza Feistela sieci podali Luby i Rackoff [116]; zobacz rozdział 7.

Więcej szczegółów na temat DES, AES i ogólnie konstrukcji szyfrów blokowych można znaleźć w tekście Knudsena i Robshawa [106]. Atak typu „meet-in-the-middle” na podwójne szyfrowanie jest dziełem Diffiego i Hellmana [59]. Atak na potrójne szyfrowanie dwukluczowe wspomniany w tekście (i omówiony w ćwiczeniu 6.13) został przeprowadzony przez Merkle i Hellmana [124]. Teoretyczną analizę bezpieczeństwa podwójnego i potrójnego szyfrowania można znaleźć w [6, 24].

DESX to kolejna technika zwiększenia efektywnej długości klucza DES. Kończek klucza DES. ko $\in \{0, 1\}^{64}$ i k. Tajny klucz składa się z wartości k_1, k_2, \dots, k_m , gdzie $m = 56$, oraz definiuje szyfr

$$\text{DESX}_{k_1, k_2, \dots, k_m}(x) \stackrel{\text{def}}{=} \text{ko} \circ \text{BIURKO}(x \oplus k_1).$$

Metodologię tę po raz pierwszy zbadali Even i Mansour [63] w nieco innym kontekście. Jego zastosowanie do DES zaproponował w niepublikowanej pracy Rivest, a jego bezpieczeństwo analizowali później Kilian i Rogaway [105, 149].

Kryptanalizę różnicową wprowadzili Biham i Shamir [29], a jej zastosowanie do DES opisano w książce tych autorów [30]. Copper-smith [45] opisuje zasady projektowania S-boxów DES w świetle publicznego odkrycia kryptoanalizy różnicowej. Kryptanalizę liniową odkrył Matsui [118], który pokazuje tam jej zastosowanie do DES. Aby uzyskać więcej informacji na temat tych zaawansowanych technik kryptoanalitycznych, odsyłamy czytelnika do samouczka

na temat kryptoanalizy różnicowej i liniowej autorstwa Heysa [91] lub do wspomnianej już książki Knudsena i Robshawa [106].

Więcej informacji na temat MD5 i SHA-1 można znaleźć w [120]. Należy jednak pamiętać, że ich leczenie poprzedza ataki Wanga i in. [175, 174]. Konstrukcje funkcji kompresji z szyfrów blokowych analizowane są w [143, 33].

Budowę gąbek opisali i przeanalizowali Bertoni i in. [28]. Dodatkowe szczegóły dotyczące konkursu SHA-3 można znaleźć na stronie internetowej NIST pod adresem <http://csrc.nist.gov/groups/ST/hash/sha-3/index.html>.

Ćwiczenia

6.1 Założmy LFSR stopnia 6 przy $c_0 = c_5 = 1$ i $c_1 = c_2 = c_3 = c_4 = 0$.

- (a) Jakie jest pierwsze 10 bitów wyprowadzanych przez ten LFSR, jeśli zaczyna się od stanu początkowego $(1, 1, 1, 1, 1, 1)$? (b) Czy jest to maksymalna długość LFSR?

6.2 W tym pytaniu rozważamy generator kombinacji nieliniowych, w którym mamy stopień-n LFSR, ale wyjściem w każdym kroku czasowym nie jest s_0 , ale zamiast tego $g(s_0, \dots, s_{n-1})$ dla pewnej funkcji nieliniowej g . Założymy, że współczynniki sprzężenia zwrotnego LFSR są znane, ale jego stan początkowy nie jest. Pokaż, że każdy z poniższych wyborów g nie daje dobrego generatora pseudolosowego:

- (a) $g(s_0, \dots, s_{n-1}) = s_0 \oplus s_1$. (b)
 $g(s_0, \dots, s_{n-1}) = (s_0 \oplus s_1) \oplus s_2$.

6.3 Niech F będzie szyfrem blokowym z n -bitową długością klucza i długością bloku. Założymy, że istnieje atak polegający na odzyskaniu klucza na F , który kończy się sukcesem z prawdopodobieństwem 1 przy użyciu n wybranych tekstów jawnych i minimalnym wysiłku obliczeniowym. Udowodnij formalnie, że F nie może być permutacją pseudolosową.

6.4 W naszym ataku na jednorundową nazwę SPN wzięliśmy pod uwagę długość bloku wynoszącą 64 bity i 16 S-boxów, z których każdy przyjmuje 4-bitowe wejście. Powtórz analizę dla przypadku 8 S-boxów, każdy z 8-bitowym wejściem. Jaka jest obecnie złożoność ataku? Powtórz analizę ponownie, stosując 128-bitową długość bloku i 16 S-boxów, z których każdy przyjmuje 8-bitowe wejście.

6.5 Rozważ zmodyfikowany SPN, w którym zamiast wykonywać etapy mieszania kluczy, podstawienia i permutacji w przeciwniej kolejności dla r (pełnych) rund, szyfr zamiast tego najpierw stosuje w rundach mieszania kluczy, następnie wykonuje rundy podstawienia i w końcu stosuje permutacje mieszania r . Przeanalizuj bezpieczeństwo tej konstrukcji.

6.6 W tym pytaniu zakładamy dwuokrągłą nazwę SPN z 64-bitową długością bloku.

- (a) Założmy, że w każdej rundzie używane są niezależne 64-bitowe podklucze, zatem klucz główny ma długość 192 bitów. Pokaż atak polegający na odzyskiwaniu klucza, wykorzystując czas znacznie krótszy niż 2192 .
- (b) Założmy, że pierwszy i trzeci podklucz są równe, a drugi podklucz jest niezależny, zatem klucz główny ma długość 128 bitów. Pokaż atak polegający na odzyskiwaniu klucza, wykorzystując czas znacznie krótszy niż 2128 .

6.7 Jaki jest wynik r-okrągłej sieci Feistela, gdy wejściem jest (L_0, R_0) w każdym z dwóch poniższych przypadków:

- (a) Każda funkcja okrągła wyprowadza wszystkie zera, niezależnie od wejścia.
- (b) Każda funkcja okrągła jest funkcją tożsamości.

6.8 Niech Feistelf1,f2 (\cdot) oznaczają dwuokrągłą sieć Feistela wykorzystującą funkcje f1 i f2 (w tej kolejności). Pokaż, że jeśli $\text{Feistelf1,f2} (L_0, R_0) = (L_2, R_2)$, to $\text{Feistelf2,f1} (R_2, L_2) = (R_0, L_0)$.

6.9 W tym ćwiczeniu opieraj się na opisie DES podanym w tym rozdziale, ale wykorzystaj fakt, że podczas faktycznej konstrukcji DES dwie połowy wyjścia ostatniej rundy sieci Feistel są zamienione miejscami. Oznacza to, że jeśli wyjściem ostatniej rundy sieci Feistela jest (L_{16}, R_{16}) , to wyjściem DES jest (R_{16}, L_{16}) .

- (a) Pokaż, że jedyną różnicą pomiędzy obliczeniem DES k i DES-1 jest kolejność użycia podkluczy.
(Oprzyj się na poprzednim ćwiczeniu.)
- (b) Pokaż, że dla $k = 056$ zachodzi $\text{DES}_k(\text{DES}_k(x)) = x$.
Wskazówka: rozważ podklucze wygenerowane na podstawie tego klucza.
- (c) Znajdź trzy inne klucze DES o tej samej właściwości. Klucze te nazywane są słabymi kluczami DES. (Uwaga: znalezione klucze będą dążyć różnić od rzeczywistych słabych kluczy DES ze względu na różnice w naszej prezentacji.)
- (d) Czy te 4 słabe klucze stanowią poważną lukę w użyciu potrójnego DES jako permutacji pseudolosowej? Wyjaśnić.

6.10 Pokaż, że DES ma tę właściwość, że $\text{DES}_k(x) = \text{DES}_{\bar{k}}(\bar{x})$ dla każdego klucza k i wejścia x (gdzie $\bar{\cdot}$ z oznacza bitowe uzupełnienie z).
(Nazywa się to właściwością DES komplementarnością.) Czy stanowi to poważną lukaną w korzystaniu z potrójnego DES jako permutacji pseudolosowej? Wyjaśnić.

6.11 Opisz ataki na następujące modyfikacje DES:

- (a) Każdy podklucz ma długość 32 bitów, a funkcja okrągła po prostu wykonuje XOR podklucza z wejściem do zaokrąglenia (tj. $\hat{f}(k, R) = k_i \oplus R$).
W przypadku tego pytania harmonogram kluczy jest nieistotny i można traktować podklucze k_i jako klucze niezależne.
- (b) Zamiast używać różnych podkluczy w każdej rundzie, w każdej rundzie używany jest ten sam 48-bitowy podklucz. Pokaż, jak odróżnić szyfr od losowej permutacji 48 2 raz.

Wskazówka: pomocne mogą być ćwiczenia 6.8 i 6.9...

6.12 (To ćwiczenie opiera się na ćwiczeniu 6.9.) Naszym celem jest pokazanie, że dla dowolnego słabego klucza k DES łatwo jest znaleźć dane wejściowe x takie, że $DES_k(x) = x$.

- (a) Założmy, że oceniamy DES_k na wejściu (L_0, R_0) , a wyjściem po 8 rundach sieci Feistela jest (L_8, R_8) z $L_8 = R_8$. Pokaż, że wynikiem $DES_k(L_0, R_0)$ jest (L_0, R_0) . (Przypomnijmy sobie z ćwiczenia 6.9, że DES przed wyświetleniem wyniku zamienia dwie połówki 16. rundy sieci Feistela.)
- (b) Pokaż, jak znaleźć wejście (L_0, R_0) z właściwością z części (a).

6.13 To pytanie ilustruje atak na potrójne szyfrowanie dwukluczowe. Niech F będzie szyfrem

blokowym o n -bitowej długości bloku i długości klucza k ustaw F_{k1}, k_2

$$(X) \stackrel{\text{def}}{=} F_{k1} (F_{k2}^{-1} (F_{k1} (x)))$$

- (a) Założmy, że mając parę (m_1, m_2) można znaleźć stałą (m_1) . Pokaż, jak odzyskać cały czas wszystkich klawiszy k_2 tak, że $m_2 = F_{k2}^{-1} (F_{k1} (m_1))$ klucz dla F (z dużym prawdopodobieństwem) w czasie około $2n$, używając trzech znanych par wejście/wyjście.
- (b) Ogólnie rzecz biorąc, w stałym czasie nie będzie możliwe znalezienie k_2 jak powyżej. Pokaż jednak, że stosując etap przetwarzania wstępne trwający $2n$, przy danym m_2 można znaleźć w (zasadniczo) stałym czasie wszystkie klucze k_2 takie, że $m_2 = F_{k2}^{-1} (F_{k1} (0^n))$.
- (c) Założmy, że k_1 jest znane i że powyższy etap przetwarzania wstępne został już wykonany. Pokaż, jak używać wartości $y = F(x)$ dla pojedynczego wybranego tekstu jawnego k_1, k_2 x w celu określenia k_2 w stałym czasie.
- (d) Połącz powyższe komponenty, aby opracować atak, który odzyska cały klucz F , uruchamiając się w około $2n$ czasie i żądając szyfrowania około $2n$ wybranych wejść.

6.14 Założmy, że schemat kluczy DES jest modyfikowany w następujący sposób: lewa połowa klucza głównego jest używana do wyprowadzania wszystkich podkluczy w rundach 1–8, podczas gdy prawa połowa klucza głównego jest używana do wyprowadzania wszystkich podkluczy klucze w rundach 9–16. Pokaż atak na ten zmodyfikowany schemat, który odzyska cały klucz w czasie około 228.

- 6.15 Niech $f : \{0, 1\}^m \times \{0, 1\} \rightarrow \{0, 1\}$ i $g : \{0, 1\}^n \times \{0, 1\} \rightarrow \{0, 1\}$ będą bezpiecznymi szyframi blokowymi z $m > n$ i zdefiniuj $Fk_1, k_2(x) = fk_1(gk_2(x))$. Pokaż atak polegający na odzyskaniu klucza na F , używając czasu $O(2m)$ i przestrzeni $O(\cdot 2^n)$.
- 6.16 Zdefiniuj $DESY_k, k(x) = DES_k(x \oplus k)$. Długość klucza DESY wynosi 120 bitów. Pokaż atak polegający na odzyskaniu klucza na DESY, który zajmuje czas i przestrzeń 2^{64} .
- 6.17 Wybieraj losowe S-boxy i miksuje permutacje dla nazw SPN o różnych rozmiarach i opracowuj przeciwko nim ataki różnicowe. Zalecamy wypróbowanie pięciu ciorundowych nazw SPN z 16-bitowymi i 24-bitowymi długościami bloków przy użyciu S-boxów z 4-bitowym wejściem/wyjściem. Napisz kod, aby obliczyć tabele różnicowe i przeprowadzić atak.
- 6.18 Zaimplementuj kompromis czas/przestrzeń dla 40-bitowego DES (tj. ustaw pierwsze 16 bitów klucza DES na 0). Oblicz potrzebny czas i pamięć oraz empirycznie oszacuj prawdopodobieństwo sukcesu. Eksperymentalnie sprawdź wzrost prawdopodobieństwa sukcesu w miarę zwiększenia liczby tabel. (Uwaga: to duży projekt!)
- 6.19 Dla każdej z poniższych konstrukcji funkcji kompresji h z szyfru blokowego F albo pokaż atak, albo udowodnij odporność na kolizje w modelu szyfru idealnego:
- $$(a) h(k, x) = Fk(x). (b) h(k, x) = Fk(x) \oplus k.$$
- 6.20 Poniżej rozważ użycie DES do skonstruowania funkcji kompresji sposobem: Zdefiniuj $h : \{0, 1\}^{112} \rightarrow \{0, 1\}^{64}$ jako $h(x_1, x_2) \stackrel{\text{def}}{=} DES_{x_1}(DES_{x_2}(064))$ gdzie $|x_1| = |x_2| = 56$.
- (a) Zapisz jawną kolizję w h . Wskazówka: skorzystaj z ćwiczenia 6.9(a-b).
 - (b) Pokaż, jak znaleźć obraz wstępny dowolnej wartości y (to znaczy x_1, x_2 tak, że $h(x_1x_2) = y$) w przybliżeniu w 256 czasie.
 - (c) Pokaż sprytniejszy atak przedobrazowy, który trwa mniej więcej 232 lata i zakończy się sukcesem z dużym prawdopodobieństwem. Wskazówka: polegaj na wynikach Załącznika A.4.
- 6.21 Niech F będzie szyfrem blokowym, dla którego łatwo jest znaleźć stałe punkty dla jakiegoś klucza: mianowicie istnieje klucz k , dla którego łatwo jest znaleźć wejścia x , dla których $Fk(x) = x$. Znajdź kolizję w konstrukcji Daviesa-Meyera po zastosowaniu do F . (Rozważ to w świetle ćwiczenia 6.12.)

Rozdział 7

*Konstrukcje teoretyczne

Elementy pierwotne z kluczem symetrycznym

W Rozdziale 3 wprowadziliśmy pojęcie pseudolosowości i zdefiniowaliśmy kilka podstawowych prymitywów kryptograficznych, w tym generatory pseudolosowe, funkcje i permutacje. W rozdziałach 3 i 4 pokazaliśmy, że te elementy podstawowe służą jako elementy składowe całej kryptografii klucza prywatnego. Zrozumienie tych prymitywów z teoretycznego punktu widzenia jest zatem niezwykle ważne. W tym rozdziale formalnie wprowadzimy koncepcję funkcji jednokierunkowych —funkcji, które nieformalnie łatwo obliczyć, ale trudno odwrócić —i pokażemy, jak generatory, funkcje i permutacje pseudolosowe można konstruować przy jedynym założeniu, że istnieją funkcje jednokierunkowe.¹ Co więcej, zobaczymy, że funkcje jednokierunkowe są niezbędne w „nietrywialnej” kryptografii klucza prywatnego. To znaczy: istnienie funkcji jednokierunkowych jest równoznaczne z istnieniem wszelkiej (nietrywialnej) kryptografii klucza prywatnego. Jest to jeden z głównych osiągnięć współczesnej kryptografii.

Konstrukcje, które pokazujemy w tym rozdziale, należy traktować jako uzupełnienie konstrukcji szyfrów strumieniowych i szyfrów blokowych omówionych w poprzednim rozdziale. W poprzednim rozdziale skupiono się na tym, jak różne prymitywy kryptograficzne są obecnie realizowane w praktyce, a intencją tego rozdziału było przedstawienie kilku podstawowych podejść i zasad projektowania, które są stosowane. Nicco rozczerowujący był jednak fakt, że żadnej z pokazanych przez nas konstrukcji nie udało się wykazać bezpieczeństwa w oparciu o słabsze (tj. bardziej rozsądne) założenia. Natomiast w niniejszym rozdziale udowodnimy, że możliwe jest konstruowanie permutacji pseudolosowych wychodząc z bardzo łagodnego założenia o istnieniu funkcji jednokierunkowych. Założenie to jest bardziej akceptowalne niż założenie, powiedzmy, że AES jest permutacją pseudolosową, zarówno dlatego, że jest to założenie słabsze jakościowo, jak i dlatego, że mamy wiele potencjalnych, jednokierunkowych funkcji liczbowych, które były badane przez wiele lat lat, jeszcze przed pojawiением się kryptografii. (Dalsze omówienie tego punktu można znaleźć na samym początku rozdziału 6). Wadą jest jednak to, że konstrukcje, które tu pokazujemy, są znacznie mniej wydajne niż te z rozdziału 6, a zatem nie są faktycznie używane. Ważnym wyzwaniem dla kryptografów pozostaje „wypełnienie tej luki” i rozwój w sp

¹ Nie jest to do końca prawda, ponieważ w tym rozdziale będą działy w większości opierać się na permutacjach jednokierunkowych. Wiadomo jednak, że wystarczą funkcje jednokierunkowe.

bezpieczne konstrukcje generatorów pseudolosowych, funkcji i permutacji, których wydajność jest porównywalna z najlepszymi dostępymi szyframi strumieniowymi i szyframi blokowymi.

Funkcje skrótu odporne na kolizje. W przeciwieństwie do poprzedniego rozdziału, tutaj nie rozważamy funkcji skrótu odpornych na kolizje. Powodem jest to, że konstrukcje takich funkcji skrótu z funkcji jednokierunkowych są nieznane i w rzeczywistości istnieją dowody sugerujące, że takie konstrukcje są niemożliwe.

Do możliwych do udowodnienia konstrukcji odpornych na kolizje funkcji skrótu —opartych na konkretnych założeniach z teorii liczb —przejdzimy w podrozdziale 8.4.2.

Uwaga dotycząca tego rozdziału. Materiał zawarty w tym rozdziale jest nieco bardziej zaawansowany niż materiał zawarty w pozostałej części tej książki. Materiał ten nie jest wyraźnie użyty gdzie indziej, dlatego w razie potrzeby ten rozdział można pominąć. Mimo to staraliśmy się przedstawić materiał w taki sposób, aby był zrozumiały (z wysiłkiem) dla zaawansowanego studenta lub początkującego studenta. Zachęcamy wszystkich czytelników do zapoznania się z sekcjami 7.1 i 7.2, które wprowadzają funkcje jednokierunkowe i zapewniają przegląd pozostałej części tego rozdziału. Wierzymy, że znajomość przynajmniej niektórych poruszanych tu tematów jest na tyle ważna, że uzasadnia podjęcie wysiłku.

7.1 Funkcje jednokierunkowe

W tej sekcji formalnie definiujemy funkcje jednokierunkowe, a następnie krótko omawiamy niektóre kandydatury, co do których powszechnie uważa się, że spełniają tę definicję. (Więcej przykładów przypuszczalnych funkcji jednokierunkowych zobaczymy w rozdziale 8.) Następnie wprowadzimy pojęcie twardych predykatów, które można postrzegać jako opisujące trudność odwracania funkcji jednokierunkowej i które będą szeroko stosowane w konstrukcje, które nastąpią w kolejnych rozdziałach.

7.1.1 Definicje

Funkcja jednokierunkowa $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ jest łatwa do obliczenia, ale trudna do odwrócenia. Pierwszy warunek można łatwo sformalizować: wymagamy po prostu, aby f było obliczalne w czasie wielomianowym. Ponieważ ostatecznie jesteśmy zainteresowani budowaniem schematów kryptograficznych, które dla probabilistycznego przeciwnika działającego w czasie wielomianowym będą trudne do złamania z wyjątkiem znikomego prawdopodobieństwa, sformalizujemy drugi warunek wymagając, aby żaden probabilistyczny algorytm działający w czasie wielomianowym nie mógł odwrócić f — to to znaleźć obraz wstępny danej wartości y — chyba że z znikomym prawdopodobieństwem. Technicznie rzecz biorąc, prawdopodobieństwo to przejmuje się w eksperymencie, w którym y jest generowane poprzez wybranie jednolitego elementu x z dziedziny f , a następnie ustawienie $y := f(x)$ (zamiast

wybierając y równomiernie z zakresu f). Przyczyna tego powinna stać się jasna na podstawie konstrukcji, które zobaczymy w dalszej części rozdziału.

Niech $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ będzie funkcją. Rozważmy następujący eksperyment zdefiniowany dla dowolnego algorytmu A i dowolnej wartości n dla parametru bezpieczeństwa:

Eksperyment odwracający InvertA,f(n)

1. Wybierz uniform $x \in \{0, 1\}^n$ i oblicz $y := f(x)$.
2. A ma na wejściu 1 n i y, a na wyjściu x.
3. Wynik eksperymentu definiuje się jako 1, jeśli $f(x) = y$, i 0 w przeciwnym razie.

Podkreślamy, że A nie musi znajdować oryginalnego obrazu wstępnie x; wystarczy, że A znajdzie dowolną wartość x, dla której $f(x) = y = f(x)$. W drugim kroku nadajemy parametr bezpieczeństwa 1 A, aby podkreślić, że A może przebiegać w wielomianie czasu w parametrze bezpieczeństwa n, niezależnie od długości y.

Możemy teraz zdefiniować, co to znaczy, że funkcja f jest jednokierunkowa.

DEFINICJA 7.1 Funkcja $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ jest jednokierunkowa, jeśli spełnione są dwa warunki:

1. (Łatwe do obliczenia:) Istnieje algorytm czasu wielomianowego Mf obliczający f; to znaczy $Mf(x) = f(x)$ dla wszystkich x.
2. (Trudno odwrócić :) Dla każdego probabilistycznego algorytmu A w czasie wielomianowym istnieje pomijalna funkcja negl taka, że

$$\Pr[\text{InvertA,f}(n) = 1] = \text{negl}(n).$$

Notacja. W tym rozdziale często będziemy wyjaśniać przestrzeń prawdopodobieństwa, dodając ją (jej część) do notacji prawdopodobieństwa. Na przykład, możemy zwięźle wyrazić drugi wymóg w powyższej definicji w następujący sposób: Dla każdego probabilistycznego algorytmu A w czasie wielomianowym istnieje pomijalna funkcja negl taka, że

$$\Pr_{x \in \{0,1\}^n} A(1^n, f(x)) = f_a^{-1}(f(x)) = \text{negl}(n).$$

(Przypomnijmy, że $x \in \{0, 1\}^n$ oznacza, że x jest wybrane równomiernie spośród $\{0, 1\}^n$). Powyższe prawdopodobieństwo przejmuje także losowość stosowaną przez A, co tutaj pozostaje ukryte.

Pomyślna inwersja funkcji jednokierunkowych. Funkcja, która nie jest jednokierunkowa, niekoniecznie jest łatwa do odwrócenia za każdym razem (lub nawet „często”). Przeciwnością drugiego warunku z Definicji 7.1 jest raczej to, że istnieje probabilistyczny algorytm wielomianowy A w czasie i niemożliwa do pominięcia funkcja y

tak, że A odwraca $f(x)$ z prawdopodobieństwem co najmniej $\gamma(n)$ (gdzie prawdopodobieństwo jest brane pod uwagę równomierny wybór $x \in \{0, 1\}^n$ i losowość A). Oznacza to z kolei, że istnieje dodatni wielomian $p(\cdot)$ taki, że dla nieskończoność wielu wartości n algorytm A odwraca f z prawdopodobieństwem co najmniej $1/p(n)$. Zatem jeśli istnieje A, które odwraca f z prawdopodobieństwem n dla wszystkich parzystych wartości n (ale zawsze nie odwraca f , gdy n jest nieparzyste), to f nie jest jednokierunkowe - nawet jeśli A powiedzie się tylko w przypadku połowy wartości n i kończy się sukcesem tylko z prawdopodobieństwem n⁻¹⁰ (dla wartości n, jeśli w ogóle się to powiedzie).

Inwersja czasu wykładniczego. Każdą funkcję jednokierunkową można odwrócić w dowolnym punkcie y czasu wykładniczego, po prostu wypróbowując wszystkie wartości $x \in \{0, 1\}^n$, aż zostanie znaleziona wartość x taka, że $f(x) = y$. Zatem istnienie funkcji jednokierunkowych jest z natury założeniem o złożoności obliczeniowej i twardości obliczeniowej. Oznacza to, że dotyczy problemu, który w zasadzie można rozwiązać, ale zakłada się, że jest trudny do skutecznego rozwiązania.

Permutacje jednokierunkowe. Często będziemy zainteresowani funkcjami jednokierunkowymi z dodatkowymi właściwościami strukturalnymi. Mówimy, że funkcja f zachowuje długość, jeśli $|f(x)| = |x|$ dla wszystkich x. Funkcja jednokierunkowa, która zachowuje długość i jest różna, nazywana jest permutacją jednokierunkową. Jeśli f jest permutacją jednokierunkową, to (y) . Mimo to nadal jest dowolna wartość y ma unikalny obraz wstępujący $x = f^{-1}(y)$ ciężko aby znaleźć x w czasie wielomianowym.

Rodziny funkcji/permutacji jednokierunkowych. Powyższe definicje funkcji jednokierunkowych i permutacji są wygodne, ponieważ uwzględniają pojedynczą funkcję w nieskończonej dziedzinie i zakresie. Jednak wiele kandydujących funkcji jednokierunkowych i permutacji nie pasuje dokładnie do tych ram. Zamiast tego istnieje algorytm generujący pewien zestaw parametrów I definiujących funkcję f_I ; jednokierunkowość oznacza tutaj zasadniczo, że f_I powinno być jednokierunkowe z prawie znikomym prawdopodobieństwem w przypadku wyboru I. Ponieważ każda wartość I definiuje inną funkcję, będziemy teraz odnosić się do rodzin funkcji jednokierunkowych (odpowiednio permutacji). Podamy teraz definicję i odsyłamy czytelnika do następnej sekcji, aby uzyskać konkretny przykład. (Patrz także rozdział 8.4.1.)

DEFINICJA 7.2 Krotka $\Pi = (\text{Gen}, \text{Samp}, f)$ probabilistycznych algorytmów wielomianowych w czasie jest rodziną funkcji, jeśli zachodzi zasada:

1. Algorytm generowania parametrów Gen na wejściu 1 n wyprowadza parametry I za pomocą $I \in \mathcal{I}$. Każda wartość wyjścia I przez Gen definiuje zbiory DI i RI, które stanowią odpowiednio dziedzinę i zakres funkcji f_I .
2. Algorytm próbkowania Samp na wejściu I wyprowadza równomiernie rozłożony element DI.
3. Deterministyczny algorytm oceny f na wejściu $I \times DI$ daje na wyjściu element $y \in RI$. Zapisujemy to jako $y := f_I(x)$.

Π jest rodziną permutacji, jeżeli dla każdej wartości I wyprowadzanej przez $\text{Gen}(1n)$ utrzymuje się, że $DI = RI$ i funkcja $f_I : DI \rightarrow DI$ jest bijekcją.

Niech Π będzie rodziną funkcji. Poniżej znajduje się naturalny odpowiednik eksperymentu wprowadzonego wcześniej.

Doświadczenie odwracające $\text{InvertA}, \Pi(n)$:

1. Przeprowadza się $\text{Gen}(1n)$ w celu uzyskania I , a następnie przeprowadza się $\text{Samp}(I)$ w celu uzyskania jednolitego $x \in DI$. Na koniec obliczane jest $y := f_I(x)$.
2. A ma dane wejściowe I i y , a wyjściem jest x .
3. Wynik eksperymentu wynosi 1, jeśli $f_I(x) = y$.

DEFINICJA 7.3 Rodzina funkcji/permutacji $\Pi = (\text{Gen}, \text{Samp}, f)$ jest jednokierunkowa, jeśli dla wszystkich probabilistycznych algorytmów wielomianowych A istnieje funkcja pomijalna taka, że

$$\Pr[\text{InvertA}, \Pi(n) = 1] = \text{negl}(n).$$

W całym tym rozdziale będziemy pracować z jednokierunkowymi funkcjami/permutacjami w nieskończonej dziedzinie (jak w definicji 7.1), zamiast pracować z rodzinami jednokierunkowych funkcji/permutacji. Służy to przede wszystkim wygodzie i nie wpływa znaczco na żadne wyniki. (Patrz ćwiczenie 7.7.)

7.1.2 Kandydackie funkcje jednokierunkowe

Funkcje jednokierunkowe są interesujące tylko wtedy, gdy istnieją. Nie wiemy, jak bezwzględnie udowodnić ich istnienie (byłby to poważny przełom w teorii złożoności), dlatego musimy się domyślać lub zakładać ich istnienie. Takie przypuszczenie opiera się na fakcie, że wielu naturalnym problemom obliczeniowym poświęcono wiele uwagi, lecz nadal nie ma algorytmu czasu wielomianowego do ich rozwiązania. Być może najbardziej znanym takim problemem jest faktoryzacja liczb całkowitych, tj. znajdowanie czynników pierwszych dużej liczby całkowitej. Łatwo jest pomnożyć dwie liczby i otrzymać ich iloczyn, ale trudniej jest wziąć liczbę i znaleźć jej współczynniki. To prowadzi nas do zdefiniowania funkcji $\text{fmult}(x, y) = x \cdot y$. Jeśli nie nałożymy żadnych ograniczeń na długości x i y , to fmult łatwo odwrócić: z dużym prawdopodobieństwem $x \cdot y$ będzie parzyste, w tym przypadku $(2, xy/2)$ jest odwrotnością.

Problem ten można rozwiązać, ograniczając dziedzinę fmult do liczb pierwszych x i y o równej długości. Powróćmy do tego pomysłu w podrozdziale 8.2.

Inna kandydująca funkcja jednokierunkowa, nie opierająca się bezpośrednio na teorii liczb, opiera się na problemie sumy podzbiorów i jest zdefiniowana przez

$$\text{fss}(x_1, \dots, x_n, j) = x_1, \dots, x_n, \quad j \in \mathbb{Z} \text{ mod } 2n,$$

gdzie każdy x_i jest n-bitowym ciągiem znaków interpretowanym jako liczba całkowita, a J jest n-bitowym ciągiem znaków interpretowanym jako określający podzbiór $\{1, \dots, N\}$. Odwrócenie fss na wyjściu (x_1, \dots, x_n, y) wymaga znalezienia podzbioru $J \subseteq \{1, \dots, n\}$ tak, że $j \in J$ $x_j = y \bmod 2^n$. Studenci,

którzy badali N P-zupełność, mogą przypomnieć sobie, że ten problem jest N P-zupełny. Ale nawet $P = NP$ nie oznacza, że fss jest jednokierunkowa: $P = NP$ oznaczałoby, że każdy algorytm czasu wielomianowego nie rozwiązuje problemu sumy podzbiorów na co najmniej jednym wejściu, podczas gdy fss jest funkcją jednokierunkową wymagane jest, aby każdy algorytm czasu wielomianowego prawie zawsze nie rozwiązywał problemu sumy podzbiorów (przynajmniej dla niektórych parametrów). Zatem nasze przekonanie, że powyższa funkcja jest jednokierunkowa, opiera się na braku znanych algorytmów pozwalających rozwiązać ten problem nawet z „małym” prawdopodobieństwem na losowych danych wejściowych, a nie tylko na fakcie, że problem jest N P-zupełny.

Na koniec pokażemy rodzinę permutacji, które uważa się za jednokierunkowe. Niech Gen będzie dzie probabilistycznym algorytmem wielomianowym, który na wejściu $1n$ generuje n-bitową liczbę pierwszą p wraz ze specjalnym elementem $g \in \{2, \dots, p-1\}$. (The ; patrz Sekcja 8.3.3.) Niech $Samp$ element g powinien być generatorem Z_p będzie dzie algorytm, który przy danych p i g generuje jednolitą liczbę całkowitą $x \in \{1, \dots, p-1\}$.

Na koniec zdefiniuj

$$fp, g(x) = [g x \bmod p].$$

(Fakt, że fp, g można obliczyć efektywnie, wynika z wyników w Załączniku B.2.3.) Można wykazać, że ta funkcja jest różnowartościowa, a zatem jest permutacją. Zakładana trudność w odwróceniu tej funkcji opiera się na przypuszczalnej trudności problemu logarytmu dyskretnego; bę dziemy mieli o wiele więcej cej do powiedzenia na ten temat w sekcji 8.3.

Na koniec zauważamy, że bardzo wydajne funkcje jednokierunkowe można uzyskać z praktycznych konstrukcji kryptograficznych, takich jak SHA-1 lub AES, przy założeniu, że są one odpowiednio odporne na kolizje lub permutację pseudolosową; patrz ćwiczenia 7.4 i 7.5. (Technicznie rzecz biorąc, nie mogą one spełniać definicji jednokierunkowości, ponieważ mają wejście/wyjście o stałej długości, więc nie możemy patrzeć na ich asymptotyczne zachowanie. Niemniej jednak można przypuszczać, że są one jednokierunkowe konkretny sens.)

7.1.3 Predykaty twarde

Z definicji funkcję jednokierunkową trudno odwrócić. Inaczej mówiąc: biorąc pod uwagę $y = f(x)$, wartość x nie może zostać obliczona w całości za pomocą żadnego algorytmu czasu wielomianowego (z wyjątkiem znikomego prawdopodobieństwa; ignorujemy to tutaj). Można odnieść wrażenie, że na podstawie $f(x)$ w czasie wielomianowym nie można określić niczego na temat x . To nie jest to regułą. Rzeczywiście, $f(x)$ może „wyiec” wiele informacji na temat x , nawet jeśli f jest jednokierunkowe. Dla banału

przykładowo, niech g będzie funkcją jednokierunkową i zdefiniujemy $f(x_1, x_2) \stackrel{\text{def}}{=} (x_1, g(x_2))$, gdzie $|x_1| = |x_2|$. Łatwo pokazać, że f jest także funkcją jednokierunkową (pozostawiamy to jako ćwiczenie), mimo że ujawnia połowę wartości wejściowych.

W naszych zastosowaniach będą dziemy musieli zidentyfikować konkretną informację o x , która jest „ukryta” przez $f(x)$. Motywuje to koncepcję twardego predykatu. Twardy predykat $hc : \{0, 1\}^n \rightarrow \{0, 1\}$ funkcji f ma tę właściwość, że $hc(x)$ jest trudne do obliczenia z prawdopodobieństwem znacznie większym niż $1/2$ przy $f(x)$. (Ponieważ hc jest funkcją boolowską, zawsze można obliczyć $hc(x)$ z prawdopodobieństwem $1/2$ w drodze losowego zgadywania.) Formalnie:

DEFINICJA 7.4 Funkcja $hc : \{0, 1\}^n \rightarrow \{0, 1\}$ jest twardym predykatem funkcji f , jeśli hc można obliczyć w czasie wielomianowym i dla każdego probabilistycznego algorytmu wielomianowego A istnieje pomijalna funkcja negl taka, że

$$\Pr_{x \in \{0,1\}^n} [A(1^n, f(x)) = hc(x)] \geq \frac{1}{2} + \text{zaniechanie}(n),$$

gdzie prawdopodobieństwo uwzględnia się jednorodny wybór x w $\{0, 1\}^n$ i losowość A .

Podkreślamy, że $hc(x)$ można efektywnie obliczyć, biorąc pod uwagę x (ponieważ funkcję hc można obliczyć w czasie wielomianowym); definicja wymaga, aby $hc(x)$ było trudne do obliczenia, biorąc pod uwagę $f(x)$. Powyższa definicja nie wymaga, aby f było jednokierunkowe; jeśli jednak f jest permutacją, to nie może mieć twardego predykatu, chyba że jest jednokierunkowy. (Patrz ćwiczenie 7.13.)

Proste pomysły nie działają. Rozważ predykat $hc(x)$

x_1, \dots, x_n bity x . Można mieć nadzieję, że to hardkor

predykat dowolnej funkcji jednokierunkowej f : jeśli f nie można odwrócić, to $f(x)$ musi ukrywać przynajmniej jeden z bitów x_i swojego obrazu wstępnego x , co wydaje się sugerować, że wyłączność lub wszystkich bitów x jest trudne do obliczenia. Pomimo odwołania argument ten jest błędny. Aby to zobaczyć, niech g będzie funkcją

i zdefiniuj $f(x) = (g(x)_i)_{i=1}^n$ (czyli $f(x)$ jest n -krotką jednokierunkową x). Nie jest trudno pokazać, że f jest jednokierunkowa, jednak jasne, że $f(x)$ nie ukrywa wartości $hc(x) = \sum_{i=1}^n x_i$, ponieważ jest to część jego wyniku; zatem $hc(x)$ nie jest twardym predykatem f . Rozszerzając to, można wykazać, że dla dowolnego stałego predykatu hc istnieje funkcja jednokierunkowa f , dla której hc nie jest twardym predykatem f .

Trywialne, twardy predykaty. Niektóre funkcje mają „trywialne” i twardy predykaty. Na przykład, niech f będzie funkcją, która usuwa ostatni bit ze swojego wejścia (tj. $f(x_1 \dots x_n) = x_1 \dots x_{n-1}$). Trudno jest określić x_n , biorąc pod uwagę $f(x)$, ponieważ x_n jest niezależne od wyniku; zatem $hc(x) = x_n$ jest twardym predykatem f . Jednakże f nie jest jednokierunkowe. Kiedy w naszych konstrukcjach użyjemy twardych predykatów, stanie się jasne, dlaczego trywialne, twardy predykaty tego rodzaju są bezużyteczne.

7.2 Od funkcji jednokierunkowych do pseudolosowości

Celem tego rozdziału jest pokazanie, jak konstruować generatory, funkcje i permutacje pseudolosowe w oparciu o dowolną jednokierunkową funkcję /permutację . W tej części omówimy te konstrukcje. Szczegóły podano w kolejnych sekcjach.

Twardy predykat dowolnej funkcji jednokierunkowej. Pierwszym krokiem jest pokazanie, że dla dowolnej funkcji jednokierunkowej istnieje twardy predykat. W rzeczywistości pozostaje otwarte, czy to prawda; pokazujemy coś słabszego, co wystarcza do naszych celów. Mianowicie pokazujemy, że mając funkcję jednokierunkową f , możemy skonstruować inną funkcję jednokierunkową g wraz z twardym predykatem g .

TWIERDZENIE 7.5 (Twierdzenie Goldreicha-Levina) Założmy, że istnieją funkcje jednokierunkowe (lub permutacje). Następnie istnieje funkcja jednokierunkowa (odpowiednio permutacja) g i twardy predykat hc g .

Niech f będzie funkcją jednokierunkową. Funkcje g i hc są zbudowane w następujący sposób: zbiór $g(x, r) \stackrel{\text{def}}{=} (f(x), r)$, dla $|x| = |r|$ i zdefiniuj

$$hc(x, r) \stackrel{\text{def}}{=} \sum_{i=1}^N x_i \cdot r_i ,$$

gdzie x_i (odpowiednio, r_i) oznacza i -ty bit x (odpowiednio, r). Zauważ, że jeśli r jest jednolite, to $hc(x, r)$ daje w wyniku wyłączny-lub losowy podzbiór bitów x . (Gdy $r_i = 1$, bit x_i jest zawarty w XOR, w przeciwnym razie nie.) Twierdzenie Goldreicha-Levina zasadniczo stwierdza, że jeśli f jest funkcją jednokierunkową, to $f(x)$ ukrywa wyłączność lub losowość podzbiór bitów x .

Generatory pseudolosowe z permutacji jednokierunkowych. Następny krokiem jest pokazanie, jak można wykorzystać twardy predykat jednokierunkowej permutacji do skonstruowania generatora pseudolosowego. (Wiadomo, że wystarczy twardy predykat funkcji jednokierunkowej, ale dowód jest niezwykle skomplikowany i wykracza poza zakres tej książki.) W szczególności pokażemy:

TWIERDZENIE 7.6 Niech f będzie permutacją jednokierunkową i niech hc będzie twardą $= f(s)$ orzeczeniem f . Następnie $hc(s)$ jest generatorem pseudolosowym z współczynnikiem ekspansji $G(s)(n) = n + 1$.

Aby dowiedzieć się, dlaczego G zgodnie z twierdzeniem stanowi generator pseudolosowy, zauważmy najpierw, że początkowe n bitów wyniku $G(s)$ (tj. bitów $f(s)$) jest naprawdę równomiernie rozłożonych, gdy s jest równomiernie rozłożone, ze względu na fakt, że f jest permutacją. Następnie fakt, że hc jest twardym predykatem f , oznacza, że $hc(s)$ „wygląda losowo” – tj. jest pseudo-

losowe —nawet przy danym $f(s)$ (zakładając ponownie, że s jest jednolite). Łącząc te obserwacje, widzimy, że cały wynik G jest pseudolosowy.

Generatory pseudolosowe z dowolną ekspansją. Istnienie generatora pseudolosowego, który rozciąga swoje ziarno nawet o jeden bit (jak właśnie widzieliśmy) jest już wysoce nietrywialne. Jednak do zastosowań (np. do wydajnego szyfrowania dużych wiadomości, jak w podrozdziale 3.3) potrzebujemy generatora pseudolosowego o znacznie więcej ksyym rozszerzeniu. Na szczęście możemy uzyskać dowolny współczynnik rozwinięcia wielomianu:

TWIERDZENIE 7.7 Jeżeli istnieje generator pseudolosowy o współczynniku rozwinięcia $cia(n) = n+1$, to dla dowolnego wielomianu istnieje generator pseudolosowy o współczynniku rozwinięcia $cia\ poli(n)$.

Dochodzimy do wniosku, że generatory pseudolosowe z dowolnym (wielomianem) rozwinięciem cie można skonstruować z dowolnej permutacji jednokierunkowej.

Funkcje/permutacje pseudolosowe z generatorów pseudolosowych. Generatory pseudolosowe wystarczą do konstruowania schematów szyfrowania klucza prywatnego z zabezpieczeniem EAV. Aby osiągnąć szyfrowanie klucza prywatnego zabezpieczonego CPA (nie wspominając o kodach uwierzytelniających wiadomości), polegaliśmy jednak na funkcjach pseudolosowych. Poniższy wynik pokazuje, że ten drugi można zbudować z pierwszego:

TWIERDZENIE 7.8 Jeżeli istnieje generator pseudolosowy ze współczynnikiem rozwinięcia $cia(n) = 2n$, to istnieje funkcja pseudolosowa.

Tak naprawdę możemy zrobić jeszcze więcej:

TWIERDZENIE 7.9 Jeżeli istnieje funkcja pseudolosowa, to istnieje silna permutacja pseudolosowa.

Łącząc wszystkie powyższe twierdzenia, a także wyniki rozdziałów 3 i 4, otrzymujemy następujące wnioski:

WNIOSEK 7.10 Zakładając istnienie permutacji jednokierunkowych, istnieją generatory pseudolosowe z dowolnym współczynnikiem rozwinięcia wielomianu, funkcjami pseudolosowymi i silnymi permutacjami pseudolosowymi.

WNIOSEK 7.11 Zakładając istnienie permutacji jednokierunkowych, istnieją schematy szyfrowania klucza prywatnego zabezpieczonego przez CCA oraz bezpieczne kody uwierzytelniania wiadomości.

Jak wspomniano wcześniej, możliwe jest uzyskanie wszystkich tych wyników wyłącznie w oparciu o istnienie funkcji jednokierunkowych.

7.3 Predykaty twardze z funkcji jednokierunkowych

W tej sekcji udowadnimy Twierdzenie 7.5, pokazując, co następuje:

TWIERDZENIE 7.12 Niech f będzie funkcją jednokierunkową i zdefiniuje g przez $g(x, r) \stackrel{\text{def}}{=} f(x), r$, gdzie $|x| = |r|$. Zdefiniuj $g_i(x, r) \stackrel{\text{def}}{=} x_i \cdot r_i$, gdzie $x = \begin{smallmatrix} N \\ x_1 \cdots x_n \end{smallmatrix}$ oraz $r = \begin{smallmatrix} n \\ r_1 \cdots r_n \end{smallmatrix}$. Wtedy g_i jest twardym predykatem g .

Ze względu na złożoność dowodu udowadniamy trzy kolejno silniejsze wyniki, których kulminacją jest to, co twierdzi twierdzenie.

7.3.1 Prosty przypadek

Najpierw pokazujemy, że jeśli istnieje wielomianowy przeciwnik A , który zawsze poprawnie oblicza $g_i(x, r)$ przy założeniu $g(x, r) = f(x), r$, to możliwe jest odwrócenie f w czasie wielomianowym. Z założenia, że f jest funkcją jednokierunkową, wynika, że taki przeciwnik A nie istnieje.

TWIERDZENIE 7.13 Niech f i g będą takie jak w Twierdzeniu 7.12. Jeśli istnieje algorytm wielomianowy A taki, że $A(f(x), r) = g(x, r)$ dla wszystkich n i wszystkich $x, r \in \{0, 1\}^n$, to istnieje wielomianowy algorytm czasowy algorytm A taki, że $A(1n, f(x)) = x$ dla wszystkich n i wszystkich $x \in \{0, 1\}^n$.

DOWÓD Konstruujemy A w następujący sposób. $A(1n, y)$ oblicza $x_i := A(y, e^i)$ dla $i = 1, \dots, n$, gdzie e oznacza n -bitowy ciąg znaków, w którym 1 jest na i -tym miejscu, a 0 wszędzie indziej. Następnie A wyprowadza $x = x_1 \cdots x_n$. Oczywiście A przebiega w czasie wielomianowym.

W wykonaniu $A(1n, f(\hat{x}))$ wartość x_i obliczona przez A spełnia

$$x_i = A(f(\hat{x}), e^i) = g(\hat{x}, e^i) = \sum_{j=1}^N x_j \cdot e^{i-j} = x_i . J$$

Zatem $x_i = \hat{x}_i$ dla wszystkich i , więc A wyprowadza poprawną odwrotność $x = \hat{x}$. ■

Jeżeli f jest jednokierunkowe, żaden probabilistyczny algorytm czasu wielomianowego nie jest w stanie odwrócić f z niezanielbywalnym prawdopodobieństwem. Zatem dochodzimy do wniosku, że nie ma algorytmu czasu wielomianowego, który zawsze poprawnie oblicza $g(x, r)$ z $(f(x), r)$. Jest to raczej słaby wynik, bardzo odległy od naszego ostatecznego celu, jakim jest pokazanie, że $g(x, r)$ nie może zostać obliczone (z prawdopodobieństwem znacznie większym niż $1/2$) przy danych $(f(x), r)$.

7.3.2 Bardziej zawiła sprawa

Pokazujemy teraz, że dla dowolnego probabilistycznego algorytmu wielomianowego czasu A trudno jest obliczyć $gl(x, r)$ z $(f(x), r)$ z prawdopodobieństwem znacznie większym niż $3/4$. Ponownie pokażemy, że dowolne takie A implikowałoby istnienie algorytmu A działającego w czasie wielomianowym, który odwraca f z niezależnym prawdopodobieństwem. Zauważ, że strategia przedstawiona w dowodzie Twierdzenia 7.13 zawodzi tutaj, ponieważ może się zdarzyć, że A nigdy się nie powiedzie, gdy $r = e$ (choć może się udać, powiedzmy, w przypadku wszystkich innych wartości r). Ponadto w tym przypadku A nie wie, czy wynik $A(f(x), r)$ jest równy $gl(x, r)$ czy nie; jedyne, co A wie, to to, że z dużym prawdopodobieństwem algorytm A jest poprawny. To jeszcze bardziej komplikuje dowód.

TWIERDZENIE 7.14 Niech f i gl będą takie jak w Twierdzeniu 7.12. Jeżeli istnieje probabilistyczny algorytm wielomianowo-czasowy A i wielomian $p(\cdot)$ taki, że

$$\Pr_{x,r \in \{0,1\}^n} A(f(x), r) = gl(x, r) + p(n) \frac{3}{4} - \frac{1}{p(n)}$$

dla nieskończenie wielu wartości n istnieje probabilistyczny algorytm wielomianowy A taki, że

$$\Pr_{x \in \{0,1\}^n} ZA(1^n, f(x)) = \frac{1}{4} (f(x)) + \frac{1}{4 \cdot p(n)}$$

dla nieskończenie wielu wartości n.

DOWÓD Główną obserwacją leżącą u podstaw dowodu tego twierdzenia jest to, że dla każdego $r \in \{0, 1\}^n$ wartości $gl(x, r)$ i e użyte w $gl(x, r)$ razem mogą oznaczać do obliczenia i-tego bitu x. (Przypomnijmy, że e to n -bitowy ciąg znaków wszędzie z wyjątkiem i-tego stanowiska.) Jest to prawdą, ponieważ

$$\begin{aligned} gl(x, r) &= gl(x, r \quad e \quad) \\ &= \sum_{nj=1}^I x_j \cdot r_j \quad \sum_{nj=1}^I x_j \cdot (r_j \quad e \quad) = xi \cdot ri \quad xi \cdot \bar{ri} = xi \end{aligned}$$

gdzie \bar{ri} jest dopełnieniem ri , a druga równość wynika z faktu, że dla $j = i$ wartość $x_j \cdot r_j$ pojawia się w obu sumach i dlatego jest anulowana.

Powyzsze pokazuje, że jeśli A odpowie poprawnie w obu przypadkach $(f(x), r)$ i $(f(x), r \quad e)$, to A może poprawnie obliczyć xi . Niestety, A nie wie, kiedy A odpowiada poprawnie, a kiedy nie; A wie tylko, że A odpowiada poprawnie z „wysokim” prawdopodobieństwem. Z tego powodu A używa wielu losowych wartości r, używając każdej z nich do uzyskać oszacowanie xi , a następnie przyjąć oszacowanie występujące w większości przypadków jako ostateczne przypuszczenie dla xi .

Na wstępie pokazujemy, że dla wielu x prawdopodobieństwo, że A odpowie poprawnie zarówno dla $(f(x), r)$, jak i $(f(x), r')$, jest wystarczająco wysokie. ^I), gdy r jest jednolite, to pozwala nam ustalić x , a następnie skupić się wyłącznie na jednolitym wyborze r , co ułatwia analizę .

Twierdzenie 7.15 Niech n będzie takie, że

$$\Pr_{x,r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] + p(n) \cdot \frac{3}{4} - \frac{1}{p(n)}.$$

Wtedy istnieje zbiór $S_n \subseteq \{0, 1\}^n$ o rozmiarze co najmniej co $x \in S_n$ i utrzymuje to

$$\Pr_{x,r \in \{0,1\}^n} [A(f(x), r) \neq gl(x, r)] \leq \frac{3}{4} - \frac{1}{2p(n)}.$$

DOWÓD Niech $\varepsilon(n) = 1/p(n)$ i zdefiniuj $S_n \subseteq \{0, 1\}^n$ jako zbiór wszystkich x , dla których

$$\Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] + 2 \cdot \frac{3}{4} \cdot \frac{\varepsilon(n)}{p(n)}.$$

Mamy

$$\begin{aligned} \Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] &= \frac{1}{n} \\ &= \frac{1}{2^{rz}} \sum_{r \in S_n} \Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] \\ &\quad + \frac{1}{2^{nr}} \sum_{r \in \{0,1\}^n \setminus S_n} \Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] \\ &\leq \frac{|S_n|}{2^{rz}} + \frac{1}{2^{rz}} \cdot \frac{3 + 4 \cdot \frac{\varepsilon(n)}{2}}{2} \\ &\leq \frac{|S_n|}{2^{rz}} + \frac{3 + \frac{\varepsilon(n)}{2}}{2}. \end{aligned}$$

Od $\frac{3}{4} + \varepsilon(n) = \Pr_{x,r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)]$, algebra prostą daje $|S_n| \leq 2^{\frac{rz}{2}}$

■

Poniżej znajduje się następna prosta konsekwencja.

Twierdzenie 7.16 Niech n będzie takie, że

$$\Pr_{x,r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] + 4 \cdot \frac{3}{4} - \frac{1}{p(n)}.$$

Wtedy istnieje zbiór $S_n = \{0, 1\}^n$ o rozmiarze co najmniej $\frac{1}{2p(n)} \cdot 2^n$ takie, że dla $x \in S_n$ i każde i i potwierdza to

$$\Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] \geq \Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r) \text{ i } f^{(1)}(x) = gl^{(1)}(x, r)] + \frac{1}{2p(n)} \cdot \frac{1}{2}.$$

DOWÓD Niech $\epsilon(n) = 1/p(n)$ i przyjmijmy, że S_n jest zbiorem gwarantowanym przez poprzednie twierdzenie. Wiemy, że dla dowolnego $x \in S_n$

$$\Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] \geq \frac{1}{2} + \frac{\epsilon(n)}{2}.$$

Naprawdę $\Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r)] \geq \frac{1}{2}$; zatem

$$\Pr_{r \in \{0,1\}^n} [A(f(x), r) = gl(x, r) \text{ i } f^{(1)}(x) = gl^{(1)}(x, r)] \geq \frac{1}{4} + \frac{\epsilon(n)}{2}.$$

Interesuje nas dolna granica prawdopodobieństwa, że A wyświetli poprawną odpowiedź zarówno dla $gl(x, r)$, jak i $gl(x, r - \epsilon)$; równoważnie, chcemy zwiększyć górną granicę prawdopodobieństwa, że A nie wyświetli poprawnej odpowiedzi w żadnym z tych przypadków. Zauważ, że $r \in \{0, 1\}^n$ i $r - \epsilon$ są niezależne, więc nie możemy po prostu pomnożyć prawdopodobieństw niepowodzenia. Możemy jednak zastosować ograniczenie unii (patrz Twierdzenie A.7) i zsumować prawdopodobieństwa niepowodzenia. Oznacza to, że prawdopodobieństwo, że A jest niepoprawne albo w $f^{(1)}(x) = gl^{(1)}(x, r - \epsilon)$ jest co najwyżej

$$\frac{1}{4} + \frac{1}{2} + \frac{\epsilon(n)}{2} = \frac{1}{2} + \frac{\epsilon(n)}{2},$$

i tak A jest poprawne zarówno w przypadku $gl(x, r)$, jak i $gl(x, r - \epsilon)$ przynajmniej z prawdopodobieństwem $\frac{1}{2} + \epsilon(n)$. To potwierdza twierdzenie. ■

W dalszej części dowodu ustalamy $\epsilon(n) = 1/p(n)$ i uwzględniamy tylko te wartości n , dla których

$$\Pr_{x \in \{0,1\}^n} [A(f(x), r) = gl(x, r) \text{ i } f^{(1)}(x) = gl^{(1)}(x, r)] \leq \frac{1}{2} + \frac{\epsilon(n)}{2}. \quad (7.1)$$

Poprzednie twierdzenie stwierdza, że dla ułamka $\epsilon(n)/2$ wejść x i dowolnego i , algorytm A odpowiada poprawnie zarówno na $(f(x), r)$, jak i $(f(x), r - \epsilon)$ zdolność co najmniej $1/2 + \epsilon(n)$ nad jednolitym wyborem r i od tej chwili skupiamy się tylko na takich wartościach x . Konstruujemy probabilistyczny algorytm wielomianowo-czasowy A , który odwraca $f(x)$ z prawdopodobieństwem co najmniej $1/2$ gdy $x \in S_n$. To wystarczy do udowodnienia Twierdzenia 7.14 od tego momentu dla nieskończenie wielu wartości n ,

$$\begin{aligned} & \Pr_{x \in \{0,1\}^n} [A(1^n, f(x)) = f^{(1)}(x)] \\ &= \Pr_{x \in \{0,1\}^n} [A(1^n, f(x)) = f^{(1)}(x) \mid x \in S_n] \cdot \Pr_{x \in \{0,1\}^n \setminus S_n} [x \in S_n] \\ &= \frac{1}{2} \cdot \frac{\epsilon(n)}{2} = \frac{1}{4p(n)}. \end{aligned}$$

Algorytm A, podany jako wejście $1n$ i y , działa następująco:

wykonaj: 1. Dla $i = 1, \dots, n$,

- Wybierz wielokrotnie jednorodny $r \in \{0, 1\}^n$ i oblicz $A(y, r)$ jako $A(y, r) = e^{f(x)}$, „oszacowanie” dla i -tego bitu obrazu wstępnienego y . Po wykonaniu tej czynności wystarczająco wiele razy (patrz poniżej), niech x_i będzie dziełem „oszacowaniem”, które występuje przez więcej czasu.

2. Wyjście $x = x_1 \dots x_n$.

Naszkicujemy analizę prawdopodobieństwa, że A poprawnie odwraca dane wejściowe y . (Pozwolimy sobie na nieco lakoniczność, gdyż pełny dowód dla bardziej skomplikowanego przypadku podany jest w następnej sekcji.) Powiedzmy $y = f(x)$ i przypomnijmy sobie, że zakładamy tutaj, że n jest takie, że równanie (7.1) zachodzi dla $i = 1 \dots n$. Napraw niektóre i.

Z poprzedniego twierdzenia wynika, że oszacowanie $A(y, r) = A(y, r) = e^{f(x)}$ jest równe $g(x, e^r)$ z prawdopodobieństwem co najmniej $\frac{1}{2}$ dla wyboru r . Uzyskując wystarczająco wiele estymatorów i przyjmując, że x_i będzie wartością więcej czasów, A może zapewnić, że x_i będzie równe $g(x, e^r)$ z prawdopodobieństwem co najmniej $\frac{1}{2}$. Oczywiście musimy się upewnić, że wielomianowo wiele oszacowań wystarczy. Na szczęście, ponieważ $\epsilon(n) = 1/p(n)$ dla pewnego wielomianu p i niezależna wartość r jest używana do otrzymania każdego oszacowania, granica Chernoffa (por. Twierdzenie A.14) pokazuje, że wielomianowo wystarczy wiele oszacowań.

Podsumowując, dla każdego i wartość x_i obliczona przez A jest niepoprawna z prawdopodobieństwem co najwyżej $\frac{1}{2}$. Suma pokazuje zatem, że A jest niepoprawne dla pewnego i z prawdopodobieństwem co najwyżej $n \cdot \frac{1}{2}$. Oznacza to, że A jest poprawne dla wszystkich i – a zatem poprawnie odwraca y – z prawdopodobieństwem co najmniej $1 - \frac{n}{2} = \frac{1}{2}$. To kończy dowód Twierdzenia 7.14. ■

Konsekwencją Twierdzenia 7.14 jest to, że jeśli f jest funkcją jednokierunkową, to dla dowolnego wielomianowego algorytmu A prawdopodobieństwo, że A poprawnie odgadnie $g(x, r)$, gdy jest dane $(f(x), r)$, jest co najwyżej pomijalne więcej niż $3/4$.

7.3.3 Pełny dowód

Zakładamy znajomość uproszczonych dowodów z poprzednich części i opieramy się na opracowanych tam pomysłach. Opieramy się na terminologii i standardowych wynikach teorii prawdopodobieństwa omówionych w Załączniku A.3.

Dowodzimy następującego twierdzenia, z którego wynika Twierdzenie 7.12:

TWIERDZENIE 7.17 Niech f i g będą takie jak w Twierdzeniu 7.12. Jeżeli istnieje probabilistyczny algorytm wielomianowo-czasowy A i wielomian $p(\cdot)$ taki, że

$$\Pr_{x, r \in \{0, 1\}^n} A(f(x), r) = g(x, r) \quad \frac{1}{2} + \frac{1}{p(n)}$$

dla nieskończoność wielu wartości n istnieje probabilistyczny algorytm wielomianowo-czasowy A oraz wielomian p(·) taki, że

$$\Pr_{x \in \{0,1\}^n} ZA(1n, f(x)) = p\left(\frac{1}{n}\right)$$

dla nieskończoność wielu wartości n.

DOWÓD Po raz kolejny ustalamy $\epsilon(n) = 1/p(n)$ i rozważamy tylko te wartości n, dla których $\Pr_{x,r \in \{0,1\}^n} A(f(x), r) = gl(x, r) = p(n)$. Poniższe stwierdzenie jest analogiczne do Twierdzenia 7.15 i jest dowodzone w ten sam sposób.

Twierdzenie 7.18 Niech n będzie takie, że

$$\Pr_{x,r \in \{0,1\}^n} A(f(x), r) = gl(x, r) = \frac{1}{n} + \epsilon(n).$$

Wtedy istnieje zbiór $S_n \subseteq \{0,1\}^n$ o rozmiarze co najmniej $\epsilon(n) \cdot 2^n$ tak, że dla każdego $x \in S_n$ i potwierdza to

$$\Pr_{r \in \{0,1\}^n} A(f(x), r) = gl(x, r) = \frac{1}{n} + \frac{\epsilon(n)}{2^n}. \quad (7.2)$$

Jeśli zaczniemy od próby udowodnienia analogii Twierdzenia 7.16, najlepsze, co możemy tutaj stwierdzić, to to, że gdy $x \in S_n$ mamy

$$\Pr_{r \in \{0,1\}^n} A(f(x), r) = gl(x, r) = gl(x, r) = \frac{1}{n} + \epsilon(n)$$

dla każdego i. Zatem, jeśli spróbujemy użyć $A(f(x), r) = gl(x, r) = e$ dla x_i^I jako oszacowanie możemy jedynie twierdzić, że to oszacowanie będzie poprawne z prawdopodobieństwem co najmniej $\epsilon(n)$, co nie możemy twierdzić, że odwrócenie wyniku daje dobry oszacowanie.

Zamiast tego projektujemy A tak, aby obliczał $gl(x, r)$ i $gl(x, r) = e$ dla x_i^I wywołując A i Tylko raz. Robimy to poprzez przebieg $A(x, r) = e$ pozwalać A po prostu „odgadnąć” samą wartość $gl(x, r)$. Naiwnym sposobem byłoby wybranie r niezależnie, tak jak poprzednio, i umożliwienie A niezależnego zgadnięcia $gl(x, r)$ dla każdej wartości r. Ale wtedy prawdopodobieństwo, że wszystkie takie domysły są prawidłowe – co, jak zobaczymy, jest konieczne, jeśli A ma wyprowadzić poprawną odwrotność – byłoby znikome, ponieważ zastosowano wielomianową liczbę r.

Kluczową obserwacją niniejszego dowodu jest to, że A może wygenerować r w sposób niezależny od par i dokonać swoich domysłów w określony sposób, tak że z niezaniedbywalnym prawdopodobieństwem wszystkie jego domysły będą prawidłowe. W szczególności, aby wygenerować m wartości r, mamy A wybierz r_1, r_2, \dots, r_m niezależne i równomiernie rozłożone ciągi s $s_1, s_2, \dots, s_m \in \{0,1\}^n$. Następnie dla każdego niepustego podzbiór I $\{1, \dots, m\}$, ustawiamy $r_I := s_i$ dla $i \in I$. Ponieważ istnieje 2^n

niepuste podzbiory, definiuje to zbiór $2\log(m+1) - 1$ m ciągów.

Każdy taki串 jest równomiernie rozłożony. Ciągi nie są niezależne, ale są niezależne parami. Aby to zobaczyć, zauważmy, że dla każdych dwóch podzbiorów $I = J$ istnieje indeks $j \in I \setminus J$ taki, że $j \in I \setminus J$. Bez utraty j jest jednorodne i $j \in I$. Następnie wartość $s \in I \setminus j$ jest zawarte w I niezależne od ogólności, przyjmijmy to, że $J \cap I = \emptyset$, który definiuje r wartość r . Ponieważ s oznacza j ,

r jest jednorodny i niezależny od r^{-1} również.

Mamy teraz dwie ważne obserwacje:

1. Biorąc pod uwagę $gl(x, s_1), \dots, gl(x, s_m)$, można obliczyć $gl(x, rI)$ dla każdego podzbiór $I \subseteq \{1, \dots, m\}$. To dlatego, że

$$gl(x, rI) = gl(x, \bigcup_{i \in I} s_i^{-1}) = \bigcup_{i \in I} gl(x, s_i).$$

2. Jeśli A po prostu odgadnie wartości $gl(x, s_1), \dots, gl(x, s_m)$ wybierając dla każdego jednolity bit, wówczas wszystkie te domysły będą poprawne z prawdopodobieństwem $1/2$. Jeśli m jest wielomianem w parametrze bezpieczeństwa n , to $1/2$ nie jest pomijalne, a zatem z niezaniebywalnym prawdopodobieństwem A poprawnie odgaduje wszystkie wartości $gl(x, s_1), \dots, gl(x, s_m)$.

Połączenie powyższych daje sposób otrzymania $m = \text{poly}(n)$ uniformnych s_i wraz z niezależnymi串 $\{r \in \mathbb{F}_q^m\}$ poprawnymi wartościami dla $\{gl(x, rI)\}$ parami niezanebywalnym prawdopodobieństwem. Wartości te można następnie wykorzystać do obliczenia x_i w taki sam sposób, jak w dowodzie Twierdzenia 7.14. Szczegóły poniżej.

Algorytm inwersji A . Podajemy teraz pełny opis algorytmu A , który otrzymuje dane wejściowe $1n$, y i próbuje obliczyć odwrotność y .

Algorytm przebiega w następujący sposób:

1. Ustaw $\epsilon := \log(2n/\epsilon(n))^{-2} + 1$.
2. Wybierz jednolite, niezależne $s^{-1}, \dots, s_m \in \{0, 1\}^n$ i $\sigma^{-1}, \dots, \sigma_m \in \{0, 1\}^n$.
3. Dla każdego niepustego podzbioru $I \subseteq \{1, \dots, m\}$, oblicz $r := \bigcup_{j \in I} s_j^{-1} := \bigcup_{j \in I} s_j \sigma^{-1}$.
4. Dla $i = 1, \dots, m$, zróć:

- (a) Dla każdego niepustego podzbioru $I \subseteq \{1, \dots, m\}$, ustawić

$$x_{ja} := \sigma^{-1} \quad A(y, rI) = e^{-1}.$$

- (b) Ustaw $x_i := \sum_{j \in I} x_{ja}$ (tzn. weź fragment, który wydawał się większością x czasu w poprzednim kroku).

5. Wyjście $x = x_1 \cdots x_n$.

Pozostaje obliczyć prawdopodobieństwo, że A wyprowadza $x = f^1(y)$.

Podobnie jak w dowodzie Twierdzenia 7.14, skupiamy się tylko na n jak w Twierdzeniu 7.18 i zakładamy $y = f(\hat{x})$ dla pewnego $\hat{x} \in S_n$. Każde σ^I reprezentuje „przypuszczenie” wartości $g_I(\hat{x}, s_i)$. Jak zauważono wcześniej, z nieistotnym prawdopodobieństwem wszystkie te domysły są prawidłowe; pokazujemy, że w zależności od tego zdarzenia A wyprowadza $x = \hat{x}$ z prawdopodobieństwem co najmniej $1/2$.

Załóżmy, że $\sigma^I = g_I(\hat{x}, s_i)$ dla wszystkich i . Następnie $\sigma^I = g_I(\hat{x}, rI)$ dla wszystkich I . Ustal indeks $i \in \{1, \dots, n\}$ i rozważ prawdopodobieństwo, że A uzyska poprawną wartość $x_i = \hat{x}_i$. Dla dowolnego niepstego I mamy przynajmniej $\Pr[x_i = \hat{x}_i | \sigma^I] = g_I(\hat{x}, rI) \geq \frac{1}{2}$ z + prawdopodobieństwo $A(\hat{x}, g_I(rI))$ nad wyborem r ; wynika to z faktu, że $\hat{x} \in S_n$ jest równomiernie rozłożone. Zatem dla dowolnego niepstego podzbioru I mamy $\Pr[x_i = \hat{x}_i] = \frac{1}{2} + \varepsilon(n)/2$. Co więcej, $\{x_i = \hat{x}_i\}_{i \in \{1, \dots, n\}}$ są parami niezależne definiuje się jako $\Pr[\{x_i = \hat{x}_i\}_{i \in \{1, \dots, n\}}]$ (sł. w kłębie słówarami), ponieważ $\{r\}$ jest niezależne. Ponieważ x_i wartość, która występuje w wiekszości czas wśród $\{x_i\}_{i \in \{1, \dots, n\}}$, możemy zastosować Twierdzenie A.13, aby otrzymać

$$\begin{aligned} \Pr[x_i = \hat{x}_i] &= \Pr[x_i = \hat{x}_i | \sigma^I] \cdot \Pr[\sigma^I] \\ &= \frac{1}{2} + \frac{\varepsilon(n)/2}{4 \cdot (\varepsilon(n)/2)^2 \cdot (2n/\varepsilon(n))^2} \\ &= \frac{1}{2} + \frac{\varepsilon(n)}{2n}. \end{aligned}$$

Powyższe dotyczy wszystkich i , więc stosując związek sumy, widzimy, że prawdopodobieństwo, że $x_i = \hat{x}_i$ dla jakiegoś i wynosi co najwyżej $1/2$. Oznacza to, że $x_i = \hat{x}_i$ dla wszystkich i (a zatem $x = \hat{x}$) z prawdopodobieństwem co najmniej $1/2$.

Łącząc wszystko w całość: Niech n będzie jak w Twierdzeniu 7.18 i $y = f(\hat{x})$. Z prawdopodobieństwo co najmniej $\varepsilon(n)/2$ mamy $\hat{x} \in S_n$. Wszystkie domysły σ^I są poprawne przynajmniej z prawdopodobieństwem

$$\frac{1}{2} + \frac{1}{2 \cdot (2n/\varepsilon(n))^2 + 1} > \frac{\varepsilon(n)^2}{5n}$$

dla n wystarczająco duże. Uzależniając oba powyższe, A wyprowadza $x = \hat{x}$ z prawdopodobieństwem co najmniej $1/2$. Ogólne prawdopodobieństwo, z jakim A odwraca swoje dane wejściowe, wynosi zatem co najmniej $\varepsilon(n)^3 / 20n = 1/(20np(n))$ dla nieskończonym wielu n . Ponieważ $20np(n)$ jest wielomianem w n , dowodzi to Twierdzenia 7.17. ■

7.4 Konstruowanie generatorów pseudolosowych

Najpierw pokażemy, jak skonstruować generatory pseudolosowe, które rozciągają dane wejściowe o jeden bit, przy założeniu, że istnieją permutacje jednokierunkowe.

Następnie pokażemy, jak rozszerzyć to, aby uzyskać dowolny współczynnik rozszerzalności wielomianu.

7.4.1 Generatory pseudolosowe z minimalnym rozszerzeniem

Niech f będzie jednokierunkową permutacją z twardym predykatem hc . Oznacza to, że $hc(s)$ „wygląda losowo” przy $f(s)$, gdy s jest jednolite. Ponadto, ponieważ f jest permutacją, samo $f(s)$ jest równomiernie rozłożone. (Zastosowanie permutacji do wartości o rozkładzie równomiernie daje wartość o rozkładzie równomiernie.) Zatem jeśli s jest jednolitym ciągiem n -bitowym, ciąg $(n+1)$ -bitowy $f(s)hc(s)$ składa się z jednolitego n -ciąg bitów plus dodatkowy bit, który wygląda jednakowo, nawet w zależności od początkowych n bitów; innymi słowy, ten ciąg $(n + 1)$ -bitowy jest pseudolosowy. Zatem algorytm G zdefiniowany przez $G(s) = f(s)hc(s)$ jest generatorem pseudolosowym.

TWIERDZENIE 7.19 Niech f będzie jednokierunkową permutacją z twardym predykatem hc . Wtedy algorytm G zdefiniowany przez $G(s) = f(s)hc(s)$ jest generatorem pseudolosowym ze współczynnikiem rozszerzalności $(n) = n + 1$.

DOWÓD Niech D będzie probabilistycznym algorytmem wielomianowym w czasie. Udowodnimy, że istnieje funkcja pomijalna taka, że

$$\Pr_{r \in \{0,1\}^{n+1}} \Pr_{s \in \{0,1\}^n} [D(r) = 1] - \Pr_{s \in \{0,1\}^n} [D(G(s)) = 1] \quad \text{negl}(n). \quad (7.3)$$

Podobny argument pokazuje, że istnieje pomijalna funkcja negl dla której

$$\Pr_{s \in \{0,1\}^n} \Pr_{r \in \{0,1\}^{n+1}} [D(G(s)) = 1] - \Pr_{r \in \{0,1\}^{n+1}} [D(r) = 1] \quad \text{negl}(n),$$

co kończy dowód.

Najpierw to zauważ

$$\begin{aligned} \Pr_{r \in \{0,1\}^{n+1}} \Pr_{s \in \{0,1\}^n} [D(r) = 1] &= \Pr_{r \in \{0,1\}^{n+1}} \Pr_{s \in \{0,1\}^n} [D(f(s)r) = 1] \\ &= \Pr_{n,r \in \{0,1\}^n} [D(f(s)r) = 1] s \in \{0,1\} \\ &= \frac{1}{s \in \{0,1\}^n} \cdot \Pr_{r \in \{0,1\}^{n+1}} [D(f(s)hc(s)) = 1] \\ &\quad + \frac{1}{s \in \{0,1\}^n} \cdot \Pr_{r \in \{0,1\}^{n+1}} [D(f(s)hc(s)) = 1], \end{aligned}$$

wykorzystując fakt, że f jest permutacją drugiej równości i że jednolity bit r jest równy $hc(s)$ z prawdopodobieństwem dokładnie $1/2$ dla trzeciej równości. Od

$$\Pr_{s \in \{0,1\}^n} \Pr_{r \in \{0,1\}^{n+1}} [D(G(s)) = 1] = \Pr_{s \in \{0,1\}^n} [D(f(s)hc(s)) = 1]$$

(z definicji G), oznacza to, że równanie (7.3) jest równoważne

$$\frac{1}{2} \cdot \Pr_{s \in \{0,1\}^n} \Pr_{r \in \{0,1\}^{n+1}} [D(f(s)hc(s)) = 1] - \Pr_{s \in \{0,1\}^n} [D(f(s)hc(s)) = 1] \quad \text{negl}(n).$$

Rozważmy następujący algorytm A, któremu na wejściu podana jest wartość $y = f(s)$ i który próbuje przewidzieć wartość $hc(s)$:

1. Wybierz jednolity $r \in \{0, 1\}$.

2. Uruchom D(roku). Jeśli D wyprowadza 0, wyprowadza r; w przeciwnym razie wypisz $\neg r$.

Oczywiście A przebiega w czasie wielomianowym. Z definicji A mamy

$$\begin{aligned} & \Pr_{s \in \{0,1\}^n} [A(f(s)) = hc(s)] \\ &= \frac{1}{n+1} \cdot \Pr_{s \in \{0,1\}^n} [A(f(s)) = hc(s) \mid r = hc(s)] \quad s \in \{0,1\}^n \\ &= \frac{1}{n+1} \cdot \Pr_{s \in \{0,1\}^n} [\neg r + 2 \cdot [A(f(s)) = hc(s) \mid r = hc(s)]] \\ &= \frac{1}{2} \cdot \Pr_{s \in \{0,1\}^n} [\Pr_{f \in \{0,1\}^n} [D(f(s)hc(s)) = 0] + Par] \quad \overline{[D(f(s)hc(s)) = 1]} \\ &= \frac{1}{2} \cdot \frac{1}{n+1} \Pr_{s \in \{0,1\}^n} [\Pr_{f \in \{0,1\}^n} [D(f(s)hc(s)) = 1] + Par] \quad \overline{[D(f(s)hc(s)) = 1]} \\ &= \frac{1}{2} + \frac{1}{2} \cdot \Pr_{s \in \{0,1\}^n} [\Pr_{f \in \{0,1\}^n} [D(f(s)hc(s)) = 1] - \Pr_{f \in \{0,1\}^n} [D(f(s)hc(s)) = 1]] \cdot s \in \{0,1\}^n \end{aligned}$$

Ponieważ hc jest twardym predykatem f , wynika z tego, że istnieje pomijalna funkcja negl, dla której

$$\frac{1}{2} \cdot \Pr_{s \in \{0,1\}^n} [\Pr_{f \in \{0,1\}^n} [D(f(s)hc(s)) = 1] - \Pr_{f \in \{0,1\}^n} [D(f(s)hc(s)) = 1]] = negl(n),$$

zgodnie z życzeniem. ■

7.4.2 Zwiększanie współczynnika rozszerzenia

Pokazujemy teraz, że współczynnik rozszerzalności generatora pseudolosowego można zwiększyć o dowolną żądaną wielkość (wielomian). Oznacza to, że poprzednia konstrukcja, ze współczynnikiem rozwinięcia $cia(n) = n + 1$, wystarczy do zbudowania generatora pseudolosowego z dowolnym (wielomianowym) współczynnikiem rozwinięcia cia .

TWIERDZENIE 7.20 Jeżeli istnieje generator pseudolosowy G o współczynniku rozwinięcia $cia(n + 1)$, to dla dowolnego wielomianu istnieje generator pseudolosowy G^* ze współczynnikiem rozwinięcia $cia(poli(n))$.

DOWÓD Najpierw rozważamy skonstruowanie generatora pseudolosowego G^* , który generuje $n + 2$ bity. G^* działa w następujący sposób: Mając początkowe ziarno $s \in \{0, 1\}^n$, it

oblicza $t_1 := G(s)$, aby otrzymać $n + 1$ bitów pseudolosowych. Początkowych n bitów t_1 używa się następnie ponownie jako zarodek G ; wynikowe $n+1$ bitów, połączone z ostatnim bitem t_1 , daje wynik $(n + 2)$ -bitowy. (Patrz rysunek 7.1.) Drugie zastosowanie G wykorzystuje ziarno pseudolosowe, a nie losowe. Dowód bezpieczeństwa, który podajemy poniżej, pokazuje, że nie ma to wpływu na pseudolosowość wyniku.

Udowodnimy teraz, że G^* jest generatorem pseudolosowym. Zdefiniuj trzy ciągi rozkładów $\{H_0^n\}_{n=1}^\infty, \{H_1^n\}_{n=1}^\infty, \{H_2^n\}_{n=1}^\infty$, gdzie każdy z $H_0^n H_1^n$ i H_2^n jest rozkładem na strunach o długości $n + 2$. W rozkładzie wybiera się H_0^n . A jednolity $t_0 \in \{0, 1\}^n$ i H_1^{n+1} a wynikiem jest $G^*(t_0)$. W dystrybucji wybierany jest ciąg analizowany jak w rozkładzie $G(s)$. Jednolity $t_1 \in \{0, 1\}^n$ to początkowe n bitów t_1 , a s_1 to bit

W rozkładzie H_2^n wynikiem jest jednolity ciąg znaków $t_2 \in \{0, 1\}^{n+2}$. Przez $t_2 \in H_2^n$ oznaczamy proces generowania ($n + 2$)-bitowego ciągu t_2 zgodnie z rozkładem H_2^n .

Napraw dowolny probabilistyczny wyróżnik wielomianu-czasu D . Najpierw twierdzimy, że istnieje pomijalna funkcja negl taka, że

$$\Pr_{t_2 \in H_0^n} [D(t_2) = 1] = \Pr_{t_2 \in H_1^n} [D(t_2) = 1] = \text{negl}(n). \quad (7.4)$$

Aby to zobaczyć, rozważmy wyróżnik wielomianu-czasu D , który na wejściu $t_1 \in \{0, 1\}^{n+1}$ analizuje t_1 jako $s_1 s_1$ z $|s_1| = n$, oblicza $t_2 := G(s_1)s_1$ i wyprowadza $D(t_2)$. Oczywiście D przebiega w czasie wielomianowym. Obseruj to:

- Jeśli t_1 jest jednorodne, rozkład na t_2 generowany przez D jest dokładnie taki rozkładu H_1^n . Zatem,

$$\Pr_{t_1 \in \{0,1\}^{n+1}} [D(t_1) = 1] = \Pr_{t_2 \in H_1^n} [D(t_2) = 1].$$

- Jeśli $t_1 = G(s)$ dla jednorodności $s \in \{0, 1\}^n$, rozkład na t_2 wygenerowany przez D jest dokładnie rozkładem H_0^n . To jest,

$$\Pr_{s \in \{0,1\}^n} [D(G(s)) = 1] = \Pr_{t_2 \in H_0^n} [D(t_2) = 1].$$

Pseudolosowość G implikuje, że istnieje pomijalna funkcja negl with

$$\Pr_{s \in \{0,1\}^n} [D(G(s)) = 1] = \Pr_{t_1 \in \{0,1\}^{n+1}} [D(t_1) = 1] = \text{negl}(n).$$

Poniżej znajduje się równanie (7.4).

Następnie ponownie twierdzimy, że istnieje funkcja pomijalna taka, że

$$\Pr_{t_2 \in H_1^n} [D(t_2) = 1] = \Pr_{t_2 \in H_2^n} [D(t_2) = 1] = \text{negl}(n). \quad (7.5)$$

Aby to zobaczyć, rozważmy wyróżnik wielomianu-czasu D, który na wejściu $w \in \{0, 1\}^{n+1}$ wybiera jednorodne $\sigma \in \{0, 1\}$, ustawia $t_2 := w\sigma$ i wyrowadza $D(t_2)$. Jeśli w jest jednolite, to t_2 również; zatem,

$$\Pr_{w \in \{0,1\}^{n+1}} [D(w) = 1] = \text{Par}_{t_2 \in H_{2,n}} [D(t_2) = 1].$$

Z drugiej strony, jeśli $w = G(s)$ dla uniforma $s \in \{0, 1\}^n$ dokładnie, następnie rozkłada się t_2 według H_1 i tak

$$\Pr_{s \in \{0,1\}^n} [D(G(s)) = 1] = \text{Par}_{t_2 \in H_{1,n}} [D(t_2) = 1].$$

Tak jak poprzednio, pseudolosowość G implikuje równanie (7.5).

Łącząc wszystko, mamy

$$\begin{aligned} \Pr_{s \in \{0,1\}^n} [D(G(s)) = 1] &= \Pr_{r \in \{0,1\}^{n+2}} [D(r) = 1] \\ &= \Pr_{t_2 \in H_{0,n}} [D(t_2) = 1] - \text{Par}_{t_2 \in H_{2,n}} [D(t_2) = 1] \\ &\quad \Pr_{t_2 \in H_{1,n}} [D(t_2) = 1] - \text{Par}_{t_2 \in H_{2,n}} [D(t_2) = 1] \\ &\quad + \Pr_{t_2 \in H_{1,n}} [D(t_2) = 1] - \text{Par}_{t_2 \in H_{2,n}} [D(t_2) = 1] \\ &\quad \text{negl}(n) + \text{negl}(n), \end{aligned} \tag{7.6}$$

używając równań (7.4) i (7.5). Ponieważ D jest arbitralnym wyróżnikiem czasu wielomianowego, dowodzi to, że G^\wedge jest generatorem pseudolosowym.

Sprawa ogólna. Ten sam pomysł co powyżej można zastosować iteracyjnie w celu wygenerowania dowolnej liczby bitów pseudolosowych. Formalnie założymy, że chcemy skonstruować generator pseudolosowy G^\wedge ze współczynnikiem rozwinięcia $n + p(n)$ dla pewnego wielomianu p . Na wejściu $s \in \{0, 1\}^n$ algorytm G^\wedge wykonuje (por. rysunek 7.1):

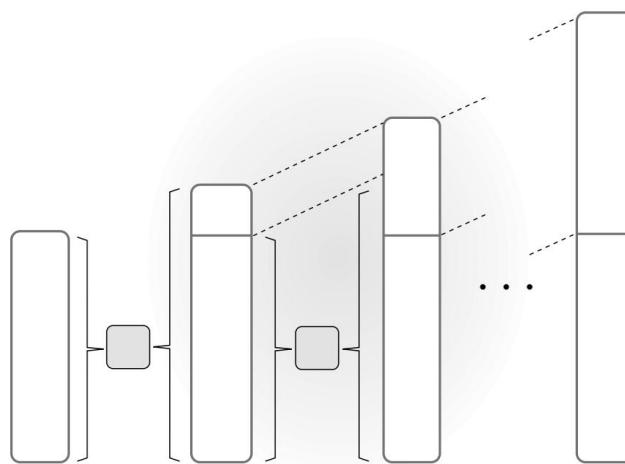
1. Ustaw $t_0 := s$. Dla $i = 1, \dots, p(n)$ wykonaj:

- (a) Niech σ_i 1 bęź dzie pierwszymi n bitami t_i 1 i niech σ_i 1 oznacza pozostałe 1 bity. (Gdy $i = 1$, $s_0 = t_0$ i σ_0 jest pustym ciągiem znaków.) (b) Ustaw $t_i :=$

$G(s_0 \dots s_{i-1})\sigma_i$ 1.

2. Wyjście $t_p(n)$.

Pokazujemy, że G^\wedge jest generatorem pseudolosowym. Dowód wykorzystuje popularną technikę zwaną argumentem hybrydowym. (Właściwie nawet w przypadku $p(n) = 2$ powyżej zastosowano prosty argument hybrydowy.) Główna różnica w porównaniu z poprzednim dowodem ma charakter techniczny. Wcześniej mogliśmy zdefiniować i jawnie pracować z trzema sekwencjami rozkładów $\{H_0\}$, $\{H_1\}$ i $\{H_2\}$. Tutaj nie jest to możliwe, ponieważ liczba rozkładów do rozważenia rośnie wraz z n .



RYSUNEK 7.1: Zwięk kszanie ekspansji generatora pseudolosowego.

Dla dowolnych $n \in \mathbb{N}$, $j \in \{0, 1\}^{p(n)}$ niech H_j będzie rozkładem długości na ciągach, a G^j zaczynając od $n+p(n)$ zdefiniowanego następująco: wybierz $s \in \{0, 1\}^n$, następnie uruchom $t \in \{0, 1\}^{n+j}$ z iteracją $j+1$ i wynik $tp(n)$. (Kiedy $j = p(n)$ oznacza to, że po prostu wybieramy uniform $tp(n) \in \{0, 1\}^{n+p(n)}$ i wyprowadzamy go.) Kluczowa obserwacja jest to, że H_0^n odpowiada wyprowadzaniu $G^j(s)$ dla uniforma $s \in \{0, 1\}^n$, podczas gdy $H_p(n)$ odpowiada wysyłaniu jednolitego $(n + p(n))$ -bitowego ciągu. Oznacza to, że naprawiamy dowolny H wyróżnik wielomianu-czasu D

$$\begin{aligned} & \Pr_{s \in \{0,1\}^n} [D(G^j(s)) = 1] = \Pr_{r \in \{0,1\}^{n+p(n)}} [D(r) = 1] \\ & = \Pr_{t \in H_0^n} [D(t) = 1] - \Pr_{t \in H_p(n)} [D(t) = 1]. \end{aligned} \quad (7.7)$$

Udowodnimy, że powyższe jest nieistotne, stąd G^j jest generatorem pseudolosowym.

Napraw D jak powyżej i rozważ wyróżnik D , który jako dane wejściowe wykonuje następujące czynności:

1. Wybierz jednolity $j \in \{1, \dots, p(n)\}$.
2. Wybierz jednolity串 $\sigma \in \{0, 1\}^{j-1}$. (Kiedy $j = 1$, to σ_j jest pusto)

putę $tp(n) \in \{0, 1\}^{n+p(n)}$. Następnie uruchom G^j zaczynając od iteracji $j+1$ do com-3. Zbiór $t_j :=$ wó . Wyjście $D(tp(n))$.

Oczywiście D przebiega w czasie wielomianowym. Analiza zachowania D jest bardziej skomplikowana niż wcześniej, chociaż podstawowe idee są takie same. Napraw nr

i powiedzmy, że D wybiera $j = j_{\text{def}}$. Jeśli w jest jednorodne, to tj j jest jednorodne i tak dystrybuje na $t = tp(n)$ jest dokładne rozkładem H_N . To jest,

$$\Pr_w \left[D(w) = 1 \mid j = j_{\text{def}} \right] = \Pr_{j \sim t \sim H_N} [D(t) = 1].$$

Ponieważ każda wartość j jest wybierana z równym prawdopodobieństwem,

$$\begin{aligned} \Pr_{p(n)} \left[D(w) = 1 \right] &= \frac{1}{n} \cdot \Pr_{j=1}^{p(n)} \left[D(w) = 1 \mid j = j_{\text{def}} \right] \\ &= \frac{1}{p(n)} \cdot \Pr_{j=1}^{p(n)} \left[D(t) = 1 \right]. \end{aligned} \quad (7.8)$$

strony, powiedzmy, że D wybiera $j = js \in \{0, 1\}^n$. i $w = G(s)$ dla uniformu. Z drugiej Definicja tj 1 jest jednorodna, więc $s \in \{0, 1\}^n$ jest jednorodna i D jest równoważny uruchomieniu G^* z iteracją $j = tp(n)$ jest teraz dokładne takie, jak $\def{d}{\text{def}}$ oblicz $tp(n)$. Oznacza to, że rozkład na t rozkład $H_j = 1$

a więc c

$$\Pr_s \left[D(G(s)) = 1 \mid j = j_{\text{def}} \right] = \Pr_{t \sim j \sim t \sim H} [D(t) = 1].$$

Dlatego,

$$\begin{aligned} \Pr_s \left[D(G^*(s)) = 1 \right] &= p(n) \cdot \frac{1}{n} \cdot \Pr_{j=1}^{p(n)} \left[D(G(s)) = 1 \mid j = j_{\text{def}} \right] \\ &= \frac{1}{p(n)} \cdot \Pr_{j=1}^{p(n)} \left[D(t) = 1 \right] \\ &= \frac{1}{p(n)} \cdot \Pr_{j=0}^{p(n)-1} \left[D(t) = 1 \right]. \end{aligned} \quad (7.9)$$

Mogimy teraz przeanalizować, jak dobrze D odróżnia wyniki G od losowych:

$$\begin{aligned} \Pr_s \left[D(G(s)) = 1 \right] - \Pr_w [D(w) = 1] &= \Pr_{t \sim H} [D(t) = 1] - \Pr_{t \sim H_N} [D(t) = 1] \\ &= \frac{1}{p(n)} \cdot \Pr_{j=0}^{p(n)-1} [D(t) = 1] - \Pr_{j=1}^{p(n)} [D(t) = 1] \\ &= \frac{1}{p(n)} \cdot \Pr_{t \sim H_0} [D(t) = 1] - \Pr_{t \sim H_N} [D(t) = 1], \end{aligned} \quad (7.10)$$

opierając się na równaniach (7.8) i (7.9) dla pierwszej równości. (Druga równość zachodzi, ponieważ w każdej sumie zawarte są te same wyrazy, z wyjątkiem pierwszego wyrazu lewej sumy i ostatniego wyrazu prawej sumy). Ponieważ G jest generatorem pseudolosowym, człon po lewej stronie równania (7.10) jest zaniedbywalny; ponieważ p jest wielomianem, oznacza to, że równanie (7.7) jest pomalne, co kończy dowód, że G^{\wedge} jest generatorem pseudolosowym. ■

Kładąc wszystko razem. Niech f będzie permutacją jednokierunkową. Biorąc generator pseudolosowy o współczynniku rozwinięcia $n + 1$ z Twierdzenia 7.19 i zwiększając współczynnik rozwinięcia do $n + 1$ stosując podejście z dowodu Twierdzenia 7.20, otrzymujemy następujący generator pseudolosowy G^{\wedge} :

$$G^{\wedge}(s) = f(0)(s) \text{hc}(f(-1)(s)) \cdots \text{hc}(s),$$

gdzie $f(i)(s)$ odnosi się do i -krotnej iteracji f . Należy zauważać, że G^{\wedge} wykorzystuje oceny f i generuje jeden bit pseudolosowy na każdą ocenę, używając twardego predykatu hc .

Połączenie z szyframi strumieniowymi. Przypomnijmy z sekcji 3.3.1, że szyfr strumieniowy (bez IV) jest definiowany przez algorytmy (Init, GetBits), gdzie Init pobiera ziarno $s \in \{0, 1\}^n$ i zwraca stan początkowy st , a GetBits przyjmuje jako dane wejściowe stan bieżący st i wyprowadza bit σ oraz zaktualizowany stan st . Konstrukcja G^{\wedge} z poprzedniego dowodu dobrze pasuje do tego paradrymatu: przyjmij Init jako trywialny algorytm generujący $st = s$ i zdefiniuj GetBits(st), aby obliczyć $G(st)$, przeanalizuj wynik jako stozę pomocą $|st| = n$, i wyprowadź bit σ zaktualizowany stan st . (Jeśli użyjemy tego szyfru strumieniowego do wygenerowania bitów wyjściowych $p(n)$, zaczynając od materiału siewnego s , wówczas otrzymamy dokładne końcowe bity $p(n) G^{\wedge}(s)$ w odwrotnej kolejności.) Powyższy dowód pokazuje, że daje to pseudolosowość generatora.

Argumenty hybrydowe. Argument hybrydowy jest podstawowym narzędziem do udowadniania nierozróżnialności, gdy podstawowy element podstawowy jest stosowany (lub kilka różnych elementów podstawowych) wielokrotnie. W sposób nieformalny technika ta polega na zdefiniowaniu szeregu pośrednich „rozkładów hybrydowych”, które stanowią pomoc pomimo dźwiga dwoma „rozkładami ekstremalnymi”, których nierozróżnialność chcemy udowodnić. (W powyższym dowodzie te ekstremalne rozkłady odpowiadają wynikowi G^{\wedge} i losowemu ciągowi znaków.) Aby zastosować technikę dowodu, powinny zostać spełnione trzy warunki. Po pierwsze, rozkłady ekstremalne powinny odpowiadać oryginalnym przypadkom będącym przedmiotem gry (W powyższym dowodzie H_{0N} zainteresowania. był równy rozkładowi wywołanemu przez G^{\wedge} , podczas $H_N^{p(n)}$). Po drugie, musi istnieć możliwość przełożenia możliwości rozróżniania kolejnych rozkładów hybrydowych na złamanie pewnych podstawowych założeń. (Powyżej zasadniczo pokazaliśmy, że odróżnienie H_i od H_{i+1} było równoznaczne z odróżnieniem wyniku G od losowego.) Wreszcie liczba rozkładów hybrydowych powinna być wielomianowa.

Zobacz także Twierdzenie 7.32.

7.5 Konstruowanie funkcji pseudolosowych

Pokażemy teraz, jak skonstruować funkcję pseudolosową z dowolnego generatora pseudolosowego (podważającego długość). Przypomnijmy, że funkcja pseudolosowa to wydajnie obliczalna funkcja F z kluczem, której nie można odróżnić od funkcji prawdziwie losowej w sensie opisany w podrozdziale 3.5.1. Dla uproszczenia ograniczamy tutaj naszą uwagę do przypadku, w którym F zachowuje długość, co oznacza, że dla $k \in \{0, 1\}^n$ funkcja F_k odwzorowuje n -bitowe wejścia na n -bitowe wyjścia.

Funkcję pseudolosową (zachowującą długość) można nieformalnie postrzegać jako generator pseudolosowy ze współczynnikiem rozszerzenia $n \cdot 2^n$; mając taki generator pseudolosowy G , moglibyśmy zdefiniować $F_k(i)$ (dla $0 \leq i < 2^n$) jako i -ty n -bitowy blok $G(k)$. Powodem, dla którego to nie działa, jest to, że F musi być efektywnie obliczone; bloków jest wykładniczo wiele i potrzebujemy sposobu obliczenia i -tego bloku bez konieczności obliczania wszystkich pozostałych bloków.

Zrobimy to, obliczając „bloki” wyników, schodząc po drzewie binarnym. Przykładową konstrukcję przedstawiamy najpierw pokazując funkcję pseudolosową pobierającą 2-bitowe dane wejściowe. Niech G będzie generatorem pseudolosowym ze współczynnikiem rozszerzenia $2n$. Jeśli użyjemy G jak w dowodzie Twierdzenia 7.20, możemy otrzymać generator pseudolosowy G^\wedge ze współczynnikiem rozszerzenia $4n$, który wykorzystuje trzy wywołania G . (Za każdym razem, gdy zastosowane zostanie G , wytwarzamy n dodatkowych bitów pseudolosowych.) Jeśli zdefiniujemy $F(i)$ (gdzie $0 \leq i < 4$ oraz i jest zakodowane jako 2-bitowy ciąg binarny) jako i -ty blok $G^\wedge(k)$, wówczas obliczenie F wymaga obliczenia G (3) miałyby całego G^\wedge , a zatem trzech wywołań G . Pokazujemy, jak to zrobić skonstruując funkcję pseudolosową F , używając tylko dwóch wywołań G na dowolnym wejściu.

Niech G_0 i G_1 będą funkcjami oznaczającymi pierwszą i drugą połowę wyniku G ; tj. $G(k) = G_0(k)G_1(k)$ gdzie $|G_0(k)| = |G_1(k)| = |k|$. Zdefiniuj F w następujący sposób:

$$F_k(00) = G_0(G_0(k)) \quad F_k(10) = G_0(G_1(k))$$

$$F_k(01) = G_1(G_0(k)) \quad F_k(11) = G_1(G_1(k)).$$

Twierdzimy, że powyższe cztery ciągi są pseudolosowe, nawet gdy patrzymy na nie razem. (To wystarczy, aby udowodnić, że F jest pseudolosowe.) Intuicyjnie dzieje się tak, ponieważ $G_0(k)G_1(k) = G(k)$ jest pseudolosowe i stąd nie do odróżnienia od jednolitego $2n$ -bitowego ciągu $k0k1$. Ale wtedy

$$G_0(G_0(k))G_1(G_0(k))G_0(G_1(k))G_1(G_1(k))$$

jest nie do odróżnienia

$$G_0(k0)G_1(k0)G_0(k1)G_1(k1) = G(k0)G(k1).$$

Ponieważ G jest generatorem pseudolosowym, powyższe jest nie do odróżnienia od jednolitego $4n$ -bitowego ciągu. Dowód formalny wykorzystuje argument hybrydowy.

Uogólniając ten pomysł, możemy otrzymać funkcję pseudolosową na n-bitach wejścia poprzez zdefiniowanie

$$F_k(x) = G_{xn}(\dots G_{x1}(k)\dots),$$

gdzie $x = x_1 \dots x_n$; patrz Konstrukcja 7.21. Intuicja wyjaśniająca, dlaczego ta funkcja jest pseudolosowa, jest taka sama jak poprzednio, ale formalny dowód komplikuje fakt, że obecnie należy wziąć pod uwagę wykładniczo wiele danych wejściowych.

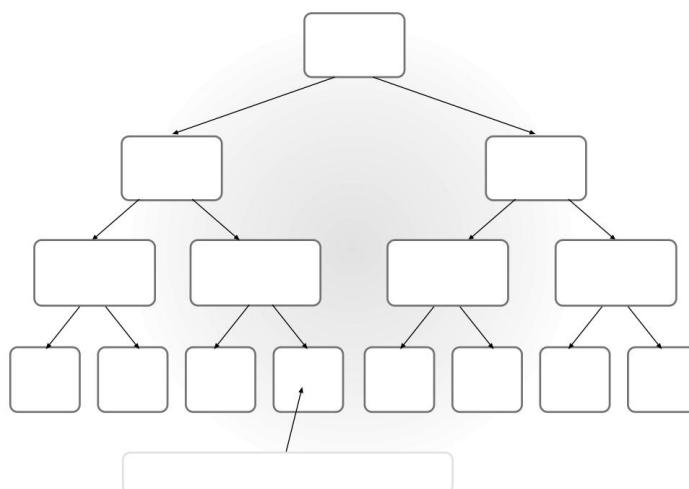
BUDOWA 7.21

Niech G będzie generatorem pseudolosowym o współczynniku rozwinęciu $(n) = 2n$ i zdefiniuj G_0, G_1 jak w tekście. Dla $k \in \{0, 1\}^n$ zdefiniuj funkcję $F_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$ jak:

$$F_k(x_1 x_2 \dots x_n) = G_{xn}(\dots (G_{x2}(G_{x1}(k)))\dots).$$

Funkcja pseudolosowa z generatora pseudolosowego.

Przydatne jest spojrzenie na tę konstrukcję jako definiującą dla każdego klucza $k \in \{0, 1\}^n$ kompletne drzewo binarne o głębi bokosći n , w którym każdy węzeł zawiera n-bitową wartość. (Patrz rysunek 7.2, gdzie $n = 3$.) Pierwiastek ma wartość k i dla każdego węzła zła wewnętrznego o wartości k jego lewe dziecko ma wartość $G_0(k)$, a jego prawe dziecko ma wartość $G_1(k)$. Wynik $F_k(x)$ dla $x = x_1 \dots x_n$ definiuje się następująco jako wartość w węźle liściu osiągnięty tą w wyniku przechodzenia przez drzewo zgodnie z bitami x , gdzie $x_i = 0$ oznacza „idź w lewo”, a $x_i = 1$ oznacza „idź w prawo”. (Funkcja jest zdefiniowana tylko dla danych wejściowych o długości n , a zatem wyprowadzane są tylko wartości na liściach.) Rozmiar drzewa jest wykładniczy w n . Niemniej jednak, aby obliczyć $F_k(x)$, nie trzeba konstruować ani przechowywać całego drzewa; potrzebnych jest tylko n ocen G .



RYSUNEK 7.2: Konstruowanie funkcji pseudolosowej.

TWIERDZENIE 7.22 Jeżeli G jest generatorem pseudolosowym ze współczynnikiem rozwinięcia $t(n) = 2n$, to Konstrukcja 7.21 jest funkcją pseudolosową.

DOWÓD Najpierw pokazujemy, że dla dowolnego wielomianu t nie jest możliwe odróżnienie $t(n)$ jednolitych $2n$ -bitowych ciągów od $t(n)$ ciągów pseudolosowych; tj. dla dowolnego wielomianu t i dowolnego algorytmu ppt A poniższe jest nieistotne:

$$\Pr A r_1 \cdots r_{t(n)} = 1 \quad \Pr AG(s_1) \cdots G(st(n)) = 1$$

gdzie pierwsze prawdopodobieństwo dotyczy jednolitego wyboru $r_1, \dots, r_{t(n)} \in \{0, 1\}^{2n}$, a drugie prawdopodobieństwo dotyczy jednolitego wyboru $s_1, \dots, st(n) \in \{0, 1\}^{2n}$.

Dowód odbywa się za pomocą argumentu hybrydowego. Napraw wielomian t i algorytm ppt rytm A i rozważ następujący algorytm A :

Wyróżnik A : A na wejściu podaje się串 znaków $w \in \{0, 1\}^{2n}$.

1. Wybierz jednolity $j \in \{1, \dots, t(n)\}$.
2. Wybierz jednolite, niezależne wartości $r_1, \dots, r_{j-1} \in \{0, 1\}^n$ i $s_{j+1}, \dots, st(n) \in \{0, 1\}^n$.
3. Wyjście $A r_1 \cdots r_{j-1} w G(s_{j+1}) \cdots G(st(n))$.

Dla dowolnych n oraz $0 \leq i \leq t(n)$ niech G_i oznacza rozkład na strunach o długości $2n \cdot t(n)$, w którym pierwsze i „bloki” o długości $2n$ są jednorodne, a $t(n) - i$ pozostałe $t(n) - i$ bloki są pseudolosowe. Należy zauważyć, że G n odpowiada rozkładowi, w którym wszystkie bloki $t(n)$ są jednorodne, natomiast G_0 odpowiada rozkładowi, w którym wszystkie bloki $t(n)$ są pseudolosowe. To jest,

$$\begin{aligned} & \Pr_{y \in G_n^{t(n)}} [A(y) = 1] = \Pr_y [A(y) = 1] \\ &= \Pr A r_1 \cdots r_{t(n)} = 1 \quad \Pr AG(s_1) \cdots G(st(n)) = 1 \end{aligned} \tag{7.11}$$

Powiedzmy, że A wybiera $j = j$. Jeśli jego wejście w jest jednolitym ciągiem $2n$ -bitowym, wówczas uruchamiane jest A na wejściu rozłożonym według G_j uniform s, następnie $A \in G_j$. Jeśli z drugiej strony $w = G(s)$ dla s . Ten jest uruchamiany na wejściu rozłożonym według G_j dla s . Oznacza to, że

$$r \Pr_{r \in \{0,1\}^{2n}} [A(r) = 1] = \frac{1}{t(n)} \cdot \prod_{j=1}^{t(n)} \Pr_{y \in G_j} [A(y) = 1]$$

I

$$\Pr_{s \in \{0,1\}^n} [A(G(s)) = 1] = \frac{1}{t(n)} \cdot \prod_{j=0}^{t(n)-1} \Pr_{G_j} [A(y) = 1].$$

Dlatego,

$$\begin{aligned} & \Pr_{\substack{r \in \{0,1\}^{2n} \\ s \in \{0,1\}^n}} [A(r) = 1] - \Pr_{\substack{f \text{ funkcja} \\ t(n)}} [A(G(s)) = 1] \\ &= \frac{1}{t(n)} \cdot \Pr_{\substack{y \in \{0,1\}^n \\ G \text{ funkcja}}} [A(y) = 1] - \Pr_{\substack{y \in \{0,1\}^n \\ A \text{ funkcja}}} [A(y) = 1]. \end{aligned} \quad (7.12)$$

Ponieważ G jest generatorem pseudolosowym, a A działa w czasie wielomianowym, wiemy, że lewa strona równania (7.12) musi być zaniedbywalna; ponieważ $t(n)$ jest wielomianem, oznacza to, że lewa strona równania (7.11) jest również zaniedbywalna.

Wracając do sedna dowodu, pokażemy teraz, że F , jak w konstrukcji 7.21, jest funkcją pseudolosową. Niech D bęź dzie dowolnym wyróżnikiem ppt , któremu na wejściu podano $1n$. Pokazujemy, że D nie jest w stanie rozróżnić przypadku, gdy ma dostęp p w Oracle do funkcji równej F_k dla jednorodnego k lub funkcji wybranej jednostajnie z Funcn . (Zobacz sekcję 3.5.1.) Aby to zrobić, używamy innego argumentu hybrydowego. Tutaj definiujemy sekwencję rozkładów wartości na liściach pełnego drzewa binarnego o głębi bokości n . Wiąząc każdy liść z ciągiem o długości n , jak w Konstrukcji 7.21, możemy równoważnie postrzegać je jako rozkłady po funkcjach odwzorowujących n -bitowe wejścia na n -bitowe wyjścia. Dla dowolnych n i $0 \leq i \leq n$, niech H_i bęź dzie następ pującym rozkładem po wartościach na liściach drzewa binarnego o głębi bokości n : najpierw wybierz wartości dla węzłów na poziomie i i niezależnie i równomiernie z $\{0, 1\}^n$. Następnie dla każdego węzła na poziomie i lub niższym o wartości k jego lewemu potomkowi przypisuje się wartość $G_0(k)$, a prawemu potomkowi przypisuje się wartość $G_1(k)$. Należy zauważać, że H_n odpowiada rozkładowi, w którym wszystkie wartości na liściach są wybierane równomiernie i niezależnie, a zatem odpowiada wyborowi jednolitej funkcji z Funcn , podczas gdy H_0 odpowiada wyborowi jednolitego klucza k w Konstrukcji 7.21, ponieważ tylko w tym przypadku pierwiastek (na poziomie 0) jest wybierany jednolicie. To jest,

$$\begin{aligned} & \Pr_{\substack{k \in \{0,1\}^n \\ f \text{ funkcja}}} [DF_k(\cdot)(1n) = 1] - \Pr_{\substack{f \text{ funkcja} \\ H_n}} [Df(\cdot)(1n) = 1] \\ &= \Pr_{\substack{f \in H_0 \\ f \text{ funkcja}}} [Df(\cdot)(1n) = 1] - \Pr_{\substack{f \in H_n \\ f \text{ funkcja}}} [Df(\cdot)(1n) = 1]. \end{aligned} \quad (7.13)$$

Pokazujemy, że równanie (7.13) jest nieistotne, co kończy dowód.

Niech $t = t(n)$ bęź dzie górną granicą wielomianu liczby zapytań, które D wykonuje do swojej wyczyni na wejściu $1n$. Zdefiniuj wyróżnik A , który próbuje rozróżnić $t(n)$ jednolitych $2n$ -bitowych ciągów od $t(n)$ ciągów pseudolosowych w następujący sposób:

Wyróżnik A : A jest

podany na wejściu $2n \cdot t(n)$ -bitowy串 $w_1 \dots w_{t(n)}$.

1. Wybierz jednolity $j \in \{0, \dots, n-1\}$. Poniżej A (domyślnie) utrzymuje drzewo binarne o głębi bokości n z n -bitowymi wartościami w (podzbiorze) wewnętrznych węzłów na głębi bokości $j+1$ i poniżej.

2. Uruchom D(1n). Kiedy D wysyła zapytanie do Oracle $x = x_1 \dots x_n$, spójrz na przedrostek $x_1 \dots x_j$. Są dwa przypadki: • Jeśli

D nigdy wcześniej nie wysyłał zapytań z tym przedrostkiem, użyj $x_1 \dots x_j$, aby dotrzeć do węzła w na j-tym poziomie drzewa.
Weź następujący nieużywany 2n-bitowy ciąg w i ustaw wartość lewego dziecka węzła w na lewą połowę w, a wartość prawego dziecka w na prawą połowę w. • Jeżeli D wykonał wcześniej zapytanie z prefiksem $x_1 \dots x_j$, to węzeł $x_1 \dots x_{j+1}$ ma już przypisaną wartość.

Używając wartości w węzle $x_1 \dots x_{j+1}$, oblicz wartość na liściu odpowiadającą $x_1 \dots x_n$ jak w Konstrukcji 7.21 i zwróć tę wartość do D.

3. Po zakończeniu wykonywania D wyprowadź bit zwrócony przez D.

A przebiega w czasie wielomianowym. Ważne jest tutaj, aby A nie musiał przechowywać całego drzewa binarnego o rozmiarze wykładniczym. Zamiast tego „uzupełnia” wartości co najwyżej $2t(n)$ węzłów w drzewie. Powiedzmy, że A wybiera $j = j$. Obseruj to:

1. Jeśli na wejściu A jest jednolity ciąg znaków o długości $2n \cdot t(n)$ -bitów, to odpowiedzi udzielone D rozkładają się dokładnie tak, jak gdyby D oddziaływał z funkcją wybraną z rozkładu $H_j + 1$. Dzieje się tak, ponieważ wartości węzłów na poziomie $j + 1$ z drzewa są jednolite i niezależne.
2. Jeśli dane wejściowe A składają się z $t(n)$ ciągów pseudolosowych —tj. $w_i = G(s_i)$ dla jednorodnego nasienia s_i —wówczas odpowiedzi udzielone D są dystrybuowane dokładnie tak, jakby D oddziaływał z funkcją wybraną z rozkładu H_j . To zachodzi, ponieważ wartości węzłów na poziomie j z drzewa (mianowicie wartości s) są jednolite i niezależne. (Te wartości s są nieznane A, ale to nie robi różnic.)

Postępując jak poprzednio, można to wykazać

$$\begin{aligned} \Pr[A \cdot r_1 \dots r_t(n) = 1] &= \Pr[A \cdot G(s_1) \dots G(s_t(n)) = 1] \\ &= \frac{1}{N} \cdot \Pr_f \left[\bigwedge_{H_{Q_n}} [Df(\cdot)(1n) = 1] \right] - \Pr_f \left[\bigwedge_{H_{N_n}} [Df(\cdot)(1n) = 1] \right]. \end{aligned} \quad (7.14)$$

Pokazaliśmy wcześniej, że równanie (7.14) musi być pomijalne. Z powyższego wynika zatem, że równanie (7.13) również musi być nieistotne. ■

7.6 Konstruowanie (silnych) permutacji pseudolosowych

Następnie pokażemy, jak permutacje pseudolosowe i silne permutacje pseudolosowe można skonstruować z dowolnej funkcji pseudolosowej. Przypomnienie sobie czegoś

z sekcji 3.5.1 wynika, że permutacja pseudolosowa jest funkcją pseudolosową, która jest również skutecznie odwracalna, podczas gdy silna permutacja pseudolosowa jest dodatkowo trudna do odróżnienia od permutacji losowej, nawet jeśli przeciwnik ma dostęp p wyroczni zarówno do permutacji, jak i jej odwrotności.

Ponowna wizyta w sieciach Feistel. Sieć Feistela, wprowadzona w podroziale 6.2.2, zapewnia sposób konstruowania funkcji odwracalnej z dowolnego zbioru funkcji. Sieć Feistela działa w serii rund. Wejściem i-tej rundy jest ciąg znaków o długości $2n$, podzielony na dwie n-bitowe połówki L_i i R_i (odpowiednio „lewa połowa” i „prawa połowa”). Wynikiem i-tej rundy jest $2n$ -bitowy ciąg znaków (L_i, R_i) gdzie

$$L_i := R_{i-1} \text{ i } R_i := L_{i-1} \oplus f_i(R_{i-1})$$

dla pewnej efektywnie obliczalnej (ale niekoniecznie odwracalnej) funkcji f_i mapującej n-bitowe wejścia na n-bitowe wyjścia. Oznaczamy przez $\text{Feistelf}_1, \dots, \text{fr}$ r-okrągłą sieć Feistela za pomocą funkcji f_1, \dots, f_s . (Oznacza to, że $\text{Feistelf}_1, \dots, \text{fr}(L_0, R_0)$ wprowadza $2n$ -bitowy ciąg znaków (L_r, R_r) .) W podroziale 6.2.2 widzieliśmy, że $\text{Feistelf}_1, \dots, \text{fr}$ jest efektywnie odwracalnym permutacją niezależnie od $\{f_i\}$.

Permutację z kluczem możemy zdefiniować za pomocą sieci Feistela, w której $\{f_i\}$ zależy od klucza. Na przykład niech $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ będzie funkcją pseudolosową i zdefiniuje permutację z kluczem $F(1)$

Jak

$$F_k^{(1)}(x) \stackrel{\text{def}}{=} \text{Feistel}_k(x).$$

(Zauważ, że F ma n -bitowy klucz i odwzorowuje $2n$ -bitowe wejścia na $2n$ -bitowe wyjścia.)

Czy $F(1)$ jest pseudolosowe? Krótka myśl pokazuje, że zdecydowanie nie. Dla dowolny klucz $k \in \{0, 1\}^n$, pierwszych n bitów wyniku $F_k^{(1)}$ (to znaczy L_1) są równe n ostatnim bitom wejścia (tj. R_0), co w przypadku funkcji losowej zachodzi z jedynie znikomym prawdopodobieństwem. $\times \{0, 1\}^n$

Próbując ponownie, zdefiniuj $F(2) : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ następująco:

$$F_{k1, k2}^{(2)}(x) \stackrel{\text{def}}{=} \text{Feistel}_{k1, k2}(x). \quad (7.15)$$

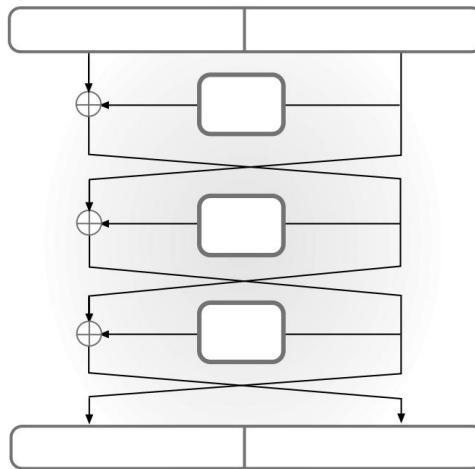
(Zauważ, że $k1$ i $k2$ są kluczami niezależnymi.) Niestety, $F(2)$ również nie jest pseudolosowe, jak zostaniesz poproszony o pokazanie w ćwiczeniu 7.16.

Biorąc to pod uwagę, może być nieco zaskakujące, że trójkątną sieć Feistela jest pseudolosowa. Zdefiniuj kluczowaną permutację $F(3)$, biorąc klucz o długości $3n$ i mapując $2n$ -bitowe wejścia na $2n$ -bitowe wyjścia, w następujący sposób:

$$F_{k1, k2, k3}^{(3)}(x) \stackrel{\text{def}}{=} \text{Feistel}_{k1, k2, k3}(x) \quad (7.16)$$

gdzie ponownie $k1, k2$ i $k3$ są niezależne. Mamy:

TWIERDZENIE 7.23 Jeżeli F jest funkcją pseudolosową, to $F(3)$ jest permutacją pseudolosową.



RYSUNEK 7.3: Trójkątna sieć Feistela zastosowana do skonstruowania permutacji pseudolosowej na podstawie funkcji pseudolosowej.

DOWÓD W standardowy sposób możemy zamiast tego zastąpić funkcje pseudolosowe użyte przy konstrukcji $F(3)$ funkcjami wybranymi jednolicie losowo. Pseudolosowość F oznacza, że ma to jedynie znikomy wpływ na wynik dowolnego probabilistycznego wyróżnika wielomianu w czasie oddziałującego z $F(3)$ jako wyrocznią. Szczegóły pozostawiamy jako ćwiczenie.

Niech D będzie probabilistycznym wyróżnikiem wielomianu w czasie. W pozostałej części dowodu, pokazujemy, że następujący jest nieistotny:

$$\Pr[D \text{Feistelf1,f2,f3}(\cdot)(1n) = 1] = \Pr[D\pi(\cdot)(1n) = 1],$$

gdzie pierwsze prawdopodobieństwo przejmuje jednolity i niezależny wybór f_1, f_2, f_3 z Func_n , a drugie prawdopodobieństwo przejmuje jednolity wybór π z Perm_{2n} . Ustal pewną wartość parametru bezpieczeństwa n i niech $q = q(n)$ oznacza wielomianową górną granicę liczby zapytań Oracle wykonanych przez D . Zakładamy bez utraty ogólności, że D nigdy nie wykonuje dwukrotnie tego samego zapytania Oracle. Koncentrując się na interakcji D z Feistelfem1, $f_2, f_3(\cdot)$, niech (L_i, R_i) oznaczają wartości pośrednie odpowiednio po rundach 1, 2 i 3, które wynikają z tego zapytania (Dzięki temu do żadnej z tych rund nie ma L_i , ale $R_i = (L_{i-1}, R_{i-2})$, i ja $R_i = (L_{i-2}, R_{i-1})$ obserwuje bezpośrednio) lub (L_i, R_i)

(L_0, R_0) i widzi wynik (L_1, R_1) i (L_2, R_2) .

Mówimy, że w R_1 doszło do kolizji, jeśli $R_{i-1}^j = R_{i-1}^l$ dla jakiegoś odrę bnegi i, j . Najpierw udowodnimy, że kolizja w R_1 zachodzi z znikomym prawdopodobieństwem. Rozważ dowolne stałe, odrę bne i, j . Jeśli $R_0^i = L_0^j$ aże $R_0^j = L_0^i$,

$$R_0^i = L_0^j \quad f_1(R_0^i) = L_1^j \quad f_1(R_0^j) = R_1^j.$$

Jeśli $R_0 \neq R^J$ następuje $f_1(R_0)$ i $f_1(R^J)$ są jednolite i niezależne, tzw.

$$\Pr[L_0 = 1 | f_1(R_0) = L] \cdot \Pr[f_1(R^J) = 1 | f_1(R_0) = L] = \Pr[f_1(R^J) = 1 | f_1(R_0) = L] = 2^{-n}.$$

Biorąc sumę związaną po wszystkich odrębnych i, j pokazuje, że prawdopodobieństwo kolizji w R_1 wynosi co najwyżej $q/2^n$.

Załóżmy, że w R_2 zachodzi kolizja, jeśli $R_i \neq R_j$ dla jakiegoś odrębnego i, j . Udowodnimy, że przy braku kolizji w R_1 prawdopodobieństwo kolizji w R_2 jest znikome. Analiza jest taka jak powyżej: rozważ dowolne stałe i, j i zauważ, że jeśli j nie ma kolizji w R_1 , to $R_i \neq R_j$. Zatem $f_2(R_i) = R_i$ i niezależne, a zatem $f_2(R_i) = f_2(R_j)$ są jednolite

$$\Pr[L_1 = 1 | f_2(R_1) = L] = \Pr[f_2(R^J) = 1 | \text{brak kolizji w } R_1] = 2^{-n}.$$

(Zauważ, że f_2 jest niezależne od f_1 , co ułatwia powyższe obliczenia.) Biorąc sumę związaną ze wszystkimi odrębnymi i, j , otrzymujemy

$$\Pr[\text{kolizja na } R_2 | \text{brak kolizji w } R_1] = q^{2^n/2}.$$

Należy pamiętać, że $L_3 = R_i = L_{i+1} \dots L_{i+2^n-1}$; więc pod warunkiem, że nie będe dzieć kolizji w R_1 wartości $L_3 = R_i = L_{i+1} \dots L_{i+2^n-1}$; wszystkie L_q są niezależne i równomiernie rozłożone w $\{0, 1\}^{2^n}$. Jeśli dodatkowo założymy, że w R_2 nie ma kolizji, to wartości L rozkładają się równomiernie pomiędzy wszystkimi ciągami $L_3 = f_3(R_1)$; zatem wszystkie uwarunkowane z q różnych wartości w $\{0, 1\}^{2^n}$. Podobnie, $R_i = L_{i+1} \dots L_{i+2^n-1}$ są równomiernie rozłożone w $\{0, 1\}^{2^n}$. Brak kolizji w R_2 , wartości R_1 rozłożone w $\{0, 1\}^{2^n}$, niezależne od siebie oraz $L_3 = R_i = L_{i+1} \dots L_{i+2^n-1}$.

Podsumowując: podczas odpytywania $F(3)$ (z jednolitymi funkcjami okrągłymi) na szeregu q różnych danych wejściowych, z wyjątkiem wartości wyjściowych (L, R_1), z znikomym prawdopodobieństwem, rozkładają się jednolito i niezależnie, ale różne wartości n-bitowe, a $\{R_3\}$ to jednolite i niezależne wartości n-bitowe. Natomiast podczas sprawdzania losowej permutacji na szeregu q różnych wejść, wyjście wartości (L, R_q) są jednolite i niezależne, ale różne, 2n-bitowe wartości. Najlepszym atakiem jest atakem穷举, ale D jest zatem przypuszczenie, że oddziaływało ono z losową permutacją, jeśli L dla pewnego odrębnego i, j . Ale nawet w tym przypadku zdarzenie to zachodzi z znikomym prawdopodobieństwem 3. Można to przekształcić w formalny dowód.



$F(3)$ nie jest silną permutacją pseudolosową, jak pokazano w ćwiczeniu 7.17. Na szczęście dodanie czwartej rundy daje silną permutację pseudolosową. Szczegóły podano w rozdziale Konstrukcja 7.24.

TWIERDZENIE 7.25 Jeśli F jest funkcją pseudolosową, to Konstrukcja 7.24 jest silną permutacją pseudolosową, która odwzorowuje 2n-bitowe dane wejściowe na 2n-bitowe wyjścia (i używa 4n-bitowego klucza).

KONSTRUKCJA 7.24

Niech F będzie funkcją z kluczem zachowującą długość. Zdefiniuj kluczowaną permutację $F(4)$ w następujący sposób:

- Wejścia: Klawisz $k = (k_1, k_2, k_3, k_4)$ z $|k_i| = n$ i dane wejściowe $x \in \{0, 1\}^{2n}$ analizowane jako (L_0, R_0) za pomocą $|L_0| = |R_0| = r_z$.
- 1} • Obliczenie:
 1. Oblicz $L_1 := R_0$ i $R_1 := L_0 \oplus F_{k_1}(R_0)$.
 2. Oblicz $L_2 := R_1$ i $R_2 := L_1 \oplus F_{k_2}(R_1)$.
 3. Oblicz $L_3 := R_2$ i $R_3 := L_2 \oplus F_{k_3}(R_2)$.
 4. Oblicz $L_4 := R_3$ i $R_4 := L_3 \oplus F_{k_4}(R_3)$.
 5. Wyjście (L_4, R_4) .

Silna permutacja pseudolosowa dowolnej funkcji pseudolosowej.

7.7 Założenia dotyczące kryptografii klucza prywatnego

Pokazaliśmy, że (1) jeśli istnieją permutacje jednokierunkowe, to istnieją generatory pseudolosowe; (2) jeśli istnieją generatory pseudolosowe, to istnieją funkcje pseudolosowe; oraz (3) jeśli istnieją funkcje pseudolosowe, to istnieją (silne) permutacje pseudolosowe. Choć nie udowodniliśmy tego tutaj, możliwe jest skonstruowanie generatorów pseudolosowych z funkcji jednokierunkowych. Mamy zatem następujące podstawowe twierdzenie:

TWIERDZENIE 7.26 Jeśli istnieją funkcje jednokierunkowe, to istnieją także generatory pseudolosowe, funkcje pseudolosowe i silne permutacje pseudolosowe.

Mogą nim być wszystkie schematy klucza prywatnego, które badaliśmy w rozdziałach 3 i 4 zbudowane z generatorów/funkcji pseudolosowych. Mamy zatem:

TWIERDZENIE 7.27 Jeśli istnieją funkcje jednokierunkowe, istnieją także schematy szyfrowania kluczem prywatnym zabezpieczone przez CCA i bezpieczne kody uwierzytelniania wiadomości.

Oznacza to, że funkcje jednokierunkowe są wystarczające dla całej kryptografii klucza prywatnego. Tutaj pokazujemy, że funkcje jednokierunkowe są również konieczne.

Pseudolosowość implikuje funkcje jednokierunkowe. Zaczniemy od pokazania, że generatory pseudolosowe implikują istnienie funkcji jednokierunkowych:

TWIERDZENIE 7.28 Jeżeli istnieje generator pseudolosowy, to istnieje również funkcja jednokierunkowa.

DOWÓD Niech G będzie generatorem pseudolosowym ze współczynnikiem rozszerzalności ($n = 2n$). (Z twierdzenia 7.20 wiemy, że istnienie generatora pseudolosowego implikuje istnienie generatora z tym współczynnikiem ekspansji.) Pokazujemy, że samo G jest jednokierunkowe. Efektywna obliczalność jest prosta (ponieważ G można obliczyć w czasie wielomianowym). Pokazujemy, że zdolność do odwracania G można przełożyć na zdolność odróżnienia wyniku G od jednorodności.

Intuicyjnie jest to prawdą, ponieważ możliwość odwrócenia G implikuje możliwość znalezienia materiału siewnego używanego przez generator.

Niech A będzie dowolnym probabilistycznym algorytmem wielomianowym. Pokazujemy, że $\Pr[\text{Invert}_A, G(n) = 1]$ jest nieistotne (por. Definicja 7.1). Aby to zobaczyć, rozważ następujący wyróżnik ppt D : na wejściu ciąg $w \in \{0, 1\}^{2n}$, uruchom $A(w)$, aby otrzymać wynik s . Jeżeli $G(s) = w$, to wyjście 1; w przeciwnym razie wyjście 0.

Przeanalizujmy teraz zachowanie D . Najpierw rozważmy prawdopodobieństwo, że D wyświetli 1, gdy jego ciąg wejściowy w jest jednolity. Ponieważ w zakresie G znajduje się co najwyżej $2n$ wartości (mianowicie wartości $\{G(s)\}_{s \in \{0, 1\}^n}$), prawdopodobieństwo, że w należy do zakresu G wynosi co najwyżej $2n/2 = n$. Gdy w nie należy do zakresu G , A nie może obliczyć odwrotności w , a zatem niemożliwe jest, aby D wyprowadziło 1. Dochodzimy do wniosku, że

$$\Pr_w_{\{0,1\}^{2n}} [D(w) = 1] = 2^{-r_{zecz}}.$$

Z drugiej strony, jeśli $w = G(s)$ dla ziarna $s \in \{0, 1\}^n$ wybranego równomiernie i losowo, to z definicji A oblicza poprawną odwrotność (i tak D wyprowadza 1) z prawdopodobieństwem dokładnie równym $\Pr[\text{Odwróć}_A, G(n) = 1]$. Zatem,

$$\Pr_w_{\{0,1\}^{2n}} [D(w) = 1] = \Pr_{s \in \{0,1\}^n} [D(G(s)) = 1] = \Pr[\text{Odwróć}_A, G(n) = 1].$$

Ponieważ G jest generatorem pseudolosowym, powyższe musi być zaniedbywalne. Ponieważ jest $-r_{zecz}$ to pomijalne, oznacza to, że $\Pr[\text{Invert}_A, G(n) = 1]$ jest również pomijalne 2, więc G jest jednokierunkowe. ■

Nietywialne szyfrowanie kluczem prywatnym implikuje funkcje jednokierunkowe. Twierdzenie 7.28 nie sugeruje, że do konstruowania bezpiecznych schematów szyfrowania klucza prywatnego potrzebne są funkcje jednokierunkowe, ponieważ może być możliwe skonstruowanie tych ostatnich bez polegania na generatorze pseudolosowym. Co więcej, możliwe jest skonstruowanie całkowicie tajnych schematów szyfrowania (patrz rozdział 2), o ile tekst jawny nie jest dłuższy niż klucz. Zatem dowód, że bezpieczne szyfrowanie klucza prywatnego implikuje funkcje jednokierunkowe, wymaga więcej uwagi.

PROPOZYCJA 7.29 Jeżeli istnieje schemat szyfrowania klucza prywatnego z zabezpieczeniem EAV, który szyfruje wiadomości dwa razy dłużej niż jego klucz, wówczas istnieje funkcja jednokierunkowa.

DOWÓD Niech $\Pi = (\text{Enc}, \text{Dec})$ bę dzie schematem szyfrowania klucza prywatnego, który ma nieroróżnialne szyfrowanie w obecności podsłuchującego i szyfruje wiadomości o długości $2n$, gdy klucz ma długość n . (Zakładamy dla uproszczenia, że klucz jest wybierany jednolicie.) Powiedzmy, że gdy używany jest klucz n -bitowy, Enc używa co najwyżej (n) bitów losowości. Oznacz szyfrowanie wiadomości m za pomocą klucza k i losowości r według Encka ($m; r$).

Zdefiniuj następującą funkcję f :

$$f(k, m, r) \stackrel{\text{def}}{=} \text{Enck}(m; r) m,$$

gdzie $|k| = n$, $|m| = 2n$ i $|r| = (n)$. Twierdzimy, że f jest funkcją jednokierunkową. Oczywiście można to skutecznie obliczyć; pokazujemy, że trudno to odwrócić.

Zakładając, że A bę dzie dowolnym algorytmem ppt, pokażemy, że $\Pr[\text{Invert}A, f(n) = 1]$ jest pomijalnie małe (por. Definicja 7.1).

Rozważmy następującą probabilistycznego przeciwnika A atakującego schemat szyfrowania klucza prywatnego Π (tj. w eksperymencie $\text{PrivKeav}_{\Pi, A}(n)$):

Przeciwnik A ($1n$)

1. Wybierz mundur $m_0, m_1 \in \{0, 1\}^{2n}$ i wyprowadź je. Odnośnie-
otrzymać w zamian szyfrogram bę dący wyzwaniem, c .
2. Uruchom $A(c, m_0)$, aby otrzymać (k, M, R) . Jeśli $f(k, M, r) = m_0$,
wynik 0; w przeciwnym razie, wynik 1.

Przeanalizujemy teraz zachowanie A . Gdy c jest szyfrowaniem m_0 , wówczas cm_0 rozkłada się dokładnie tak, jak $f(k, m_0, r)$ dla jednolitych k, m_0 i r . Dlatego A wyprowadza prawidłową odwrotność cm_0 (a zatem A wyprowadza 0) z prawdopodobieństwem dokładnie równym $\Pr[\text{Invert}A, f(n) = 1]$.

Z drugiej strony, gdy c jest szyfrowaniem m_1 , wówczas c jest niezależne od m_0 . Dla dowolnej ustalonej wartości szyfrogramu wyzwania c istnieje co najwyżej 2^n możliwych wiadomości (po jednej dla każdego możliwego klucza), którym c może odpowiadać. Ponieważ m_0 jest jednolitym ciągiem $2n$ -bitowym, oznacza to prawdopodobieństwo, że istnieje jakiś klucz k , dla którego $\text{Deck}(c) = m_0$ wynosi co najwyżej $\frac{2^n}{2^n} = 2^{-n}$. Daje to górną granicę prawdopodobieństwa, z jakim A może ewentualnie wyprowadzić prawidłową odwrotność c przy f , a zatem górną granicę prawdopodobieństwa, z jakim A wyprowadzi w tym przypadku 0.

Łącząc powyższe, mamy:

$$\begin{aligned} \Pr[\text{PrivKeav}_{\Pi, A}(n) = 1] &= \frac{1}{2} \cdot \Pr[A \text{ wyjścia } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[A \text{ wyjścia } 1 \mid b = 1] \\ &= \frac{1}{2} \cdot \Pr[\text{Odwróć}A, f(n) = 1] + \frac{1}{2} \cdot 1 - \Pr[\text{Odwróć}A, f(n) = 1] \\ &= \frac{1}{2} + \frac{1}{2} \cdot \Pr[\text{Odwróć}A, f(n) = 1]. \end{aligned}$$

Bezpieczeństwo Π oznacza, że $\Pr[\text{PrivKey}_{\Pi,A}(n) = 1] \geq 1 - \text{negl}(n)$ dla jakiejś pomijalnej funkcji negl . To z kolei oznacza, że $\Pr[\text{Invert}_{A,f}(n) = 1]$ jest nieistotne, co kończy dowód, że f jest jednokierunkowe. ■

Kody uwierzytelniania wiadomości implikują funkcje jednokierunkowe. Prawda jest również, że kody uwierzytelniające wiadomości spełniające definicję 4.2 implikują istnienie funkcji jednokierunkowych. Podobnie jak w przypadku szyfrowania kluczem prywatnym, dowód tego faktu jest dość subtelny, ponieważ bezwarunkowe kody uwierzytelniające wiadomości istnieją, jeśli istnieje a priori ograniczona liczba wiadomości, które zostaną uwierzytelnione. (Patrz sekcja 4.6.) Zatem dowód opiera się na fakcie, że Definicja 4.2 wymaga zabezpieczeń nawet wtedy, gdy przeciwnik widzi znaczniki uwierzytelniające dowolnej (wielomianowej) liczby wiadomości. Dowód jest nieco skomplikowany, więc c nie podajemy go tutaj.

Dyskusja. Dochodzimy do wniosku, że istnienie funkcji jednokierunkowych jest konieczne i wystarczające dla każdej (nietrywialnej) kryptografii klucza prywatnego. Innymi słowy, funkcje jednokierunkowe są założeniem minimalnym, jeśli chodzi o kryptografię klucza prywatnego. Co ciekawe, wydaje się, że nie dotyczy to funkcji skrótu i szyfrowania kluczem publicznym, w przypadku których wiadomo, że funkcje jednokierunkowe są konieczne, ale nie wiadomo (lub nie uważa się ich) za wystarczające.

7.8 Nierozróżnialność obliczeniowa

Pojęcie nierozróżnialności obliczeniowej ma kluczowe znaczenie w teorii kryptografii i leży u podstaw wielu tego, co widzieliśmy w rozdziale 3 i tym rozdziale. Nieformalnie dwa rozkłady prawdopodobieństwa są obliczeniowo nierozróżnialne, jeśli żaden wydajny algorytm nie jest w stanie ich rozróżnić (lub rozróżnić). Bardziej szczegółowo, rozważmy dwa rozkłady X i Y na ciągach o określonej długości; oznacza to, że X i Y przypisują pewne prawdopodobieństwo każdemu ciągowi w $\{0, 1\}^n$. Kiedy mówimy, że jakiś algorytm D nie jest w stanie rozróżnić tych dwóch rozkładów, mamy na myśli, że D nie jest w stanie stwierdzić, czy otrzyma串 znaków próbkowany według rozkładu X , czy też串 znaków próbkowany według rozkładu Y . Inaczej mówiąc, jeśli wyobrażymy sobie, że D wyprowadza „0”, gdy uważa, że jego dane wejściowe zostały próbkowane zgodnie z X , i wyprowadza „1”, jeśli uważa, że jego dane wejściowe zostały próbkowane zgodnie z Y , wówczas prawdopodobieństwo, że D wyprowadza „1” powinno być w przybliżeniu tą samą niezależnie od tego, czy D otrzymuje próbki z X , czy z Y . Innymi słowy chcemy

$$\Pr_{s \sim X} [D(s) = 1] = \Pr_{s \sim Y} [D(s) = 1]$$

być małym.

Powinno to przypominać sposób, w jaki zdefiniowaliśmy generatory pseudolosowe i rzeczywiście wkrótce formalnie zdefiniujemy na nowo pojęcie generatora pseudolosowego, używając tej terminologii.

Formalna definicja nierozróżnialności obliczeniowej odnosi się do zespołów prawdopodobieństwa, które są nieskończonymi ciągami rozkładów prawdopodobieństwa.

(Ten formalizm jest niezbędny do sensownego podejścia asymptotycznego). Chociaż pojęcie to można uogólnić, dla naszych celów rozważamy zespoły prawdopodobieństw, w których podstawowe rozkłady są indeksowane liczbami naturalnymi. Jeśli dla każdej liczby naturalnej n mamy rozkład X_n , to $X = \{X_n\}_{n=1}^{\infty}$ jest zespołem prawdopodobieństwa. Czy sto jest tak, że $X_n = Y_t(n)$ dla jakiejś funkcji t, w takim przypadku piszemy $\{Y_t(n)\}_{n=1}^{\infty}$ zamiast $\{X_n\}_{n=1}^{\infty}$.

Będziemy zainteresowani jedynie efektywnie próbowanymi zespołami prawdopodobieństwa. Zespół $X = \{X_n\}_{n=1}^{\infty}$ można skutecznie próbować, jeśli istnieje probabilistyczny algorytm wielomianowy S taki, że zmienne losowe $S(1^n)$ i X_n mają identyczny rozkład. Oznacza to, że algorytm S jest skutecznym sposobem próbkowania X.

Możemy teraz formalnie zdefiniować, co to znaczy, że dwa zespoły są obliczeniowo nierozróżnialne.

DEFINICJA 7.30 Dwa zespoły prawdopodobieństwa $X = \{X_n\}_{n=1}^{\infty}$ i $Y = \{Y_n\}_{n=1}^{\infty}$ są obliczeniowo nierozróżnialne, oznaczane $X \xrightarrow{c} Y$, jeśli dla każdego probabilistycznego wyróżnika wielomianowo-czasowego D istnieje funkcja pomijalna taka, że:

$$\Pr_x [D(1^n, x) = 1] - \Pr_y [D(1^n, y) = 1] \leq \text{negl}(n).$$

W definicji D ma dane wejście jednoargumentowe 1^n , więc może działać w wielomianu czasu w n. Jest to ważne, gdy wyjścia X_n i Y_n mogą mieć długość mniejszą niż n. W skrócie w wyrażenach prawdopodobieństwa będziemy czasami pisać X jako symbol zastępczy dla losowej próbki z rozkładu X. Oznacza to, że zapiszemy $\Pr_x[D(1^n, X_n) = 1]$ zamiast $\Pr_x[D(1^n, x) = 1]$.

Relacja nierozróżnialności obliczeniowej jest przechodnia: jeśli $X \xrightarrow{c} Y$ i $Y \xrightarrow{c} Z$, to $X \xrightarrow{c} Z$.

Pseudolosowość i generatory pseudolosowości. Pseudolosowość jest po prostu szczególnym przypadkiem nierozróżnialności obliczeniowej. Dla dowolnego wlotu U oznaczają rozkład równomierny na $\{0, 1\}^s$. Możemy zdefiniować taki generator pseudolosowy w następujący sposób:

DEFINICJA 7.31 Niech (\cdot) będzie wielomianem i niech G będzie (deterministycznym) algorytmem wielomianowo-czasowym, gdzie dla wszystkich s obowiązuje, że $|G(s)| = |\{0, 1\}^s|$. Mówimy, że G jest generatorem pseudolosowym, jeśli spełnione są dwa poniższe warunki:

1. (Rozwinęcie:) Dla każdego n zachodzi to, że $(n) > n$.

2. (Pseudolosowość:) Zespół $\{G(U_n)\}_n \subset N$ jest obliczeniowo nie do odróżnienia od zespołu $\{U(n)\}_n \subset N$.

Można również sformułować wiele innych definicji i założeń zawartych w tej książce jako szczególne przypadki lub warianty nieroróżnialności obliczeniowej.

Wiele próbek. Ważnym twierdzeniem dotyczącym nieroróżnialności obliczeniowej jest to, że wielomianowo wiele próbek (efektywnie możliwych do próbkowania) zespoły nieroróżnialne obliczeniowo są również nieroróżnialne obliczeniowo.

TWIERDZENIE 7.32 Niech X i Y będą efektywnie próbkowanymi zespołami prawdopodobieństwa, które są obliczeniowo nieroróżnialne. Następnie dla każdego wielomianu p zespół $X = \{(X_n, \dots, X_{p(n)})\}_n \subset N$ jest obliczeniowo nieroróżnialny - (1) zdolny z zespołu $Y = \{(Y_n, \dots, Y_{p(n)})\}_n \subset N$.

Na przykład, niech G będzie generatorem pseudolosowym ze współczynnikiem rozszerzenia 2^n , w tym przypadku zespoły $\{G(U_n)\}_n \subset N$ i $\{U_{2n}\}_n \subset N$ są obliczeniowo nie do odróżnienia. W dowodzie Twierdzenia 7.22 pokazaliśmy, że dla dowolnego wielomianu t zespoły

$$\{(G(U_1), \dots, G(U_t))\}_n \subset N \text{ i } \{(U_{2n}, \dots, U_{2n})\}_n \subset N$$

są również nie do odróżnienia obliczeniowego. Twierdzenie 7.32 jest dowodzone za pomocą argumentu hybrydowego dokładnie w ten sam sposób.

Referencje i dodatkowe lektury

Pojęcie funkcji jednokierunkowej zostało po raz pierwszy zaproponowane przez Diffiego i Hellmana [58], a później sformalizowane przez Yao [179]. Predykaty hard-core wprowadzili Blum i Micali [37], a fakt, że istnieje hard-core predykaty predykatu dla każdej funkcji jednokierunkowej udowodnili Goldreich i Levin [79].

Pierwszą konstrukcję generatorów pseudolosowych (przy określonym założeniu twardości z teorii liczb) przedstawili Blum i Micali [37]. Konstrukcja generatora pseudolosowego z dowolnej jednokierunkowej permutacji była podana przez Yao [179], a wynik, że generatorы pseudolosowe można skonstruować z dowolnej funkcji jednokierunkowej, pokazali Håstad i in. [85]. Funkcje pseudolosowe zdefiniowali i skonstruowali Goldreich, Goldwasser i Micali [78] oraz ich rozszerzenie na (silne) permutacje pseudolosowe pokazali Luby i Rackoff [116]. Fakt, że funkcje jednokierunkowe są a