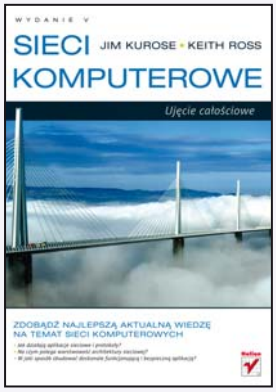
Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Sieci komputerowe. Ujęcie całościowe. Wydanie V



Autorzy: James F. Kurose, Keith W. Ross
Tłumaczenie: Tomasz Walczak
ISBN: 978-83-246-2632-8
Tytuł oryginału: Computer Networking: A Top-Down Approach (5th Edition)
Format: 172×245, stron: 968

Zdobądź najlepszą aktualną wiedzę na temat sieci komputerowych

- Jak działają aplikacje sieciowe i protokoły?
- Na czym polega warstwowość architektury sieciowej?
- W jaki sposób zbudować doskonale funkcjonującą i bezpieczną aplikację?

Istnieje wiele książek na temat sieci komputerowych, jednak podręcznik, który trzymasz w rękach, wyraźnie wyróżnia się na ich tle. Dzieje się tak z powodu niebanalnej konstrukcji tej publikacji, opierającej się na metodzie omawiania zagadnień „od góry do dołu", od ogółu do szczegółu, a więc prezentowania jako pierwszej warstwy aplikacji, a następnie kolejnych, niższych warstw – aż do warstwy fizycznej. W ten sposób zwraca się szczególną uwagę na tę warstwę, która rozwija się najszybciej i jest najbardziej interesującym elementem sieci.

Z książki „Sieci komputerowe. Ujęcie całościowe. Wydanie V" dowiesz się wszystkiego na temat programowania aplikacji, protokołów wyższych warstw oraz aktualnych zabezpieczeń sieciowych. Oprócz rzetelnie przedstawionej podstawowej wiedzy znajdziesz tu znacznie bardziej szczegółowe informacje o sieciach P2P opartych na protokole BitTorrent, dodatkowe materiały na temat sieci DHT, zaktualizowane omówienie sieci dostępowych (między innymi kablowych, FTTH i DSL), a także informacje dotyczące historii sieci komputerowych i Internetu. Korzystając z tego podręcznika, z łatwością zaprojektujesz własną, świetnie funkcjonującą aplikację sieciową.

- Sieci dostępowe i nośniki fizyczne
- Architektura aplikacji sieciowych
- Warstwy: aplikacji, transportowa, sieci, łącza danych i fizyczna
- Przepustowość w sieciach komputerowych
- Programy klienta i serwera
- Technologia WWW i protokół http
- Sieci bezprzewodowe i mobilne
- Multimedia
- Aplikacje z obszaru P2P
- Bezpieczeństwo w sieciach komputerowych
- Zarządzanie sieciami

Wszystko, co chciałbyś wiedzieć o aplikacjach sieciowych, protokołach i Internecie

Spis treści

ROZDZIAŁ 8

Bezpieczeństwo w sieciach komputerowych

Wiele stron temu, w podrozdziale 1.6, opisaliśmy niektóre z najczęstszych i najbardziej szkodliwych ataków internetowych, między innymi ataki za pomocą szkodliwego oprogramowania, przez odmowę usług (DoS), przechwytywanie danych, podszywanie się pod nadawcę i modyfikowanie oraz usuwanie komunikatów. Choć od tego czasu Czytelnicy nauczyli się wielu rzeczy o sieciach komputerowych, nadal nie dowiedzieli się, jak zabezpieczać je przed atakami wymienionymi w podrozdziale 1.6. Uzbrojeni w nowo nabytą wiedzę ekspercką na temat sieci i protokołów internetowych Czytelnicy mogą teraz przystąpić do szczegółowej analizy bezpiecznej komunikacji, a przede wszystkim tego, jak zabezpieczać sieci komputerowe przed napastnikami.

Poznajmy Alicję i Bartka, dwie osoby, które zamierzają się porozumieć i chcą, aby ich komunikacja była „bezpieczna". Ponieważ jest to książka poświęcona sieciom, nadmieńmy, że Alicją i Bartkiem mogą być dwa routery, które muszą wymienić tablice routingu, klient i serwer, które muszą nawiązać bezpieczne połączenie transportowe, albo dwie aplikacje e-mail, które muszą bezpiecznie wymienić pocztę — są to konkretne przypadki, które zbadamy w dalszej części rozdziału. Alicja i Bartek to osoby cieszące się dużą popularnością wśród ekspertów od bezpieczeństwa, być może dlatego, że przyjemniej jest posługiwać się imionami niż ogólnikowymi oznaczeniami w rodzaju „A" lub „B". Zwykło się mawiać, że bezpieczna komunikacja jest potrzebna w romansach pozamałżeńskich, podczas wojny i w transakcjach handlowych. Ponieważ wolimy ten pierwszy scenariusz od pozostałych dwóch, będziemy wyobrażać sobie Alicję i Bartka jako parę zakochanych.

Stwierdziliśmy, że Alicja i Bartek chcą komunikować się „bezpiecznie", ale co to właściwie znaczy? Jak się niebawem przekonamy, bezpieczeństwo (jak miłość) ma wiele aspektów. Bez wątpienia Alicja i Bartek chcieliby, aby ich komunikacja pozostała ukryta przed innymi osobami (na przykład zazdrosnym małżonkiem). Prawdopodobnie chcieliby też mieć pewność, że rzeczywiście porozumiewają się ze sobą, a jeśli ktoś zmodyfikuje ich wiadomości, to będzie można wykryć tę modyfikację. W pierwszej części rozdziału omówimy techniki, które zapewniają szyfrowanie i deszyfrowanie komunikacji, uwierzytelnianie porozumiewających się stron oraz integralność wiadomości.

W drugiej części rozdziału zbadamy, jak wykorzystać podstawowe zasady kryptograficzne do tworzenia bezpiecznych protokołów sieciowych. Ponownie zastosujemy podejście góra-dół i przyjrzymy się bezpiecznym protokołom w każdej z czterech górnych warstw (zaczniemy od warstwy aplikacji). Pokażemy, jak zabezpieczać pocztę elektroniczną i połączenia TCP, jak zapewnić pełne bezpieczeństwo w warstwie sieci oraz jak chronić bezprzewodowe sieci lokalne. W trzeciej części rozdziału rozważymy bezpieczeństwo operacyjne, związane z ochroną sieci firmowych przed atakami. Przede wszystkim dokładnie przyjrzymy się, w jaki sposób zwiększyć bezpieczeństwo takich sieci za pomocą zapór i systemów wykrywania włamań.

8.1. Czym jest bezpieczeństwo sieci?

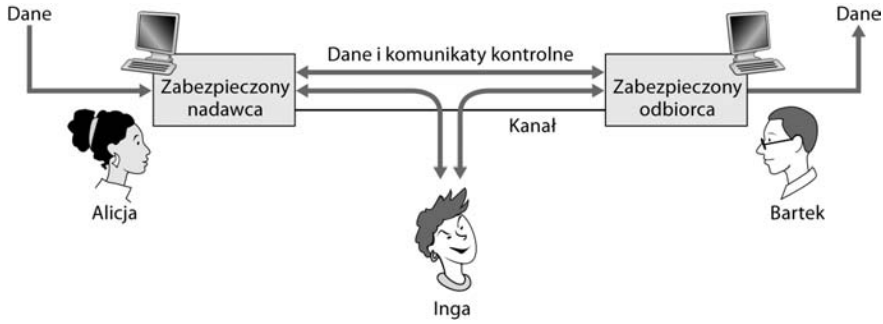
Na początek wróćmy do naszych kochanków, Alicji i Bartka, którzy chcą porozumiewać się „bezpiecznie". Co to oznacza? Alicja chce, aby jej wiadomość mógł zrozumieć tylko Bartek, mimo że komunikują się przez niezabezpieczone medium, w którym intruz (Inga) może przechwycić, odczytać i przeprowadzić obliczenia na danych przesyłanych od Alicji do Bartka. Ponadto Bartek chce być pewien, że odebrana wiadomość rzeczywiście została wysłana przez Alicję, a Alicja — że rzeczywiście porozumiewa się z Bartkiem. Alicja i Bartek chcą również mieć pewność, że treść ich wiadomości nie została po drodze zmodyfikowana. Chcą wreszcie gwarancji, że w ogóle będą mogli się porozumieć, tzn. że nikt nie odmówi im dostępu do zasobów niezbędnych do komunikacji. Wziąwszy pod uwagę te kwestie, możemy zidentyfikować pożądane właściwości **bezpiecznej komunikacji**:

- *Poufność.* Tylko nadawca i odbiorca powinni móc zrozumieć treść przesłanej wiadomości. Ponieważ wiadomość może zostać przechwycona, trzeba ją **zaszyfrować** (ukryć dane) w taki sposób, aby ewentualny podsłuchiwacz nie mógł jej zrozumieć. Ów aspekt poufności jest prawdopodobnie tym, z czym większości osób kojarzy się termin *bezpieczna komunikacja*. Kryptograficzne techniki szyfrowania i deszyfrowania danych omówimy w podrozdziale 8.2.

- *Uwierzytelnianie punktów końcowych.* Zarówno nadawca, jak i odbiorca powinni dysponować środkami potwierdzania tożsamości partnera — upewniania się, że jest on rzeczywiście tym, za kogo się podaje. Podczas rozmowy w cztery oczy można łatwo rozwiązać ten problem za pomocą identyfikacji wzrokowej. Kiedy porozumiewający się partnerzy używają nośnika, który nie pozwala obejrzeć drugiej osoby, uwierzytelnianie staje się bardziej skomplikowane. Dlaczego mielibyśmy na przykład wierzyć, że wiadomość e-mail z informacją, iż przesłał ją nasz przyjaciel, rzeczywiście pochodzi od tego przyjaciela?
- *Integralność wiadomości.* Jeśli nawet nadawca i odbiorca zdołają uwierzytelnić się nawzajem, będą również chcieli mieć gwarancję, że treść ich komunikacji nie zostanie zmodyfikowana podczas transmisji (przypadkowo lub celowo). Integralność wiadomości mogą zagwarantować techniki obliczania sum kontrolnych, z którymi spotkaliśmy się już w niezawodnych protokołach transportowych i łącza danych. Techniki uwierzytelniania punktów końcowych i zapewniania integralności wiadomości zbadamy w podrozdziale 8.3.
- *Bezpieczeństwo operacyjne.* Prawie wszystkie organizacje (firmy, uniwersytety itd.) mają obecnie sieci podłączone do publicznego internetu. Te sieci potencjalnie mogą zostać zaatakowane poprzez publiczny internet. Napastnicy mogą spróbować umieścić robaki na hostach sieciowych, odkryć sekrety korporacji, odwzorować konfigurację wewnętrznej sieci lub przeprowadzić atak DoS. W podrozdziale 8.8 pokażemy, że urządzenia zapewniające bezpieczeństwo operacyjne, na przykład zapory i systemy wykrywania włamań, służą do przeciwdziałania atakom na sieci firmowe. Zapora działa między siecią organizacji i siecią publiczną oraz kontroluje przekazywanie pakietów do sieci i poza nią. System wykrywania włamań przeprowadza „dogłębną inspekcję pakietów" i alarmuje administratorów sieci o podejrzanych działaniach.

Skoro ustaliliśmy już, co będziemy rozumieć przez bezpieczeństwo sieci, rozważmy teraz, do jakich informacji ma dostęp intruz i jakie może podjąć działania. Scenariusz ataku przedstawiono na rysunku 8.1. Alicja (nadawca) chce wysłać dane do Bartka (odbiorcy). Aby bezpiecznie przenieść dane, spełniając kryteria poufności, uwierzytelniania i integralności, Alicja i Bartek będą wymieniać komunikaty kontrolne i komunikaty danych (podobnie jak nadawcy i odbiorcy TCP wymieniają segmenty kontrolne i segmenty danych). Niektóre lub wszystkie te komunikaty będą zaszyfrowane. Jak opisaliśmy w podrozdziale 1.6, intruz ma następujące możliwości działania:

- *podsłuchiwanie* — przeglądanie i rejestrowanie zawartości komunikatów kontrolnych oraz komunikatów danych przesyłanych kanałem;
- *modyfikowanie*, *wstawianie* lub *usuwanie* komunikatów albo ich zawartości.



Rysunek 8.1. Nadawca, odbiorca i intruz (Alicja, Bartek i Inga)

Jak się okaże, bez odpowiednich środków zaradczych możliwości te pozwolą intruzowi na przypuszczenie wielu różnych ataków: podsłuchiwanie komunikacji (i wykradania haseł lub danych), przybieranie tożsamości innej osoby, przejmowanie bieżącej sesji, blokowanie dostępu do usług poprzez przeciążenie zasobów systemu itd. Podsumowanie zgłoszonych ataków można znaleźć w CERT Coordination Center [CERT 2009]. Warto też zajrzeć do [Cisco Security 2009; Voydock 1983; Bhimani 1996; Skoudis 2006].

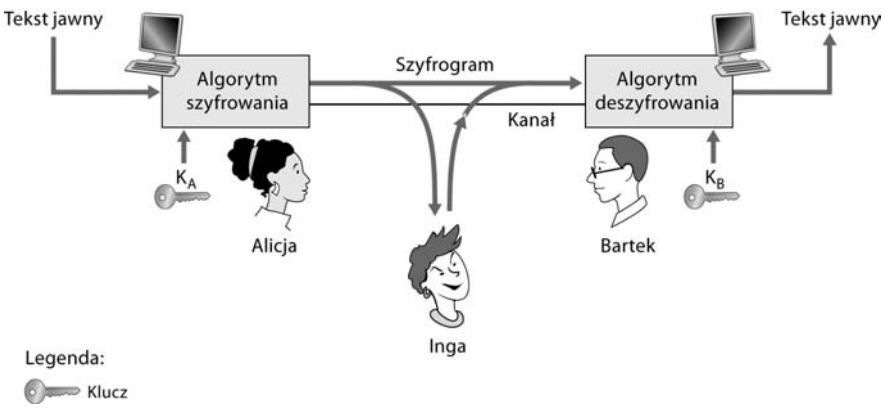
Skoro ustaliliśmy już, że w internecie czyhają bardzo realne zagrożenia, zastanówmy się, jakie są internetowe odpowiedniki Alicji i Bartka, naszych przyjaciół, którzy chcą się bezpiecznie porozumiewać? Oczywiście, Bartek i Alicja mogą być użytkownikami dwóch systemów końcowych — prawdziwą Alicją i prawdziwym Bartkiem, którzy chcą bezpiecznie wymienić pocztę. Mogą również być uczestnikami elektronicznej transakcji handlowej. Bartek może być osobą, która chce przesłać do serwera numer swojej karty kredytowej, aby kupić coś w sklepie internetowym. Podobnie Alicja może być osobą, która chce połączyć się ze swoim bankiem. Strony komunikacji mogą również być elementami infrastruktury sieciowej. Wiemy już, że bezpiecznej komunikacji wymagają serwery nazw domenowych (DNS, podrozdział 2.5) i procesy routingu (podrozdział 4.6). To samo dotyczy aplikacji do zarządzania siecią, które zbadamy w rozdziale 9. Intruz, który potrafiłby aktywnie zakłócać lub kontrolować wyszukiwania DNS (zagadnienie to omawiamy w podrozdziale 2.5), obliczenia routingu [Murphy 2003] albo mechanizmy zarządzania siecią [RFC 2574], mógłby narobić ogromnego zamieszania w internecie.

Wyjaśniliśmy ogólne zasady bezpiecznej komunikacji, poznaliśmy kilka najważniejszych definicji i zbadaliśmy sens zabezpieczania sieci, więc teraz możemy przejść do kryptografii. Jak się wkrótce Czytelnicy przekonają, kryptografia zapewnia nie tylko poufność, ale również uwierzytelnianie i integralność, a zatem jest kamieniem węgielnym bezpieczeństwa sieci.

8.2. Zasady kryptografii

Choć kryptografia ma historię sięgającą czasów Juliusza Cezara, współczesne techniki kryptograficzne — również te używane w internecie — opierają się na badaniach prowadzonych w ciągu ostatnich 30 lat. Książki *The Codebreakers* [Kahn 1967] oraz *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography* [Singh 1999] oferują fascynujące spojrzenie na długą historię kryptografii. Kompletne studium kryptografii zajęłoby całą książkę [Kaufman 1995, Schneier 1995], więc tutaj zajmiemy się tylko jej najważniejszymi aspektami, szczególnie tymi, które mają związek z internetem. Nadmieńmy też, że choć w tym podrozdziale skupimy się na poufności, to techniki kryptograficzne są również nierozerwalnie splątane z uwierzytelnianiem, integralnością, niezaprzeczalnością i innymi aspektami bezpieczeństwa.

Techniki kryptograficzne pozwalają nadawcy przekształcić dane tak, aby intruz nie mógł uzyskać żadnych informacji z przechwyconej wiadomości. Oczywiście, odbiorca musi mieć możliwość odtworzenia oryginalnych danych. Kilka najważniejszych terminów zilustrowano na rysunku 8.2.



Rysunek 8.2. Komponenty kryptograficzne

Przypuśćmy, że Alicja chce przesłać wiadomość do Bartka. Wiadomość Alicji w pierwotnej postaci (na przykład „`Bartku, kocham cię. Alicja`”) określa się mianem **tekstu jawnego** lub **zwykłego tekstu**. Alicja szyfruje tekst jawny za pomocą **algorytmu szyfrowania**, uzyskując **szyfrogram** nieczytelny dla osób postronnych. Co ciekawe, w wielu współczesnych systemach kryptograficznych, również tych używanych w internecie, sama technika szyfrowania jest *znana* — opublikowana, ustandaryzowana i dostępna dla wszystkich (na przykład [RFC 1321; RFC 2437; RFC 2420; NIST 2001]), również dla potencjalnego intruza! Oczywiście jeśli każdy zna metodę szyfrowania danych, to musi istnieć jakaś tajna informacja, która zapobiegnie odczytaniu danych przez intruza. Tą informacją są klucze.

Na rysunku 8.2 Alicja dostarcza **klucz** K_A — łańcuch liczb lub znaków — na wejście algorytmu szyfrowania. Algorytm przyjmuje klucz oraz tekst jawny m i na ich podstawie tworzy szyfrogram. Notacja $K_A(m)$ oznacza szyfrogram tekstu jawnego m uzyskany za pomocą klucza K_A. Z kontekstu wiadomo, jaki algorytm korzysta z tego klucza. Podobnie Bartek przekazuje klucz K_B **algorytmowi deszyfrowania**, który przyjmuje ten klucz oraz szyfrogram i na ich podstawie odtwarza tekst jawny; tzn. jeśli Bartek odbierze zaszyfrowaną wiadomość $K_A(m)$, odszyfrowuje ją poprzez obliczenie $K_B(K_A(m)) = m$. W **systemach z kluczem symetrycznym** klucze Alicji i Bartka są jednakowe i zachowywane w tajemnicy. W **systemach z kluczem publicznym** używa się pary kluczy. Jeden z kluczy jest znany zarówno Bartkowi, jak i Alicji (w rzeczywistości całemu światu). Drugi klucz jest znany albo Bartkowi, albo Alicji (ale nie obojgu jednocześnie). W następnych dwóch punktach zbadamy dokładniej systemy z kluczem symetrycznym i kluczem publicznym.

8.2.1. Kryptografia z kluczem symetrycznym

Wszystkie algorytmy kryptograficzne polegają na zastępowaniu jednej rzeczy drugą, na przykład pobieraniu fragmentu tekstu jawnego oraz obliczaniu i podstawianiu odpowiedniego szyfrogramu w celu utworzenia wiadomości zaszyfrowanej. Zanim przestudiujemy współczesny system kryptograficzny oparty na kluczach, zbadajmy bardzo stary, bardzo prosty algorytm szyfrowania przypisywany Juliuszowi Cezarowi i zwany **szyfrem Cezara** (szyfr to metoda szyfrowania danych).

Szyfr Cezara działa w następujący sposób: bierzemy każdą literę tekstu jawnego i zastępujemy ją literą znajdującą się o k liter dalej w alfabecie (alfabet jest „zawinięty", tzn. po literze „*ż*" następuje litera „*a*"). Jeśli na przykład $k = 3$, litera „*a*" w tekście jawnym staje się literą „*c*" w szyfrogramie; litera „*ą*" w tekście jawnym staje się literą „*ć*" w szyfrogramie itd. Wartość k pełni tu funkcję klucza. Na przykład tekst jawny „`bartku, kocham cię. alicja`" jest przekształcany w szyfrogram „`dctymz, mrekco elh. cnlełc`". Choć szyfrogram rzeczywiście wygląda bezsensownie, złamanie kodu byłoby bardzo łatwe, ponieważ istnieje tylko 31 możliwych wartości klucza.

Ulepszeniem szyfru Cezara jest **szyfr monoalfabetyczny**, który również polega na zamianie liter. Nie używa się jednak regularnego wzorca (na przykład przesunięcia o k liter), ale dowolnych podstawień, z tym zastrzeżeniem, że każda litera musi mieć unikatowy zamiennik. Na rysunku 8.3 pokazano jedną z możliwych metod kodowania tekstu jawnego.

Litera tekstu jawnego:	a	ą	b	c	ć	d	e	ę	f	g	h	i	j	k	l	ł	m	n	ń	o	ó	p	r	s	ś	t	u	w	y	z	ź	ż
Litera szyfrogramu:	w	s	c	ż	e	a	m	b	g	ą	ź	h	ę	t	ń	d	p	r	z	i	ś	f	ć	j	ł	y	l	ó	k	o	u	n

Rysunek 8.3. Szyfr monoalfabetyczny

W tym przypadku tekst jawny „bartku, kocham cię. alicja" jest przekształcany w szyfrogram „cwćytl, tiżźwp żhb. wńhżęa". Podobnie jak w przypadku szyfru Cezara szyfrogram wygląda bezsensownie. Wydawałoby się też, że szyfr monoalfabetyczny jest znacznie lepszy od szyfru Cezara, ponieważ pozwala połączyć litery w pary na 32! sposobów (liczba rzędu 10^{35}), a nie na skromne 31. Próba złamania kodu i odszyfrowania wiadomości przez wypróbowanie wszystkich 10^{35} kombinacji zajęłaby tyle czasu, że jest z góry skazana na niepowodzenie. Jednakże analiza statystyczna języka, w którym napisano tekst jawny — na przykład ustalenie, że w typowym polskim tekście najczęściej występują litery „a" oraz „i" (obie z częstością powyżej 7%) i że niektóre dwu- i trzyliterowe sekwencje liter pojawiają się znacznie częściej niż inne (na przykład „ch", „ci", „nie" itp.) — pozwala dość łatwo złamać ten kod. Jeśli intruz dysponuje jakąś wiedzą o treści wiadomości, ma jeszcze łatwiejsze zadanie. Przykładowo, jeśli Inga jest żoną Bartka i podejrzewa, że romansuje on z Alicją, to może przypuszczać, że w tekście znajdują się słowa „bartek" lub „alicja". Gdyby Inga wiedziała na pewno, że te dwa imiona występują w szyfrogramie, mogłaby natychmiast ustalić 10 spośród 32 par liter, zmniejszając o 10^{14} liczbę możliwości do wypróbowania. Gdyby zresztą Inga podejrzewała Bartka o romans, mogłaby oczekiwać, że znajdzie w wiadomości inne specyficzne słowa.

Kiedy zastanawiamy się, jak łatwo byłoby Indze złamać szyfr używany przez Bartka i Alicję, możemy wyróżnić trzy scenariusze, w zależności od informacji, do jakich intruz ma dostęp:

- *Atak ze znanym szyfrogramem.* W niektórych przypadkach intruz ma dostęp wyłącznie do szyfrogramu i nie wie nic o zawartości tekstu jawnego. Wyjaśniliśmy już, że w **ataku ze znanym szyfrogramem** może pomóc analiza statystyczna.
- *Atak ze znanym tekstem jawnym.* Jak już stwierdziliśmy, gdyby Inga wiedziała, że w szyfrogramie występują słowa „bartek" i „alicja", mogłaby ustalić pary (tekst jawny, szyfrogram) dla liter *b*, *a*, *r*, *t*, *e*, *k*, *l*, *i*, *c* oraz *j*. Mogłaby również zarejestrować wszystkie przesyłane szyfrogramy, a później odnaleźć odszyfrowaną wersję któregoś z nich zapisaną przez Bartka na kartce papieru. Kiedy intruz zna niektóre pary (tekst jawny, szyfrogram), mówimy o **ataku ze znanym tekstem jawnym**.
- *Atak z wybranym tekstem jawnym.* W **ataku z wybranym tekstem jawnym** intruz może wybrać dowolny tekst jawny i uzyskać jego szyfrogram. W przypadku prostych algorytmów, które omawialiśmy do tej pory, Inga mogłaby nakłonić Alicję do wysłania wiadomości „Pchnąć w tę łódź jeża lub ośm skrzyń fig" i całkowicie złamać szyfr. Niebawem pokażemy, że w bardziej zaawansowanych technikach szyfrowania atak z wybranym tekstem jawnym nie zawsze pozwala złamać szyfr.

Pięćset lat temu wymyślono techniki szyfrowania **polialfabetycznego**. Polegają one na użyciu wielu szyfrów monoalfabetycznych, przy czym określony szyfr monoalfabetyczny służy do kodowania litery o określonym położeniu w tekście jawnym. Zatem ta sama litera może być kodowana na różne sposoby, w zależności od pozycji, jaką zajmuje w tekście jawnym. Przykład szyfru polialfabetycznego pokazano na rysunku 8.4. Składa się on z dwóch szyfrów Cezara (o parametrach $k = 5$ i $k = 19$). Moglibyśmy zdecydować, że będziemy używać tych dwóch szyfrów w powtarzającej się sekwencji C_1, C_2, C_2, C_1, C_2. Pierwsza litera tekstu jawnego byłaby kodowana za pomocą szyfru C_1, druga i trzecia — za pomocą szyfru C_2, czwarta — za pomocą szyfru C_1, a piąta — za pomocą szyfru C_2. Następnie wzór powtarzałby się — szósta litera byłaby kodowana za pomocą szyfru C_1, siódma — za pomocą szyfru C_2 itd. Tekst jawny „`bartku, kocham cię. alicja`" zostałby zatem zaszyfrowany do postaci „`ęogźak, aefzdc rmu. dąźfnd`". Zauważmy, że pierwsza litera „a" tekstu jawnego jest zakodowana za pomocą szyfru C_1, a druga litera „a" — za pomocą C_2. W tym przykładzie „kluczem" szyfrowania i deszyfrowania jest znajomość dwóch kluczy Cezara ($k = 5$, k = *19*) oraz wzoru C_1, C_2, C_2, C_1, C_2.

Litera tekstu jawnego:	a	ą	b	c	ć	d	e	ę	f	g	h	i	j	k	l	ł	m	n	ń	o	ó	p	r	s	ś	t	u	w	y	z	ź	ż
$C_1(k = 5)$:	d	e	ę	f	g	h	i	j	k	l	ł	m	n	ń	o	ó	p	r	s	ś	t	u	w	y	z	ź	ż	a	ą	b	c	ć
$C_2(k = 19)$:	o	ó	p	r	s	ś	t	u	w	y	z	ź	ż	a	ą	b	c	ć	d	e	ę	f	g	h	i	j	k	l	ł	m	n	ń

Rysunek 8.4. Szyfr polialfabetyczny złożony z dwóch szyfrów Cezara

Szyfry blokowe

Przejdźmy teraz do współczesności i zbadajmy, jak obecnie wygląda szyfrowanie z kluczem symetrycznym. Istnieją dwie ogólne klasy technik szyfrowania z kluczem symetrycznym — **szyfry strumieniowe** i **szyfry blokowe**. Szyfry strumieniowe opisujemy pokrótce w podrozdziale 8.7, gdzie badamy zabezpieczenia bezprzewodowych sieci lokalnych. W tym miejscu skoncentrujemy się na szyfrach blokowych, które są stosowane w wielu bezpiecznych protokołach internetowych, w tym w PGP (do zabezpieczenia poczty elektronicznej), SSL (do zabezpieczania połączeń TCP) i IPsec (do zabezpieczania transportu w warstwie sieci).

W szyfrach blokowych szyfrowany komunikat jest przetwarzany w blokach po k bitów. Na przykład jeśli $k = 64$, komunikat jest dzielony na 64-bitowe bloki, a każdy z nich jest szyfrowany niezależnie. Aby zakodować blok, algorytm szyfrujący stosuje odwzorowanie jeden do jednego i przekształca k-bitowy blok tekstu jawnego na k-bitowe bloki szyfrogramu. Przyjrzyjmy się przykładowi. Załóżmy, że $k = 3$, więc szyfr blokowy odwzorowuje 3-bitowe dane wejściowe (tekst jawny) na 3-bitowe dane wyjściowe (szyfrogram). Jedno z możliwych odwzorowań przedstawia tabela 8.1. Warto zauważyć, że jest to odwzorowanie jeden do jednego. Oznacza to, że dla każdych danych wejściowych tworzone

Wejście	Wyjście	Wejście	Wyjście
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

Tabela 8.1. Określony 3-bitowy szyfr blokowy

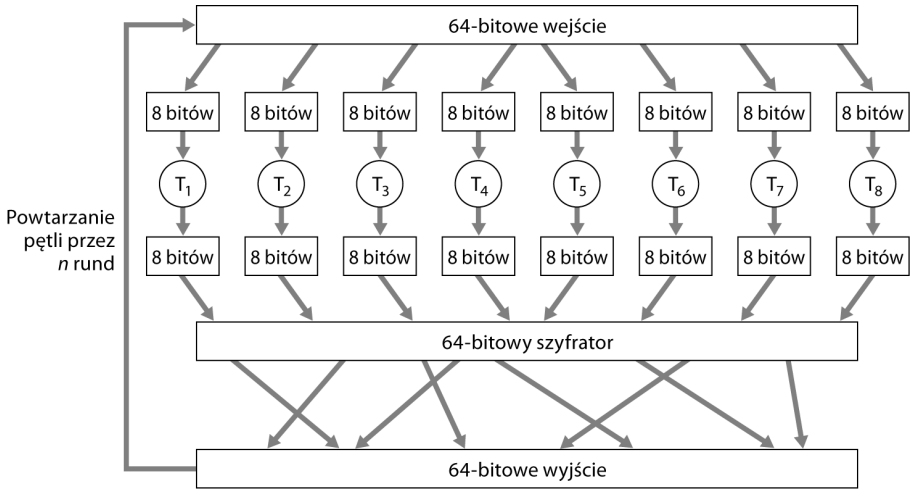
są inne dane wyjściowe. Ten szyfr blokowy dzieli komunikat na 3-bitowe bloki i szyfruje każdy z nich na podstawie powyższego odwzorowania. Czytelnik powinien sprawdzić, że komunikat 010110001111 zostanie zaszyfrowany do postaci 101000111001.

Kontynuujmy przykład z 3-bitowymi blokami. Zauważmy, że odwzorowanie z tabeli 8.1 to tylko jedno z wielu możliwych rozwiązań. Ile różnych odwzorowań istnieje? Aby odpowiedzieć na to pytanie, warto zauważyć, że odwzorowanie jest niczym więcej jak permutacją wszystkich możliwych danych wejściowych. Tych danych jest 2^3, czyli 8 (są one wymienione w kolumnie danych wejściowych). Dla tych ośmiu możliwości istnieje 8! = 40 320 różnych permutacji. Ponieważ każda z nich określa odwzorowanie, tych ostatnich też jest 40 320. Każde odwzorowanie można traktować jako klucz. Jeśli Alicja i Bartek znają odwzorowanie (klucz), mogą szyfrować i odszyfrowywać przesyłane komunikaty.

Atak typu brute force na ten szyfr polega na próbie odszyfrowania szyfrogramu za pomocą każdego odwzorowania. Przy tylko 40 320 odwzorowaniach (dla $k = 3$) można to szybko zrobić na komputerze PC. Aby zapobiec atakom brute force, w szyfrach blokowych zwykle stosuje się dużo większe bloki — z $k = 64$ lub nawet więcej. Zauważmy, że liczba możliwych odwzorowań dla ogólnego szyfru o bloku o wielkości k wynosi $2^k!$, co daje bardzo dużą wartość nawet dla umiarkowanych wartości k (na przykład $k = 64$).

Choć szyfry blokowe z kompletną tabelą, takie jak opisany wcześniej, już przy niewielkich wartościach k pozwalają opracować solidne systemy szyfrowania z kluczem symetrycznym, są niestety trudne w stosowaniu. Dla $k = 64$ i określonego odwzorowania Alicja i Bartek muszą przechowywać tabelę o 2^{64} wartościach wejściowych. Jest to niewykonalne zadanie. Ponadto gdyby Alicja i Bartek mieli wymieniać klucze, musieliby odtwarzać tę tabelę. Dlatego szyfry blokowe z kompletną tabelą, określające wstępnie ustalone odwzorowanie między wszystkimi wartościami wejściowymi i wyjściowymi (jak we wcześniejszym przykładzie), są nieakceptowalne.

Zamiast tego w szyfrach blokowych zwykle wykorzystywane są funkcje symulujące tabele utworzone w wyniku losowej permutacji. Przykład (zaadaptowany z [Kaufman 1995]) takiej funkcji dla $k = 64$ bity przedstawia rysunek 8.5.



Rysunek 8.5. Przykładowy szyfr blokowy

Ta funkcja najpierw dzieli 64-bitowy blok na osiem 8-bitowych fragmentów. Każdy z tych fragmentów jest przetwarzany za pomocą możliwej do obsłużenia tabeli o wielkości osiem na osiem bitów. Na przykład do przetwarzania pierwszego fragmentu służy tabela T_1. Następnie osiem wyjściowych fragmentów jest łączonych w 64-bitowy blok. System zamienia wtedy pozycje 64 bitów w tym bloku, aby wygenerować 64-bitowe dane wyjściowe. Są one ponownie stosowane jako 64-bitowe dane wejściowe, co zaczyna następny cykl. Po *n* takich cyklach funkcja zwraca 64-bitowy blok szyfrogramu. Celem stosowania rund jest sprawienie, aby każdy bit wejściowy oddziaływał na większość (jeśli nie wszystkie) z ostatecznie uzyskanych bitów wyjściowych. Gdyby uruchomiono tylko jedną rundę, bit wejściowy wpływałby na tylko osiem z 64 bitów wyjściowych. Kluczem w takim algorytmie szyfrowania blokowego jest osiem tabel permutacji (zakładamy, że funkcja mieszająca bity jest powszechnie znana).

Obecnie istnieje wiele popularnych szyfrów blokowych, w tym DES (ang. *Data Encryption Standard*), 3DES i AES (ang. *Advanced Encryption Standard*). Każdy z tych standardów wykorzystuje funkcje zamiast wstępnie przygotowanych tabel, jak przedstawia to rysunek 8.5 (choć poszczególne szyfry są bardziej skomplikowane). Każdy z algorytmów używa też łańcucha bitów jako klucza. Na przykład DES wykorzystuje 64-bitowe bloki z 56-bitowym kluczem. AES używa 128-bitowych bloków i może działać z kluczami o długości 128, 192 i 256 bitów. Klucz algorytmu określa konkretne odwzorowania i permutacje w postaci „minitabeli". Atak brute force na każdy z tych szyfrów polega na cyklicznym sprawdzaniu wszystkich kluczy i stosowaniu ich w algorytmie deszyfrującym. Zauważmy, że przy kluczu o długości *n* istnieje 2^n możliwych kluczy. W instytucie NIST [NIST 2001] oszacowano, że maszynie, która potrafi złamać

56-bitowy szyfr DES w jedną sekundę (czyli zdoła wypróbować w tym czasie każdy z 2^{56} kluczy), złamanie 128-bitowego klucza AES zajęłoby około 149 trylionów lat.

Wiązanie bloków

W aplikacjach sieciowych zwykle trzeba szyfrować długie komunikaty (lub długie strumienie danych). Jeśli zastosujemy szyfr blokowy przez prosty podział wiadomości na k-bitowe bloki i niezależne zaszyfrowanie każdego z nich, wystąpi subtelny, ale istotny problem. Aby go dostrzec, należy zauważyć, że bloki z tekstem jawnym mogą być identyczne. Na przykład tekst jawny w kilku blokach może brzmieć „HTTP/1.1". Dla takich identycznych bloków szyfr blokowy utworzy oczywiście takie same szyfrogramy. Napastnik może odgadnąć tekst jawny, jeśli zauważy identyczne bloki w szyfrogramie. Może nawet zdołać odszyfrować całą wiadomość przez zidentyfikowanie takich samych bloków i wykorzystanie wiedzy na temat struktury zastosowanego protokołu [Kaufman 1995].

Aby rozwiązać ten problem, należy wprowadzić w proces losowość, aby identyczne bloki tekstu jawnego prowadziły do utworzenia odmiennych szyfrogramów. W celu wyjaśnienia tego pomysłu przyjmijmy, że $m(i)$ oznacza i-ty blok tekstu jawnego, $c(i)$ to i-ty blok szyfrogramu, a $a \oplus b$ oznacza operację XOR na dwóch łańcuchach bitów (a i b). Przypomnijmy, że $0 \oplus 0 = 1 \oplus 1 = 0$ i $0 \oplus 1 = 1 \oplus 0 = 1$, a suma XOR dwóch łańcuchów bitów jest obliczana bit po bicie, na przykład $10101010 \oplus 11110000 = 01011010$. Ponadto oznaczmy algorytm szyfru blokowego z kluczem S jako K_S. Podstawowa reguła wygląda tak — nadawca tworzy losową k-bitową liczbę $r(i)$ dla i-tego bloku i oblicza $c(i) = K_S(m(i) \oplus r(i))$. Zauważmy, że dla każdego bloku wybierana jest nowa k-bitowa liczba. Następnie nadawca wysyła wartości $c(1)$, $r(1)$, $c(2)$, $r(2)$, $c(3)$, $r(3)$ itd. Ponieważ odbiorca otrzymuje $c(i)$ i $r(i)$, może odtworzyć każdy blok tekstu jawnego przez obliczenia $m(i) = K_S(m(i)) \oplus r(i)$. Należy zauważyć, że choć wartość $r(i)$ jest przesyłana jako tekst jawny i może zostać zarejestrowana przez Ingę, nie zdoła ona uzyskać tekstu jawnego $m(i)$, ponieważ nie zna klucza K_S. Ponadto nawet jeśli dwa bloki tekstu jawnego $m(i)$ i $m(j)$ są identyczne, odpowiadające im bloki szyfrogramu $c(i)$ i $c(j)$ różnią się od siebie (o ile liczby losowe $r(i)$ i $r(j)$ są inne, co jest wysoce prawdopodobne).

Rozważmy na przykład 3-bitowy szyfr blokowy z tabeli 8.1. Załóżmy, że tekst jawny to 010010010. Jeśli Alicja zaszyfruje go bezpośrednio (bez wprowadzania losowości), wynikowy szyfrogram będzie miał wartość 101101101. Jeżeli Inga zarejestruje ten szyfrogram, to ze względu na to, że trzy bloki szyfrogramu są identyczne, będzie mogła słusznie przyjąć, że każdy z trzech bloków tekstu jawnego jest taki sam. Teraz załóżmy, że Alicja wygeneruje losowe bloki $r(1) = 001$, $r(2) = 111$ i $r(3) = 100$ oraz zastosuje opisaną wcześniej technikę do utworzenia szyfrogramu $c(1) = 100$, $c(2) = 010$ i $c(3) = 000$. Zauważmy, że trzy bloki

szyfrogramu różnią się od siebie, choć bloki tekstu jawnego są takie same. Alicja wyśle wtedy dane $c(1)$, $r(1)$, $c(2)$ i $r(2)$. Czytelnik powinien sprawdzić, że Bartek zdoła uzyskać pierwotny tekst jawny za pomocą wspólnego klucza K_S.

Uważni Czytelnicy zauważą, że wprowadzenie losowości rozwiązuje jeden problem, ale wprowadza inny. Alicja musi teraz przesyłać dwa razy więcej bitów. Dla każdego bitu szyfrogramu musi wysłać także bit losowy, co podwaja potrzebną przepustowość. Aby można było „zjeść ciastko i mieć ciastko", w szyfrach blokowych zwykle stosuje się technikę **wiązania bloków** (ang. *Cipher Block Chaining* — CBC). Podstawowa zasada jej działania polega na *wysyłaniu tylko jednej losowej wartości z pierwszym komunikatem, po czym nadawca i odbiorca mogą wykorzystać obliczone zakodowane bloki zamiast kolejnych liczb losowych*. Opiszmy dokładniej funkcjonowanie techniki wiązania bloków:

1. Przed zaszyfrowaniem komunikatu (lub strumienia danych) nadawca generuje losowy k-bitowy łańcuch nazywany **wektorem inicjującym** (ang. *Initialization Vector* — IV). Oznaczmy ten wektor przez $c(0)$. Nadawca wysyła do odbiorcy wektor IV jako *tekst jawny*.
2. Dla pierwszego bloku nadawca oblicza wartość $m(1) \oplus c(0)$, czyli sumę XOR pierwszego bloku tekstu jawnego i wektora IV. Następnie przekazuje wynik do algorytmu szyfru blokowego, aby uzyskać odpowiedni blok szyfrogramu ($c(1) = K_S(m(1) \oplus c(0))$. Nadawca wysyła zaszyfrowany blok $c(1)$ do odbiorcy.
3. Dla i-tego bloku nadawca generuje i-ty blok szyfrogramu $c(i) = K_S(m(i) \oplus c(i-1))$.

Zbadajmy teraz skutki zastosowania tego podejścia. Po pierwsze, odbiorca nadal może odtworzyć pierwotną wiadomość. Kiedy otrzyma szyfrogram $c(i)$, odszyfruje go za pomocą klucza K_S, aby uzyskać $s(i) = m(i) \oplus c(i-1)$. Ponieważ odbiorca zna również wartość $c(i-1)$, może otrzymać blok tekstu jawnego na podstawie obliczeń $m(i) = s(i) \oplus c(i-1)$. Po drugie, nawet jeśli dwa bloki tekstu jawnego są identyczne, odpowiadające im szyfrogramy prawie zawsze będą różne. Po trzecie, choć nadawca wysyła wektor IV jako tekst jawny, intruz nie będzie mógł odszyfrować bloków szyfrogramu, ponieważ nie zna tajnego klucza S. Wreszcie nadawca wysyła tylko jeden dodatkowy blok (wektor IV), dlatego dla długich komunikatów, składających się z setek bloków, wzrost potrzebnej przepustowości jest pomijalny.

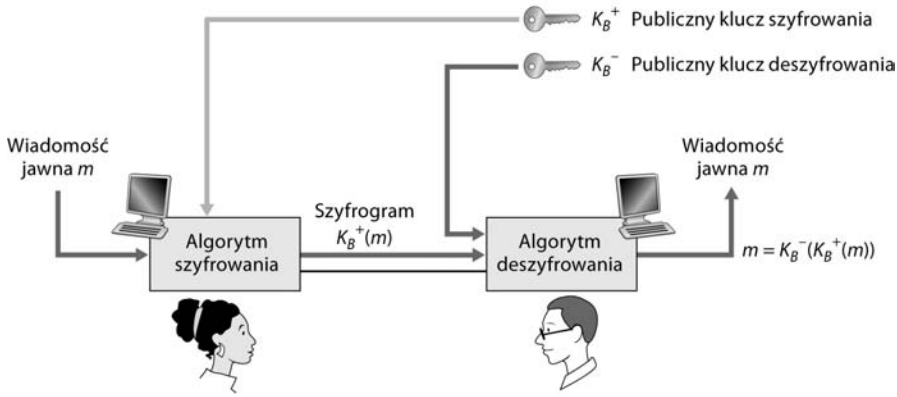
W ramach przykładu obliczmy na podstawie 3-bitowego szyfru blokowego z tabeli 8.1 szyfrogram dla tekstu jawnego 010010001 i wektora IV = $c(0) = 001$. Nadawca najpierw używa wektora IV do obliczenia $c(1) = K_S(m(1) \oplus c(0)) = 100$, a następnie oblicza $c(2) = K_S(m(2) \oplus c(1)) = K_S(010 \oplus 100) = 000$ i $c(3) = K_S(m(3) \oplus c(2)) = K_S(010 \oplus 000) = 101$. Czytelnik powinien sprawdzić, że odbiorca — znając wartość wektora IV i klucz K_S — zdoła odtworzyć pierwotny tekst jawny.

Wiązanie bloków ma poważny wpływ na projektowanie bezpiecznych protokołów sieciowych. Trzeba w nich udostępnić mechanizm do przekazywania wektora IV od nadawcy do odbiorcy. W dalszej części rozdziału pokażemy, jak odbywa się to w kilku protokołach.

8.2.2. Szyfrowanie z kluczem publicznym

Przez ponad 2000 lat (od wynalezienia szyfru Cezara do lat 70. XX wieku) zaszyfrowana komunikacja wymagała, aby obie jej strony znały wspólny sekret — symetryczny klucz używany do szyfrowania i deszyfrowania. Problem w tym, że obie strony muszą jakoś uzgodnić wspólny klucz, co jednak wymaga (zapewne *bezpiecznej*) komunikacji! Strony mogłyby najpierw spotkać się osobiście i uzgodnić klucz (na przykład dwóch centurionów Cezara mogłoby spotkać się w rzymskiej łaźni), a potem porozumiewać się przy użyciu szyfru. Jednak w połączonym siecią świecie często zdarza się, że strony nigdy się nie spotykają i porozumiewają się wyłącznie przez sieć. Czy strony mogą się bezpiecznie komunikować bez tajnego, uprzednio uzgodnionego klucza? W 1976 roku Diffie i Hellman [Diffie 1976] zademonstrowali algorytm (obecnie znany jako wymiana kluczy Diffiego-Hellmana) opracowany właśnie w tym celu — radykalnie odmienną i niezwykle elegancką metodę bezpiecznej komunikacji, która doprowadziła do powstania współczesnych systemów kryptograficznych z kluczem publicznym. Systemy kryptograficzne z kluczem publicznym mają kilka innych zdumiewających cech, które sprawiają, że nadają się one nie tylko do szyfrowania, ale także do uwierzytelniania i podpisów cyfrowych. Co ciekawe, niedawno wyszło na jaw, że idee podobne do opisanych w [Diffie 1976] i [RSA 1978] zostały również przedstawione w serii tajnych raportów badaczy z Communications-Electronics Security Group w Wielkiej Brytanii [Ellis 1987]. Jak to często bywa, doskonałe pomysły pojawiają się niezależnie w wielu miejscach; na szczęście postępy w tej dziedzinie nie pozostały w ukryciu, lecz stały się własnością publiczną.

Koncepcja szyfrowania z kluczem publicznym jest bardzo prosta. Przypuśćmy, że Alicja chce porozumieć się z Bartkiem. Jak pokazano na rysunku 8.6, nie współdzielą oni jednego tajnego klucza (jak w systemach z kluczem symetrycznym). Bartek (odbiorca wiadomości Alicji) ma dwa klucze — **klucz publiczny** dostępny dla *każdego* (łącznie z Ingą, intruzem) oraz **klucz prywatny** znany tylko Bartkowi. Publiczny i prywatny klucz Bartka będziemy oznaczać symbolami (odpowiednio) K_B^+ i K_B^-. Aby porozumieć się z Bartkiem, Alicja najpierw zdobywa jego klucz publiczny, po czym szyfruje swoją wiadomość m przy użyciu tego klucza oraz znanego (na przykład ustandaryzowanego) algorytmu szyfrowania; oznacza to, że Alicja oblicza $K_B^+(m)$. Bartek odbiera wiadomość Alicji i używa znanego (na przykład ustandaryzowanego) algorytmu deszyfrowania, aby odtworzyć oryginalną wiadomość. Oznacza to, że



Rysunek 8.6. Kryptografia z kluczem publicznym

Bartek oblicza $K_B^-(K_B^+(m))$. Istnieją jednak takie algorytmy szyfrowania-deszyfrowania oraz metody wyboru klucza prywatnego i publicznego, że $K_B^-(K_B^+(m)) = m$, tzn. zastosowanie publicznego klucza Bartka K_B^+ do wiadomości m (w celu uzyskania $K_B^+(m)$), a następnie zastosowanie prywatnego klucza Bartka K_B^- do zaszyfrowanej wersji m (tzn. obliczenie $K_B^-(K_B^+(m))$) daje z powrotem m. Jest to godne uwagi osiągnięcie! W ten sposób Alicja może użyć publicznie dostępnego klucza Bartka, aby wysłać do niego tajną wiadomość, a żadne z nich nie musi przekazywać drugiemu tajnego klucza! Niebawem przekonamy się, że możemy zamienić klucz prywatny z publicznym i osiągnąć ten sam zdumiewający wynik, tzn. $K_B^-(K_B^+(m)) = K_B^+(K_B^-(m)) = m$.

Zasady kryptografii z kluczem publicznym są więc nieskomplikowane. Od razu przychodzą jednak na myśl dwa problemy. Po pierwsze, intruz, który przechwyci wiadomość Alicji, znajdzie w niej tylko „śmieci", ale zna zarówno klucz (publiczny klucz Bartka, dostępny dla każdego), jak i algorytm używany przez Alicję. Intruz może więc przeprowadzić atak z wybranym tekstem jawnym, korzystając ze standardowego algorytmu szyfrowania i publicznego klucza Bartka w celu zaszyfrowania dowolnej wiadomości! Intruz może na przykład szyfrować wiadomości (albo fragmenty) wiadomości, które — jak podejrzewa — Alicja mogłaby przesyłać Bartkowi. Jak widać, jeśli kryptografia z kluczem publicznym ma działać, dobór kluczy oraz szyfrowanie i deszyfrowanie muszą odbywać się w taki sposób, aby ustalenie prywatnego klucza Bartka albo odszyfrowanie wiadomości Alicji było niemożliwe (albo tak trudne, że praktycznie niemożliwe). Po drugie, ponieważ klucz Bartka jest publiczny, każdy może wysłać do niego wiadomość, *podając się* za Alicję. W algorytmach z pojedynczym wspólnym kluczem fakt, że nadawca zna tajny klucz, jednoznacznie identyfikuje go przed odbiorcą. W przypadku kryptografii z kluczem publicznym jest inaczej, ponieważ każdy może wysłać wiadomość do Bartka, korzystając z jego publicznie dostępnego klucza. Aby powiązać nadawcę z wiadomością, potrzebny jest podpis cyfrowy; zagadnieniem tym zajmiemy się w podrozdziale 8.3.

RSA

Choć można opracować wiele algorytmów i kluczy, które rozwiązywałyby te problemy, synonimem kryptografii z kluczem publicznym stał się **algorytm RSA** (skrót pochodzi od nazwisk jego twórców, Rona Rivesta, Adiego Shamira i Leonarda Adlemana). Zobaczmy najpierw, jak działa RSA, a następnie zastanówmy się, dlaczego działa.

Algorytm RSA wykonuje wiele operacji arytmetycznych z wykorzystaniem arytmetyki modulo-n. Opiszmy pokrótce arytmetykę modularną. Przypomnijmy, że x mod n oznacza po prostu resztę z dzielenia x przez n. Na przykład 19 mod 5 = 4. W arytmetyce modularnej wykonywane są normalne działania dodawania, mnożenia i potęgowania. Jednak wynik każdej operacji jest zastępowany przez całkowitoliczbową resztę z dzielenie go przez n. Dodawanie i mnożenie w arytmetyce modularnej ułatwiają poniższe wygodne zależności:

$$[(a \bmod n)+(b \bmod n)] \bmod \mathrm{n} = (a+b) \bmod n$$
$$[(a \bmod n)-(b \bmod n)] \bmod \mathrm{n} = (a-b) \bmod n$$
$$[(a \bmod n)\cdot(b \bmod n)] \bmod \mathrm{n} = (a\cdot b) \bmod n$$

Z trzeciej zależności wynika, że $(a \bmod n)^d \bmod n = a^d \bmod n$. Ta tożsamość wkrótce okaże się bardzo przydatna.

Przypuśćmy, że Alicja chce wysłać do Bartka komunikat zaszyfrowany algorytmem RSA, co ilustruje rysunek 8.6. Przy analizie tego algorytmu zawsze warto pamiętać o tym, że wiadomość to nic więcej jak wzorzec bitów, a każdy wzorzec ma niepowtarzalną reprezentację w postaci liczby całkowitej (i długości wzorca). Załóżmy na przykład, że komunikat to wzorzec bitów 1001. Można go przedstawić jako dziesiętną liczbę całkowitą 9. Dlatego szyfrowanie wiadomości za pomocą algorytmu RSA odpowiada szyfrowaniu niepowtarzalnych liczb całkowitych reprezentujących komunikat.

RSA ma dwa powiązane ze sobą aspekty:

- wybór klucza publicznego i klucza prywatnego;
- algorytm szyfrowania i deszyfrowania.

Aby wybrać klucz publiczny i prywatny, Bartek musi wykonać następujące czynności:

1. Wybrać dwie duże liczby pierwsze p i q. Jak duże powinny być te liczby? Im są większe, tym trudniej złamać RSA, ale tym dłużej trwa kodowanie i dekodowanie. Firma RSA Laboratories zaleca, aby iloczyn p i q miał długość rzędu 1024 bitów. Sposoby znajdowania dużych liczb pierwszych można znaleźć w [Caldwell 2007].
2. Obliczyć $n = pq$ oraz $z = (p-1)(q-1)$.

3. Wybrać liczbę *e* mniejszą niż *n*, która nie ma wspólnych dzielników (poza jedynką) z liczbą *z* (w takim przypadku mówi się, że liczby *e* i *z* są względnie pierwsze). Użyto litery *e*, ponieważ wartość ta będzie służyć do szyfrowania (ang. *encryption*).
4. Znaleźć liczbę *d* taką, że *ed*–1 dzieli się bez reszty przez *z*. Użyto litery *d*, ponieważ wartość ta będzie służyć do deszyfrowania. Innymi słowy, dysponując wartością *e*, wybieramy liczbę *d* taką, aby:

$$ed \bmod z = 1$$

5. Klucz publiczny K_B^+, który Bartek udostępnia całemu światu, jest parą liczb (*n*, *e*); klucz prywatny K_B^- jest parą liczb (*n*, *d*).

Alicja szyfruje, a Bartek odszyfrowuje wiadomość w następujący sposób:

- Przypuśćmy, że Alicja chce wysłać Bartkowi wzorzec bitowy reprezentowany przez liczbę całkowitą *m* taką, że $m < n$. Aby ją zakodować, Alicja oblicza m^e, a następnie oblicza resztę z dzielenia m^e przez *n*. Zatem zaszyfrowana wartość *c* tekstu jawnego *m* wysyłana przez Alicję to:

$$c = m^e \bmod n$$

  Wzorzec bitowy odpowiadający szyfrogramowi (*c*) jest wysyłany do Bartka.
- Aby odszyfrować odebrany szyfrogram *c*, Bartek oblicza:

$$m = c^d \bmod n$$

  co wymaga użycia jego klucza prywatnego (*n*, *d*).

Rozważmy prosty przykład użycia RSA. Przypuśćmy, że Bartek wybiera $p = 5$ i $q = 7$ (oczywiście liczby te są zbyt małe, aby były bezpieczne). W takim przypadku $n = 35$, a $z = 24$. Bartek wybiera $e = 5$, ponieważ 5 i 24 nie mają wspólnych dzielników. Wreszcie Bartek wybiera $d = 29$, ponieważ $5 \cdot 29 - 1$ (tzn. *ed*–1) dzieli się bez reszty przez 24. Bartek upublicznia te dwie wartości ($n = 35$ i $e = 5$), ale nie ujawnia wartości $d = 29$. Znając dwie publiczne wartości, Alicja chce wysłać do Bartka litery „*l*", „*o*", „*v*" i „*e*". Interpretując każdą z nich jako wartość z zakresu 1 – 26 („*a*" to 1, „*z*" to 26), Alicja i Bartek szyfrują i deszyfrują wiadomość w sposób przedstawiony w tabelach 8.2 i 8.3. Zauważmy, że w tym przykładzie każda z czterech liter jest traktowana jak odrębny komunikat. Bardziej realistyczne rozwiązanie wymaga przekształcenia liter na ich 8-bitowe reprezentacje w formacie ASCII i zaszyfrowanie liczby całkowitej odpowiadającej uzyskanemu 32-bitowemu wzorcowi. Jednak taki realistyczny przykład prowadzi do uzyskania liczb, które są zdecydowanie zbyt długie, aby można umieścić je w podręczniku!

Litera tekstu jawnego	m: reprezentacja liczbowa	m^e	szyfrogram $c = m^e \bmod n$
l	12	238832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

Tabela 8.2. Szyfrowanie RSA Alicji, e = 5, n = 35

Szyfrogram c c	c^d	$m = c^d \bmod n$	Litera tekstu jawnego
17	481968572106750915091411825223071697	12	l
15	12783403948858939111232757568359375	15	o
22	851643319086537701956194499721106030592	22	v
10	10000000000000000000000000000000	5	e

Tabela 8.3. Deszyfrowanie RSA Bartka, d = 29, n = 35

Zważywszy, że już „dziecinny" przykład z tabel 8.2 i 8.3 doprowadził do powstania bardzo dużych liczb, a wcześniej dowiedzieliśmy się, że wartości p i q powinny liczyć kilkaset bitów, trzeba zastanowić się nad praktycznymi aspektami RSA. Jak wybiera się duże liczby pierwsze? Jak następnie wybiera się e i d? Jak wykonuje się potęgowanie, kiedy wykładnikami są duże liczby? Omówienie tych ważnych kwestii wykracza poza ramy niniejszej książki; szczegóły można znaleźć w [Kaufman 1995] oraz wymienionych tam źródłach.

Klucze sesji

Warto wiedzieć, że potęgowanie wymagane przez RSA jest dość czasochłonne. Algorytm DES jest przynajmniej 100 razy szybszy w implementacji programowej i od 1000 do 10 000 razy szybszy w implementacji sprzętowej [RSA Fast 2007]. W rezultacie algorytmu RSA często używa się w połączeniu szyframi z kluczem symetrycznym. Jeśli na przykład Alicja chce szybko przesłać Bartkowi dużą ilość zaszyfrowanych danych, robi to w następujący sposób: najpierw wybiera klucz, który posłuży do szyfrowania danych; czasem określa się go mianem **klucza sesji**, K_S. Alicja musi przekazać Bartkowi klucz sesji, ponieważ jest to wspólny klucz symetryczny używany przez szyfr z takim kluczem (na przykład DES lub AES). Zatem Alicja szyfruje klucz sesji publicznym kluczem RSA Bartka, tzn. oblicza $c = (K_S)^e \bmod n$. Bartek otrzymuje zaszyfrowany za pomocą RSA klucz sesji c i odszyfrowuje go, aby uzyskać klucz sesji K_S. Teraz Bartek zna klucz sesji, którego Alicja będzie używać do transferu danych zaszyfrowanych za pomocą DES.

Dlaczego RSA działa?

Szyfrowanie i deszyfrowanie RSA wydaje się niemal magiczne. Dlaczego zastosowanie opisanych wyżej algorytmów pozwala odtworzyć oryginalną wiadomość? Aby zrozumieć, jak działa RSA, przyjmiemy $n = pq$, gdzie p i q są dużymi liczbami pierwszymi używanymi w algorytmie RSA.

Przypomnijmy, że w szyfrowaniu RSA wiadomość m (reprezentowana przez liczbę całkowitą) jest najpierw podnoszona do potęgi e z wykorzystaniem arytmetyki modulo n:

$$c = m^e \bmod n$$

Deszyfrowanie polega na podniesieniu tej wartości do potęgi d, znów z wykorzystaniem arytmetyki modulo n. Wynik szyfrowania, po którym następuje operacja deszyfrowania, można zatem zapisać jako $(m^e \bmod n)^d \bmod n$. Zobaczmy, co możemy powiedzieć o tej wartości. Jak wcześniej wspomnieliśmy, jedną z ważnych cech arytmetyki modularnej jest, że $(a \bmod n)^d \bmod n = a^d \bmod n$ dla dowolnych wartości a, n i d. Dlatego, wykorzystując zależność $a = m^e$ z tej właściwości, otrzymujemy:

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

Pozostaje więc wykazać, że $m^{ed} \bmod n = m$. Choć próbujemy wyeliminować magię z algorytmu RSA, będziemy musieli teraz posłużyć się dość magicznym twierdzeniem z teorii liczb. Mówiąc ściślej, udowodniono, że jeśli p i q są liczbami pierwszymi, a $n = pq$ i $z = (p - 1)(q - 1)$, to $x^y \bmod n$ jest równe $x^{(y \bmod z)} \bmod n$ [Kaufman 1995]. Stosując to twierdzenie dla $x = m$ i $y = ed$:

$$m^{ed} \bmod n = m^{(ed \bmod z)} \bmod n$$

Jak jednak wiemy, wybraliśmy takie wartości e i d, że $ed \bmod z = 1$. Otrzymujemy więc:

$$m^{ed} \bmod n = m^1 \bmod n = m$$

Właśnie takiego wyniku oczekiwaliśmy! Najpierw podnosząc wartość do potęgi e (tzn. szyfrując), a następnie podnosząc ją do potęgi d (tzn. deszyfrując), otrzymaliśmy pierwotną wartość m. Jeszcze *bardziej* zdumiewające jest to, że najpierw podnosząc wartość do potęgi d, a później do potęgi e — tzn. odwracając kolejność szyfrowania i deszyfrowania — również uzyskujemy pierwotną wartość m! Te wspaniałe zależności wynikają bezpośrednio z arytmetyki modularnej:

$$(m^d \bmod n)^e \bmod n = m^{de} \bmod n = m^{ed} \bmod n = (m^e \bmod n)^d \bmod n$$

Bezpieczeństwo RSA wynika z tego, że nie są znane żadne szybkie algorytmy dzielenia liczby — w tym przypadku publicznej wartości n — na czynniki

pierwsze p i q. Gdybyśmy potrafili ustalić wartości p i q, to znając publiczną wartość e, moglibyśmy łatwo obliczyć tajny klucz d. Z drugiej strony, nie wiadomo, czy *nie istnieje* jakiś szybki algorytm dzielenia liczb na czynniki pierwsze, więc w tym sensie bezpieczeństwo RSA nie jest gwarantowane.

Innym popularnym algorytmem szyfrowania z kluczem publicznym jest algorytm Diffiego-Hellmana, który pokrótce zbadamy w jednym z problemów w końcowej części rozdziału. Algorytm ten nie jest równie wszechstronny jak RSA, ponieważ nie można go stosować do szyfrowania wiadomości o dowolnej długości. Można go jednak wykorzystać do utworzenia symetrycznego klucza sesji, który z kolei posłuży do szyfrowania komunikatów.

8.3. Integralność komunikatów i uwierzytelnianie punktów końcowych

W poprzednim podrozdziale Czytelnicy zobaczyli, jak zastosować szyfrowanie do zapewniania poufności komunikacji między dwoma jednostkami. W tym miejscu skoncentrujemy się na równie ważnym aspekcie kryptografii — gwarantowaniu **integralności komunikatów** (jest to tak zwane uwierzytelnianie wiadomości). Obok integralności komunikatów w tym podrozdziale omówimy dwa powiązane zagadnienia — podpisy cyfrowe i uwierzytelnianie punktów końcowych.

Do zdefiniowania problemu zachowania integralności komunikatów ponownie posłużmy się postaciami Alicji i Bartka. Załóżmy, że Bartek otrzymał wiadomość (zaszyfrowaną lub w formie tekstu jawnego) i wierzy, iż jej nadawcą jest Alicja. Aby uwierzytelnić komunikat, Bartek musi zweryfikować, że:

- wiadomość rzeczywiście została wysłana przez Alicję;
- wiadomość nie została zmodyfikowana po drodze do Bartka.

W podrozdziałach od 8.4 do 8.7 pokażemy, że problem integralności komunikatów jest kluczowy w niemal wszystkich bezpiecznych protokołach sieciowych.

Jako konkretny przykład rozważmy sieć komputerową, w której algorytm routingu stanu łącza (na przykład OSPF) określa trasy między wszystkimi parami routerów w sieci (zobacz rozdział 4.). W algorytmach stanu łącza każdy router musi rozsyłać komunikaty o stanie łącza do wszystkich pozostałych routerów w sieci. Taki komunikat obejmuje listę wszystkich bezpośrednich sąsiadów routera i kosztów bezpośredniej komunikacji z nimi. Kiedy router otrzyma komunikaty o stanie łącza od wszystkich pozostałych routerów, może utworzyć kompletną mapę sieci, uruchomić algorytm routingu określający ścieżkę o najmniejszym koszcie i skonfigurować tabelę przekazywania. Inga może przeprowadzić stosunkowo łatwy atak na algorytm routingu przez dystrybucję fałszywych

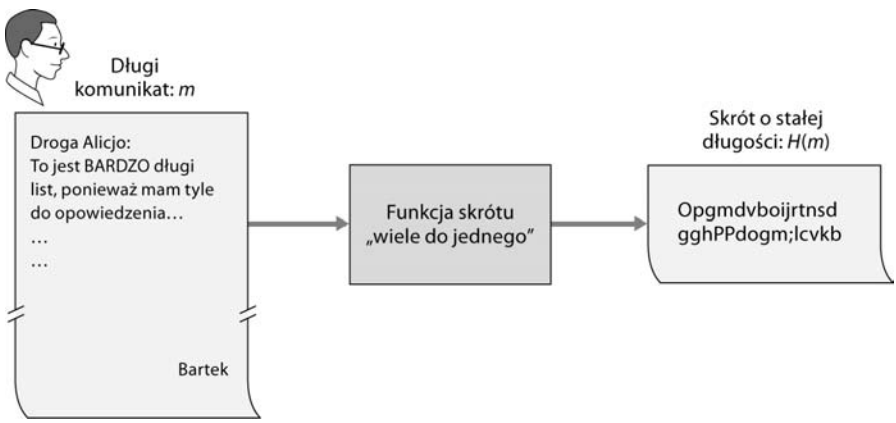
komunikatów z nieprawidłowymi informacjami o stanie łącza. Stąd wynika konieczność zachowania integralności komunikatów. Kiedy router B otrzyma komunikat o stanie łącza od routera A, będzie mógł zweryfikować, że to router A przygotował tę wiadomość i nikt nie zmodyfikował jej po drodze.

W tym podrozdziale opisujemy popularną technikę zapewniania integralności komunikatów stosowaną w wielu bezpiecznych protokołach sieciowych. Jednak zanim do tego przejdziemy, musimy omówić następne ważne zagadnienie w kryptografii — kryptograficzne funkcje skrótu.

8.3.1. Kryptograficzne funkcje skrótu

Jak przedstawia to rysunek 8.7, funkcja skrótu przyjmuje dane wejściowe m i oblicza łańcuch $H(m)$ o stałej długości (tak zwany skrót). Internetowa suma kontrolna (rozdział 3.) i kody CRC (rozdział 4.) są zgodne z tą definicją. **Kryptograficzna funkcja skrótu** musi dodatkowo mieć poniższe właściwości:

- Znalezienie dwóch dowolnych komunikatów x i y, takich że $H(x) = H(y)$, powinno być obliczeniowo niewykonalne.



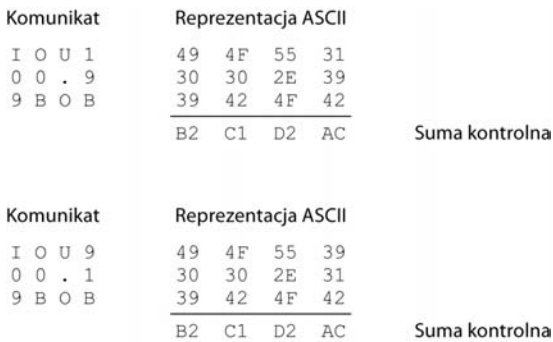
Rysunek 8.7. Funkcje skrótu

W mniej formalnym języku oznacza to, że intruz nie zdoła na podstawie obliczeń zastąpić jednego komunikatu (chronionego funkcją skrótu) innym. Powoduje to, że jeśli (m, $H(m)$) to komunikat i jego skrót utworzone przez nadawcę, intruz nie może sfałszować treści innej wiadomości y, tak aby miała ona taką samą wartość skrótu jak pierwotny komunikat.

Sprawdźmy, dlaczego prosta suma kontrolna źle spełniałaby funkcję kryptograficznego skrótu wiadomości. Zamiast posługiwać się arytmetyką z uzupełnieniem do jedności (jak w internetowej sumie kontrolnej), potraktujmy każdy znak jako bajt i dodawajmy je do siebie w czterobajtowych grupach. Przypuśćmy, że Bartek jest winny Alicji 100,99 dolara i wysyła do niej poświadczenie wie-

rzytelności w postaci łańcucha tekstu „IOU100.99BOB". Kody ASCII (w notacji szesnastkowej) tych liter to 49, 4F, 55, 31, 30, 30, 2E, 39, 39, 42, 4F, 42.

Na górze rysunku 8.8 pokazano, że 4-bajtowa suma kontrolna tej wiadomości wynosi B2 C1 D2 AC. Nieco inna wiadomość (znacznie bardziej kosztowna dla Bartka) znajduje się na dole rysunku 8.8. Komunikaty „IOU100.99BOB" oraz „IOU900.19BOB" mają *tę samą* sumę kontrolną. Zatem ten prosty algorytm sumy kontrolnej nie spełnia dwóch spośród wymienionych wyżej wymagań. Dysponując oryginalnymi danymi, możemy łatwo znaleźć inny zbiór danych z identyczną sumą kontrolną. Do celów bezpieczeństwa potrzebujemy więc bardziej zaawansowanej funkcji skrótu.



Rysunek 8.8. Wiadomość pierwotna i sfałszowana mają tę samą sumę kontrolną!

Obecnie powszechnie używany jest algorytm skrótu wiadomości MD5 opracowany przez Rona Rivesta [RFC 1321]. Oblicza on 128-bitowy skrót wiadomości w czteroetapowym procesie składającym się z uzupełniania (dodawanie jedynki i takiej liczby zer, aby długość wiadomości spełniała pewne kryteria), dołączania (dodawania 64-bitowej reprezentacji długości wiadomości przed uzupełnieniem), inicjalizacji akumulatora i końcowej pętli, w której bloki wiadomości zawierające po 16 słów są przetwarzane w czterech rundach. Opis MD5 (wraz z implementacją w języku C) można znaleźć w [RFC 1321].

Drugim najczęściej używanym algorytmem skrótu jest Secure Hash Algorithm (SHA-1) [FIPS 1995]. Algorytm ten opiera się na zasadach podobnych do tych, które zastosowano w MD4 [RFC 1320], poprzedniku MD5. SHA-1 jest standardowym algorytmem skrótu wiadomości przyjętym przez rząd Stanów Zjednoczonych. Tworzy on 160-bitowy skrót. Większa długość skrótu sprawia, że jest bezpieczniejszy od MD5.

8.3.2. Kod MAC

Wróćmy do problemu zachowywania integralności komunikatów. Teraz, kiedy Czytelnicy wiedzą już, jak działają funkcje skrótu, przeprowadźmy pierwszą próbę zapewnienia integralności wiadomości:

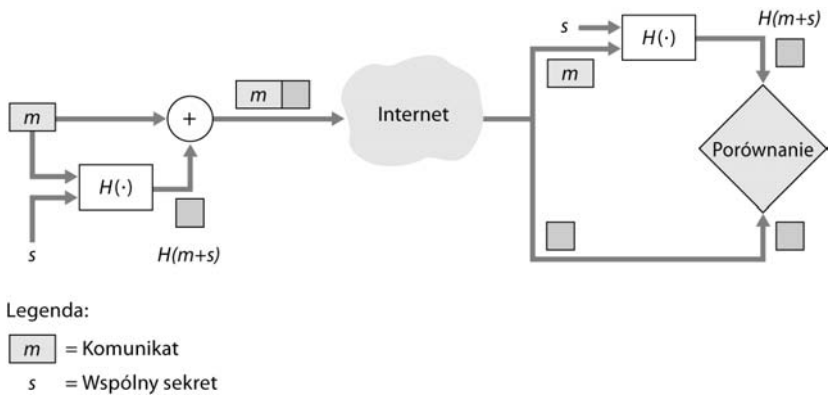
1. Alicja tworzy komunikat *m* i oblicza skrót *H*(*m*) (na przykład za pomocą algorytmu SHA-1).
2. Alicja dołącza skrót *H*(*m*) do wiadomości *m*, tworząc w ten sposób komunikat rozszerzony (*m*, *H*(*m*)), który wysyła do Bartka.
3. Bartek otrzymuje komunikat rozszerzony (*m*, *h*) i oblicza *H*(*m*). Jeśli *H*(*m*) = *h*, Bartek uznaje, że wszystko jest w porządku.

To podejście ma oczywistą wadę. Inga może utworzyć fałszywy komunikat *m`*, w którym podaje się za Alicję, obliczyć skrót *H*(*m`*) i wysłać do Bartka dane (*m`*, *H*(*m`*)). Kiedy Bartek otrzyma wiadomość, w kroku 3. komunikat przejdzie weryfikację, dlatego Bartek nie będzie niczego podejrzewał.

Aby zapewnić integralność komunikatu, Alicja i Bartek obok kryptograficznej funkcji skrótu potrzebują wspólnego sekretu *s*. Ten sekret (jest to po prostu łańcuch bitów) to tak zwany **klucz uwierzytelniający**. Za pomocą tego klucza można zapewnić integralność wiadomości w następujący sposób:

1. Alicja tworzy komunikat *m*, scala *s* z *m*, aby utworzyć sekwencję *m*+*s*, i oblicza skrót *H*(*m*+*s*) (na przykład za pomocą algorytmu SHA-1). Wartość *H*(*m*+*s*) jest nazywana **kodem MAC** (ang. *Message Authentication Code*).
2. Alicja dołącza następnie kod MAC do komunikatu *m*, tworząc w ten sposób komunikat rozszerzony (*m*, *H*(*m*+*s*)), który wysyła do Bartka.
3. Bartek otrzymuje komunikat rozszerzony (*m*, *h*), a ponieważ zna *s*, może obliczyć kod MAC *H*(*m*+*s*). Jeśli *H*(*m*+*s*) = *h*, Bartek uznaje, że wszystko jest w porządku.

Procedurę tę w uproszczeniu przedstawia rysunek 8.9. Czytelnicy powinni zauważyć, że opisywany tu kod MAC (ang. *Message Authentication Code*) to nie adres MAC (ang. *Medium Access Control*) wykorzystywany w protokołach warstwy łącza danych!



Rysunek 8.9. Kod MAC

Korzystną cechą kodu MAC jest to, że nie wymaga on stosowania algorytmu szyfrującego. W wielu zastosowaniach, w tym w opisanych wcześniej algorytmach stanu łącza, dla komunikujących się ze sobą jednostek ważna jest tylko integralność komunikatów, a nie ich poufność. Za pomocą kodu MAC strony mogą uwierzytelniać przesyłane wiadomości bez konieczności stosowania skomplikowanych algorytmów szyfrujących w procesie zapewniania integralności.

Jak można się domyślić, przez lata zaproponowano wiele różnych standardów tworzenia kodów MAC. Najpopularniejszy z nich to **HMAC**. Można go stosować razem z algorytmami MD5 i SHA-1. W standardzie HMAC dane i klucz uwierzytelniający są dwukrotnie przekazywane do funkcji skrótu [Kaufman 1995; RFC 2104].

Nadal jednak trzeba rozwiązać pewien ważny problem. Jak można przesyłać wspólny klucz uwierzytelniający między komunikującymi się jednostkami? Na przykład w algorytmie stanu łącza trzeba w jakiś sposób przekazać wspólny klucz uwierzytelniający do każdego routera w systemie autonomicznym (zauważmy, że wszystkie routery mogą korzystać z tego samego klucza). Administrator sieci może to zrobić przez fizyczne podejście do każdego routera. Jeśli jest leniwy, a wszystkie routery mają własne klucze publiczne, administrator może przesłać klucz uwierzytelniający do dowolnego z routerów przez zaszyfrowanie danych za pomocą klucza publicznego routera i przesłanie zaszyfrowanego klucza przez sieć.

8.3.3. Podpisy cyfrowe

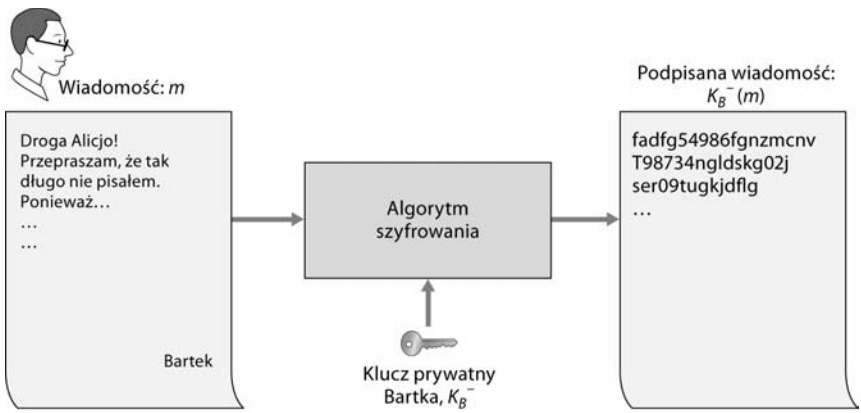
Pomyślmy, jak często składamy podpis na papierze. Podpisujemy czeki, pokwitowania, dokumenty i listy. Nasz podpis potwierdza, że to właśnie my (a nie kto inny) przeczytaliśmy dokument i przyjęliśmy go do wiadomości. W świecie cyfrowym często zachodzi potrzeba zidentyfikowania właściciela lub twórcy dokumentu albo wskazania, że czytająca go osoba zgadza się z jego treścią. Cele te można osiągnąć dzięki technice kryptograficznej zwanej **podpisem cyfrowym**.

Podobnie jak podpis ręczny podpis cyfrowy powinien być weryfikowalny i niepodrabialny. Musi istnieć możliwość dowiedzenia, że dokument rzeczywiście został podpisany przez konkretną osobę (podpis musi być weryfikowalny) i że *tylko* ta osoba mogła podpisać dokument (podpisu nie można podrobić).

Zastanówmy się teraz nad tym, jak zaprojektować system podpisów cyfrowych. Zauważmy, że kiedy Bartek podpisuje wiadomość, musi umieścić w niej coś niepowtarzalnego. Bartek może rozważyć zastosowanie podpisu w formie kodu MAC. Aby go utworzyć, należy połączyć niepowtarzalny klucz z komunikatem, a następnie wygenerować na ich podstawie skrót. Jednak Alicja do zweryfikowania takiego podpisu będzie potrzebować kopii klucza, co oznacza, że nie będzie on niepowtarzalny. Dlatego kod MAC nie rozwiązuje problemu.

Przypomnijmy, że w kryptografii z kluczem publicznym Bartek ma dostęp zarówno do klucza publicznego, jak i prywatnego, przy czym oba są niepowtarzalne. Dlatego metoda ta jest doskonałym kandydatem na system podpisów cyfrowych. Zobaczmy, jak on działa.

Przypuśćmy, że Bartek chce cyfrowo podpisać dokument *m*. Dokumentem tym może być plik albo wiadomość, którą Bartek zamierza przesłać. Jak pokazano na rysunku 8.10, w celu podpisania dokumentu Bartek po prostu używa swojego klucza prywatnego K_B^- w celu obliczenia $K_B^-(m)$. Początkowo może wydawać się dziwne, że Bartek podpisuje dokument swoim kluczem prywatnym (którego, jak Czytelnicy widzieli w podrozdziale 8.2, zwykle używa do deszyfrowania wiadomości zaszyfrowanych jego kluczem publicznym). Przypominamy jednak, że szyfrowanie i deszyfrowanie to zwykłe operacje matematyczne (w przypadku RSA podnoszenie do potęgi *e* lub *d*) i że celem Bartka nie jest ukrycie treści dokumentu, ale podpisanie go w sposób weryfikowalny, niepodrabialny i niezaprzeczalny. Bartek ma dokument, *m*, a jego cyfrowym podpisem dokumentu jest $K_B^-(m)$.



Rysunek 8.10. Tworzenie cyfrowego podpisu dokumentu

Czy cyfrowy podpis $K_B^-(m)$ spełnia wymagania weryfikowalności, niepodrabialności i niezaprzeczalności? Przypuśćmy, że Alicja ma *m* oraz $K_B^-(m)$ i chce dowieść w sądzie, że Bartek rzeczywiście podpisał dokument i był jedyną osobą, która mogła to zrobić. Alicja bierze publiczny klucz Bartka K_B^+ i stosuje go do podpisu cyfrowego $K_B^-(m)$ związanego z dokumentem *m*, tzn. oblicza $K_B^+(K_B^-(m))$ i po chwili dramatycznego napięcia prezentuje dokument *m* dokładnie odpowiadający oryginałowi! Następnie Alicja dowodzi, że dokumentu nie mógł podpisać nikt oprócz Bartka, ponieważ:

- Ktokolwiek podpisał dokument, musiał użyć prywatnego klucza K_B^- do obliczenia takiego podpisu $K_B^-(m)$, że $K_B^+(K_B^-(m)) = m$.

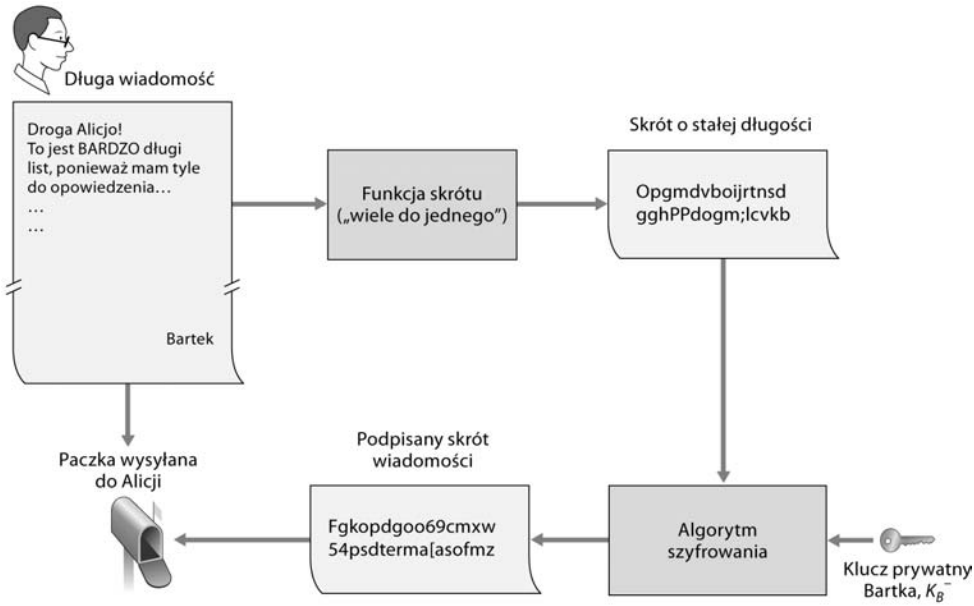
- Jedyną osobą, która zna klucz prywatny K_B^-, jest Bartek. Z opisu algorytmu RSA w podrozdziale 8.2 wiemy, że znajomość klucza publicznego K_B^+ nie pozwala odkryć klucza prywatnego K_B^-. Zatem K_B^- może znać tylko ta osoba, która wygenerowała parę kluczy (K_B^+, K_B^-) — czyli Bartek (zakładamy, że Bartek nie podał nikomu swojego klucza K_B^- i że nikt mu go nie ukradł).

Trzeba również zaznaczyć, że jeśli oryginalny dokument *m* zostanie zmodyfikowany do jakiejś innej postaci *m′*, podpis utworzony przez Bartka dla *m* nie będzie ważny dla *m′*, ponieważ $K_B^+(K_B^-(m))$ nie równa się *m′*. Widzimy więc, że szyfrowanie z kluczem publicznym zapewnia także integralność wiadomości, ponieważ umożliwia odbiorcy zweryfikowanie tożsamości nadawcy i tego, że komunikat nie został zmodyfikowany.

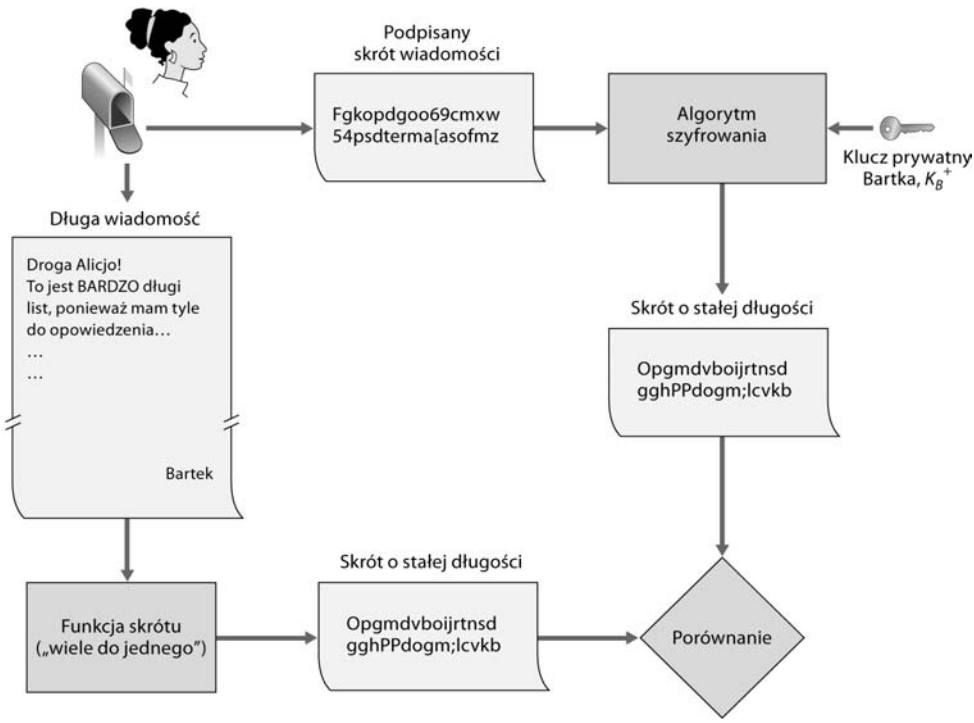
Wadą podpisywania danych przez szyfrowanie jest to, że proces szyfrowania i deszyfrowania jest złożony obliczeniowo. Z uwagi na koszty tych operacji podpisywanie danych przez pełne szyfrowanie i odszyfrowywanie byłoby przesadą. Wydajniejszym podejściem jest zastosowanie w podpisach cyfrowych funkcji skrótu. W punkcie 8.3.2 napisaliśmy, że algorytm skrótu przyjmuje wiadomość *m* dowolnej długości i oblicza jej „odcisk palca" o stałej długości wyznaczany przez wzór *H*(*m*). Za pomocą funkcji skrótu Bartek podpisuje skrót wiadomości zamiast jej samej, czyli oblicza $K_B^-(H(m))$. Ponieważ wartość *H*(*m*) jest zwykle znacznie mniejsza niż pierwotny komunikat *m*, pozwala to znacznie zmniejszyć złożoność obliczeniową procesu tworzenia podpisu cyfrowego.

Na rysunku 8.11 skrótowo przedstawiono procedurę tworzenia podpisu cyfrowego (na przykładzie wiadomości wysyłanej przez Bartka do Alicji). Bartek przetwarza długą wiadomość za pomocą funkcji skrótu, aby utworzyć skrót wiadomości. Następnie cyfrowo podpisuje skrót wiadomości swoim kluczem prywatnym. Pierwotna wiadomość (pisana tekstem jawnym) wraz z cyfrowo podpisanym skrótem (od tej pory nazywanym podpisem cyfrowym) zostaje wysłana do Alicji. Na rysunku 8.12 przedstawiono skrótowo procedurę sprawdzania podpisu. Alicja odszyfrowuje podpis cyfrowy za pomocą publicznego klucza nadawcy, aby uzyskać skrót wiadomości. Alicja stosuje też funkcję skrótu na odebranej wiadomości, aby uzyskać drugi skrót. Jeśli oba skróty są identyczne, Alicja wie, że wiadomość nie została zmodyfikowana, a jej autorem jest Bartek.

Zanim przejdziemy dalej, porównajmy pokrótce podpisy cyfrowe z kodami MAC. Występują między nimi podobieństwa, ale też ważne różnice. W obu technikach punktem wyjścia jest wiadomość (lub dokument). Aby utworzyć kod MAC na podstawie komunikatu, należy dołączyć do niego klucz uwierzytelniający, a następnie utworzyć skrót dla połączonych danych. Zauważmy, że tworzenie kodu MAC nie wymaga ani szyfrowania z kluczem publicznym, ani szyfrowania z kluczem symetrycznym. Aby przygotować podpis cyfrowy, najpierw trzeba uzyskać skrót wiadomości, a następnie zaszyfrować ją za pomocą



Rysunek 8.11. Wysyłanie wiadomości podpisanej cyfrowo



Rysunek 8.12. Weryfikowanie integralności podpisanej wiadomości

klucza prywatnego (metodą szyfrowania z kluczem publicznym). Dlatego tworzenie podpisu cyfrowego to bardziej złożona technika. Wymaga ona infrastruktury PKI (ang. *Public Key Infrastructure*) i urzędu certyfikacyjnego, co opisujemy w dalszej części punktu. W podrozdziale 8.4 wyjaśniamy, że w PGP (jest to popularny system obsługi bezpiecznej poczty elektronicznej) do zapewniania integralności wiadomości wykorzystano podpisy cyfrowe. Pokazaliśmy już, że w protokole OSPF służą do tego kody MAC. W podrozdziałach 8.5 i 8.6 opisujemy, że kody te są wykorzystywane także w popularnych bezpiecznych protokołach warstwy transportowej i sieci.

Certyfikacja kluczy publicznych

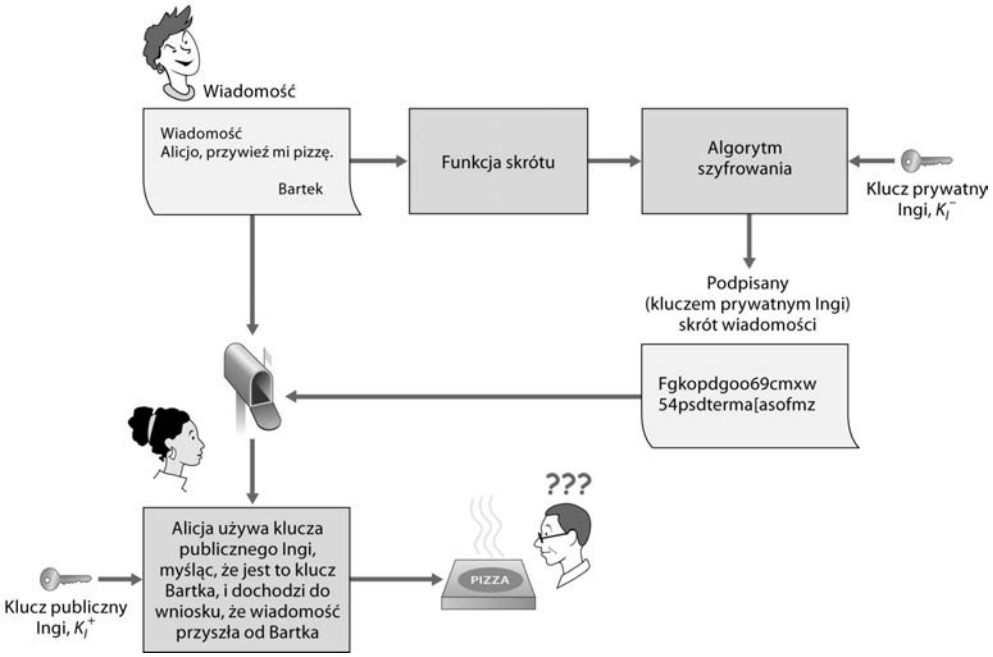
Ważnym zastosowaniem podpisów cyfrowych jest **certyfikacja kluczy publicznych**, czyli potwierdzanie, że dany klucz publiczny należy do określonej jednostki. Certyfikacja kluczy publicznych jest stosowana w wielu popularnych bezpiecznych protokołach sieciowych, na przykład w IPsec i SSL.

Aby zrozumieć problem związany z kluczami, rozważmy internetową wersję tradycyjnego żartu z zamawianiem pizzy. Przypuśćmy, że Alicja zajmuje się dowozem pizzy i zbiera zamówienia przez internet. Bartek, wielki miłośnik pizzy, wysyła do Alicji wiadomość pisaną tekstem jawnym, w której podaje swój adres i określa typ pizzy. Bartek dołącza też podpis cyfrowy (to znaczy podpisany skrót wiadomości pisanej tekstem jawnym), aby udowodnić, że to on wysłał komunikat. Alicja może uzyskać klucz publiczny Bartka (z jego osobistej strony WWW, z serwera kluczy albo z wiadomości e-mail) i zweryfikować podpis. W ten sposób upewnia się, że zamówienie złożył Bartek, a nie jakiś nastoletni żartowniś.

Wszystko działa świetnie, dopóki nie pojawi się Inga. Jak pokazano na rysunku 8.13, Inga postanawia spłatać figla. Inga wysyła do Alicji wiadomość, w której podaje się za Bartka, podaje jego adres domowy i zamawia pizzę. Ponadto Inga wysyła Alicji swój klucz publiczny, twierdząc, że należy on do Bartka. Inga dołącza też podpis cyfrowy, ale skrót wiadomości podpisuje swoim kluczem prywatnym. W tym przykładzie Alicja zastosuje klucz publiczny Ingi (sądząc, że jest to klucz Bartka) do podpisu cyfrowego i dojdzie do wniosku, że wiadomość jawna została rzeczywiście utworzona przez Bartka. Bartek będzie bardzo zdziwiony, gdy do jego drzwi zastuka doręczyciel z pizzą z pepperoni i anchovies!

Z powyższego przykładu wynika, że jeśli kryptografia z kluczem publicznym ma spełnić swoje zadanie, strona komunikacji musi mieć pewność, że dysponuje prawdziwym kluczem publicznym drugiej strony (użytkownika, routera, przeglądarki itd.). Przykładowo, kiedy Alicja komunikuje się z Bartkiem przy użyciu kryptografii z kluczem publicznym, musi wiedzieć na pewno, że klucz publiczny, który ma należeć do Bartka, rzeczywiście do niego należy.

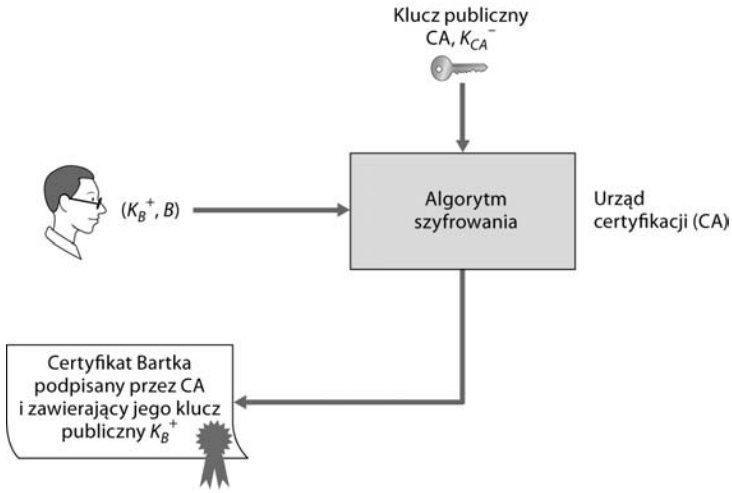
Kojarzeniem klucza publicznego z określoną jednostką zwykle zajmuje się **urząd certyfikacyjny** (ang. *Certification Authority* — CA), którego zadaniem jest weryfikowanie tożsamości i wydawanie certyfikatów. CA pełni następujące funkcje:



Rysunek 8.13. Inga podszywa się pod Bartka, używając kryptografii z kluczem publicznym

- CA potwierdza, że jednostka (osoba, router itp.) jest tym, za kogo się podaje. Nie ma żadnych sztywno określonych procedur certyfikacji. Musimy liczyć na to, że urząd CA rygorystycznie zweryfikował tożsamość jednostki. Gdyby na przykład Inga mogła wejść do urzędu CA „Przeminęło z wiatrem", oznajmić „Jestem Alicja" i otrzymać certyfikat poświadczający tożsamość Alicji, to nie powinniśmy darzyć zaufaniem kluczy publicznych wystawianych przez ten urząd. Z drugiej strony, moglibyśmy pokładać większą (albo mniejszą!) ufność w urzędzie CA finansowanym ze środków publicznych. Jednostce skojarzonej z kluczem publicznym możemy ufać tylko w takim stopniu, w jakim ufamy urzędowi CA i jego technikom weryfikacji tożsamości. Cóż za misterna sieć zaufania!
- Kiedy urząd CA potwierdzi tożsamość jednostki, tworzy **certyfikat**, który wiąże klucz publiczny z tą tożsamością. Certyfikat zawiera klucz publiczny oraz globalnie unikatowe informacje o właścicielu klucza (na przykład nazwisko człowieka albo adres IP urządzenia). Certyfikat jest podpisany cyfrowo przez urząd CA. Procedurę tę przedstawiono na rysunku 8.14.

Zobaczmy teraz, jak można użyć certyfikatów do zwalczania złośliwych zamawiaczy pizzy, takich jak Inga. Kiedy Bartek składa zamówienie, wysyła także podpisany przez urząd CA certyfikat. Alicja używa publicznego klucza CA, aby sprawdzić ważność certyfikatu Bartka, i wyodrębnia klucz publiczny Bartka.



Rysunek 8.14. Bartek uzyskuje certyfikat od urzędu CA

Organizacje International Telecommunications Union (ITU) oraz IETF opracowały standardy dotyczące urzędów certyfikacji. Standard ITU X.509 [ITU 1993] definiuje usługę uwierzytelniania oraz składnię używaną w certyfikatach. [RFC 1422] opisuje zarządzanie kluczami na użytek bezpiecznej internetowej poczty elektronicznej. Standard jest zgodny z X.509, ale wykracza poza jego ramy, określając procedury i konwencje obowiązujące w architekturze zarządzania kluczami. W tabeli 8.4 opisano najważniejsze pola certyfikatu.

Nazwa pola	Opis
Version	Numer wersji standardu X.509.
Serial Number	Unikatowy identyfikator certyfikatu wydanego przez CA.
Signature	Określa algorytm, za pomocą którego CA podpisał certyfikat.
Issuer Name	Nazwa CA wydającego certyfikat w formacie Distinguished Name (DN) [RFC 2253].
Validity period	Początkowa i końcowa data ważności certyfikatu.
Subject name	Nazwa (w formacie DN) jednostki, której klucz publiczny jest związany z certyfikatem.
Subject public key	Klucz publiczny jednostki oraz informacje o algorytmie klucza publicznego (i parametrach algorytmu), którego należy używać w połączeniu z tym kluczem.

Tabela 8.4. Wybrane pola w certyfikacie X.509 i kluczu publicznym RFC 1422

8.3.4. Uwierzytelnianie punktów końcowych

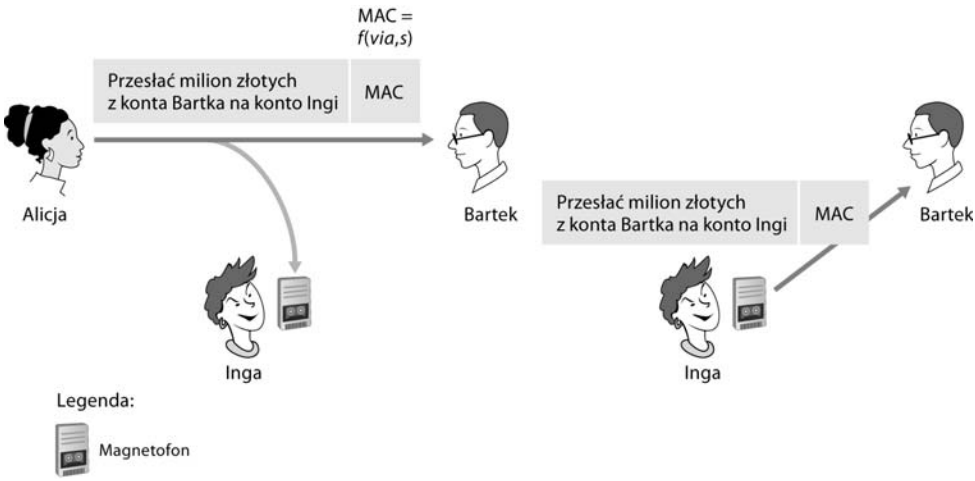
Uwierzytelnianie punktów końcowych to potwierdzanie swojej tożsamości. Ludzie uwierzytelniają się na wiele różnych sposobów: rozpoznajemy twarze innych osób, kiedy się z nimi spotykamy, rozpoznajemy głosy podczas rozmowy telefonicznej, a urzędnicy celni porównują nasze twarze z fotografiami w paszporcie. Podczas uwierzytelniania przez sieć strony nie mogą wykorzystać informacji biometrycznych, takich jak wygląd zewnętrzny albo brzmienie głosu. Przekonamy się zresztą, że uwierzytelniającymi się stronami często są elementy sieciowe, takie jak routery albo procesy w aplikacjach typu klient-serwer. W takim przypadku uwierzytelnianie musi odbywać się wyłącznie na podstawie wymienianych komunikatów i danych.

Najpierw zastanowimy się, czy do uwierzytelniania punktów końcowych można wykorzystać kody MAC (omówiliśmy je w punkcie 8.3.2). Załóżmy, że Alicja i Bartek znają wspólny sekret *s*, Alicja chce wysłać wiadomość (na przykład segment TCP) do Bartka, a Bartek chce mieć pewność, że nadawcą odebranej wiadomości jest Alicja. Naturalne rozwiązanie oparte na kodach MAC działa tak: Alicja tworzy na podstawie wiadomości kod MAC, dołącza do komunikatu ten kod i wysyła uzyskaną rozszerzoną wiadomość do Bartka. Kiedy ten odbierze rozszerzoną wiadomość, użyje dołączonego kodu MAC do zweryfikowania zarówno źródła, jak i integralności komunikatu. Ponieważ tylko Alicja i Bartek znają wspólny sekret, to jeśli obliczenia kodu MAC wykonane przez Bartka dadzą wynik zgodny z kodem MAC z rozszerzonej wiadomości, Bartek będzie miał pewność, że to Alicja wysłała komunikat (i że nikt nie zmodyfikował go w czasie przesyłania).

Ataki przez odtwarzanie i identyfikatory jednorazowe

Czy na pewno tak jest? W rzeczywistości Bartek nie ma całkowitej pewności co do źródła wiadomości, ponieważ mógł zostać zwiedziony **atakiem przez odtwarzanie** (ang. *playback attack*). Jak przedstawia to rysunek 8.15, Inga musi jedynie podsłuchać i zarejestrować rozszerzoną wiadomość od Alicji, a następnie odtworzyć ten komunikat. Powtórzona wiadomość może brzmieć na przykład tak: „Można przesłać milion złotych z konta Bartka na konto Ingi", co spowoduje transfer dwóch milionów złotych. Inny przykładowy komunikat to: „Łącze między routerem Alicja i routerem Cezary przestało działać". Jeśli ta wiadomość zostanie wysłana po naprawieniu wspomnianego łącza, może doprowadzić do błędnego skonfigurowania tabel przekazywania.

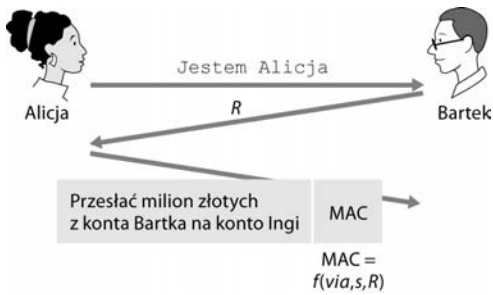
Niepowodzenie pokazane na rysunku 8.15 wynika z tego, że Bartek nie może odróżnić pierwotnego uwierzytelnienia Alicji od jego odtworzonej kopii. Oznacza to, że Bartek nie może stwierdzić, czy Alicja porozumiewa się z nim „na żywo" (tzn. rzeczywiście znajduje się na drugim końcu połączenia), czy też jej komunikaty stanowią zarejestrowaną i odtworzoną kopię wcześniejszego uwierzytelnienia. Bardzo (*bardzo*) uważni Czytelnicy zapewne pamiętają, że podobny



Rysunek 8.15. Atak przez odtwarzanie

problem trzeba było rozwiązać w protokole 3-etapowego negocjowania TCP — nie chcieliśmy, aby serwer akceptował połączenie, jeśli otrzymany segment SYN był starą kopią (retransmisją) segmentu SYN wcześniejszego połączenia. Jak serwer ustalał, czy klient rzeczywiście próbuje się z nim połączyć? Wybierał wstępny numer sekwencyjny, który nie był używany od dłuższego czasu, wysyłał ten numer klientowi, a następnie czekał, aż klient odeśle segment ACK zawierający ten numer. Możemy wykorzystać tę samą ideę do celów uwierzytelniania.

Identyfikator jednorazowy (ang. *nonce*) to wartość, którą protokół wykorzystuje tylko raz. Innymi słowy, jeśli protokół użyje identyfikatora jednorazowego, to nigdy nie posłuży się nim ponownie. Jak ilustruje to rysunek 8.16, identyfikatory jednorazowe są używane w następujący sposób:



Rysunek 8.16. Wykorzystanie identyfikatorów jednorazowych do zabezpieczenia się przed atakami przez odtwarzanie

1. Bartek wybiera identyfikator jednorazowy *R* i wysyła go Alicji. Zauważmy, że identyfikatory te są przesyłane jako tekst jawny. Alicja tworzy kod MAC na podstawie pierwotnej wiadomości, wspólnego sekretu *s* i identyfikatora

jednorazowego *R*. W celu wygenerowania kodu MAC Alicja może na przykład połączyć wspólny sekret i identyfikator jednorazowy z komunikatem, a następnie przekazać efekt tej operacji do funkcji skrótu. Następnie Alicja dodaje kod MAC do komunikatu, tworzy wiadomość rozszerzoną i przesyła ją do Bartka.

2. Bartek oblicza kod MAC na podstawie komunikatu (zawartego w wiadomości rozszerzonej), identyfikatora jednorazowego *R* i wspólnego sekretu *s*. Jeśli uzyskany kod MAC jest równy temu kodowi z wiadomości rozszerzonej, Bartek wie nie tylko, iż to Alicja wygenerowała komunikat, ale też że zrobiła to *po* przesłaniu przez Bartka identyfikatora jednorazowego (wiadomo to, ponieważ identyfikator jest potrzebny do obliczenia prawidłowego kodu MAC).

W podrozdziałach 8.5 i 8.6 przy omawianiu protokołów SSL i IPsec pokażemy, że połączenie identyfikatorów jednorazowych i kodów MAC często wykorzystuje się w bezpiecznych protokołach sieciowych do zapewniania integralności wiadomości i uwierzytelniania punktów końcowych.

Co się jednak dzieje, kiedy Alicja chce przesłać wiele wiadomości, na przykład serię segmentów TCP? Czy Bartek będzie musiał dla każdego z nich przesłać Alicji nowy identyfikator jednorazowy? W praktyce potrzebny jest tylko jeden taki identyfikator. Przy opisywaniu protokołu SSL w podrozdziale 8.5 pokażemy, że pojedynczy identyfikator jednorazowy w połączeniu z numerami sekwencyjnymi umożliwia Bartkowi zweryfikowanie „świeżości" wszystkich wiadomości otrzymanych od Alicji.

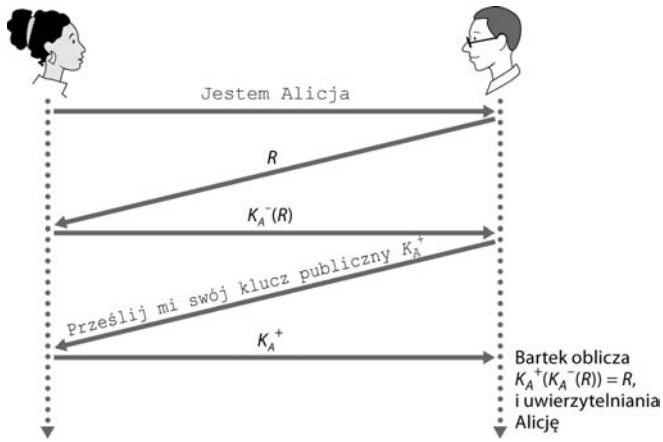
Uwierzytelnianie za pomocą kryptografii z kluczem publicznym

Zastosowanie identyfikatora jednorazowego i kryptografii z kluczem symetrycznym pozwoliło nam zaprojektować udany protokół uwierzytelniania. Powstaje naturalne pytanie, czy możemy rozwiązać problem uwierzytelniania za pomocą identyfikatora jednorazowego i kryptografii z kluczem publicznym. Użycie klucza publicznego pozwoliłoby uniknąć trudności, które pojawiają się w każdym systemie ze współużytkowanym kluczem — nie musielibyśmy się martwić, jak obie strony komunikacji mają poznać wartość klucza. Oto naturalny protokół wykorzystujący kryptografię z kluczem publicznym:

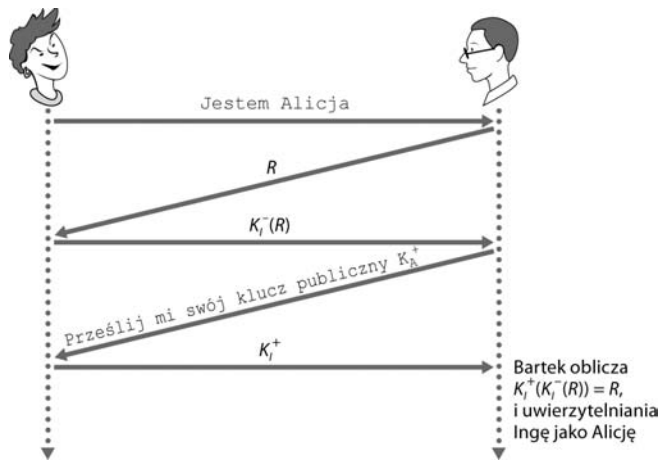
1. Alicja wysyła Bartkowi komunikat „`Jestem Alicja`".
2. Bartek wybiera identyfikator jednorazowy *R* i wysyła go Alicji. Również w tym przypadku identyfikator będzie gwarantował, że komunikacja odbywa się „na żywo".
3. Alicja szyfruje identyfikator jednorazowy swoim kluczem *prywatnym* K_A^- i wysyła Bartkowi obliczoną wartość $K_A^-(R)$. Ponieważ klucz K_A^- zna tylko Alicja, nikt inny nie może wygenerować tej wartości.

4. Bartek odszyfrowuje odebrany komunikat za pomocą klucza publicznego Alicji K_A^+, to znaczy oblicza $K_A^+(K_A^-(R))$. Z podrozdziału 8.2 poświęconego kryptografii RSA wiemy, że $K_A^+(K_A^-(R)) = R$. W ten sposób Bartek oblicza R i uwierzytelnia Alicję.

Działanie tego protokołu zilustrowano na rysunku 8.17. Czy protokół ten jest bezpieczny? Ponieważ wykorzystuje on kryptografię z kluczem publicznym, Bartek musi uzyskać klucz publiczny Alicji. Prowadzi to do interesującego scenariusza, pokazanego na rysunku 8.18, w którym Inga może podszyć się pod Alicję.



Rysunek 8.17. Prawidłowe działanie protokołu uwierzytelniania za pomocą klucza publicznego



Rysunek 8.18. Luka w zabezpieczeniach protokołu uwierzytelniania za pomocą klucza publicznego

1. Inga wysyła Bartkowi komunikat „`Jestem Alicja`".
2. Bartek wybiera identyfikator jednorazowy R i wysyła go Alicji, ale Inga przechwytuje komunikat.
3. Inga szyfruje identyfikator jednorazowy swoim kluczem prywatnym K_I^- i wysyła Bartkowi obliczoną wartość $K_I^-(R)$. Dla Bartka wartość $K_I^-(R)$ jest po prostu garścią bitów; nie wie on, czy bity reprezentują $K_I^-(R)$, czy $K_A^-(R)$.
4. Bartek musi teraz uzyskać klucz publiczny Alicji K_A^+, aby zastosować go do otrzymanej właśnie wartości. Wysyła do Alicji komunikat z prośbą o przesłanie klucza K_A^+ (albo pobiera klucz z witryny WWW Alicji). Inga przechwytuje także ten komunikat i przesyła Bartkowi K_I^+, tzn. swój klucz publiczny. Bartek oblicza $K_I^+(K_I^-(R)) = R$ i uwierzytelnia Ingę jako Alicję!

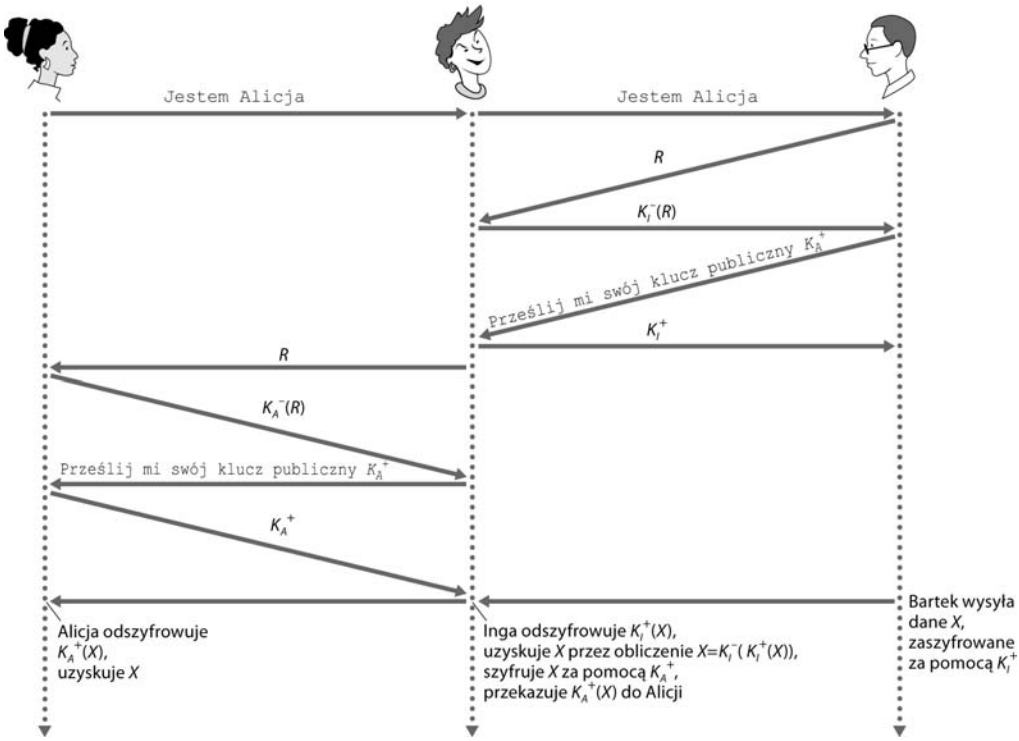
Z powyższego scenariusza wynika, że protokół uwierzytelniania za pomocą klucza publicznego jest bezpieczny w takim stopniu, w jakim bezpieczna jest dystrybucja kluczy publicznych. Na szczęście do bezpiecznej dystrybucji takich kluczy można wykorzystać certyfikaty, o czym Czytelnicy dowiedzieli się z podrozdziału 8.3.

W scenariuszu z rysunku 8.18 Bartek i Alicja mogliby się w końcu domyślić, że coś jest nie tak, ponieważ Bartek twierdziłby, że porozumiewał się z Alicją, a Alicja wiedziałaby, że nie porozumiewała się z Bartkiem. Istnieje jeszcze bardziej podstępny atak, który może pozostać niewykryty. W scenariuszu z rysunku 8.19 Alicja i Bartek rozmawiają ze sobą, ale — dzięki tej samej luce w protokole uwierzytelniania — Inga może *niezauważanie* wkraść się między nich. Jeśli Bartek zacznie wysyłać do Alicji dane zaszyfrowane kluczem otrzymanym od Ingi, Inga będzie mogła odtworzyć tekst jawny, po czym przekazać dane Alicji (uprzednio szyfrując je prawdziwym kluczem publicznym Alicji).

Bartek spokojnie wysyła zaszyfrowane dane, a Alicja spokojnie odbiera dane zaszyfrowane jej własnym kluczem publicznym; żadne z nich nie zdaje sobie sprawy z obecności Ingi. Jeśli Bartek i Alicja później się spotkają i omówią swoją interakcję, okaże się, że Alicja odebrała dokładnie to, co przesłał jej Bartek, więc nie nabiorą oni żadnych podejrzeń. Jest to przykład ataku typu „**człowiek w środku**" (tutaj bardziej odpowiednia byłaby nazwa „kobieta w środku").

8.4. Zabezpieczanie poczty elektronicznej

W poprzednich podrozdziałach omówiliśmy fundamentalne kwestie bezpieczeństwa sieciowego, w tym kryptografię z kluczem symetrycznym i publicznym, uwierzytelnianie, dystrybucję kluczy, ochronę integralności i podpisy cyfrowe. Teraz zbadamy, jak używa się tych narzędzi do zapewnienia bezpieczeństwa w internecie.



Rysunek 8.19. Atak typu „człowiek w środku"

Co ciekawe, zabezpieczenia można stosować w dowolnej spośród czterech warstw stosu protokołów internetowych. Jeśli zabezpieczymy specyficzny protokół warstwy aplikacji, korzystająca z niego aplikacja zyska dostęp do usług bezpieczeństwa takich jak poufność, uwierzytelnianie lub integralność. Jeśli zabezpieczymy protokół warstwy transportowej, dostęp do usług bezpieczeństwa zyskają wszystkie aplikacje korzystające z tego protokołu. Jeśli zaoferujemy usługi bezpieczeństwa w warstwie sieci, chronione będą wszystkie segmenty warstwy transportowej (a zatem wszystkie dane warstwy aplikacji). Jeśli zabezpieczymy łącze, to chronione będą wszystkie przepływające przez nie ramki.

W podrozdziałach od 8.4 do 8.7 dowiemy się, jak funkcjonują mechanizmy bezpieczeństwa w warstwie aplikacji, transportowej, sieci i łącza danych. Zgodnie z ogólną strukturą niniejszej książki zaczniemy od góry stosu protokołów i omówimy bezpieczeństwo w warstwie aplikacji. Wykorzystamy konkretną aplikację — pocztę elektroniczną — jako studium zabezpieczeń warstwy aplikacji. Następnie przejdziemy w dół stosu protokołów. Zbadamy protokół SSL (który zapewnia bezpieczeństwo w warstwie transportowej), IPsec (który zapewnia bezpieczeństwo w warstwie sieci), a wreszcie protokół bezprzewodowych sieci lokalnych IEEE 802.11.

Niektórzy zapewne zastanawiają się, dlaczego mechanizmy bezpieczeństwa funkcjonują w wielu warstwach internetu. Czy nie wystarczyłoby zapewnić bezpieczeństwa w warstwie sieci i na tym poprzestać? Istnieją dwie odpowiedzi na to pytanie. Po pierwsze, choć zabezpieczenia warstwy sieci mogą utajnić komunikację przez szyfrowanie wszystkich danych w datagramach (to znaczy wszystkich segmentów warstwy transportu), nie zapewniają bezpieczeństwa na poziomie użytkownika. Przykładowo sklep internetowy nie może wykorzystać zabezpieczeń warstwy IP do uwierzytelnienia klienta, który chce zrobić zakupy. Zatem oprócz utajnienia komunikacji w niższych warstwach potrzebne są również mechanizmy bezpieczeństwa operujące w wyższych warstwach. Po drugie, łatwiej jest wprowadzać nowe usługi internetowe (w tym usługi bezpieczeństwa) w wyższych warstwach stosu protokołów. Zamiast czekać na powszechne wdrożenie mechanizmów bezpieczeństwa w warstwie sieci, co zapewne potrwa jeszcze wiele lat, wielu programistów bierze sprawy we własne ręce i wbudowuje zabezpieczenia w ulubione aplikacje. Klasycznym przykładem jest oprogramowanie Pretty Good Privacy (PGP), które oferuje bezpieczną pocztę elektroniczną (omówimy je dalej w tym podrozdziale). PGP wymaga tylko aplikacji serwera oraz klienta; była to jedna z pierwszych technologii bezpieczeństwa, które znalazły zastosowanie w internecie.

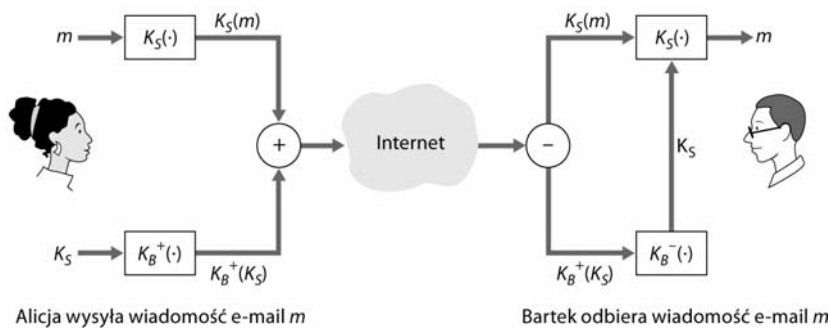
8.4.1. Bezpieczna poczta elektroniczna

W tym punkcie użyjemy technik opisanych w podrozdziałach 8.2 – 8.4 do zaprojektowania bezpiecznego systemu poczty elektronicznej. Ten wysokopoziomowy projekt będziemy opracowywać krok po kroku, stopniowo wprowadzając nowe usługi bezpieczeństwa. Przypomnijmy sobie przykład wspomniany w podrozdziale 8.1 — romans między Alicją a Bartkiem. Wyobraźmy sobie, że Alicja chce wysłać Bartkowi wiadomość e-mail, a Inga próbuje zakłócić ich komunikację.

Zanim przystąpimy do projektowania bezpiecznego systemu poczty elektronicznej dla Alicji i Bartka, zastanówmy się, jakie powinien on mieć cechy. Pierwszą i najważniejszą jest *poufność*. Jak stwierdziliśmy w podrozdziale 8.1, ani Alicja, ani Bartek nie chcą, aby Inga odczytała wiadomość Alicji. Drugą istotną cechą bezpiecznego systemu poczty elektronicznej jest *uwierzytelnianie nadawcy*. Mówiąc ściślej, kiedy Bartek otrzyma wiadomość: „`Już Cię nie kocham. Nie chcę Cię więcej widzieć. Niegdyś Twoja Alicja.`”, to będzie chciał upewnić się, że została ona wysłana przez Alicję, a nie przez Ingę. Kolejną funkcją, którą doceniliby nasi kochankowie, jest *ochrona integralności*, tzn. gwarancja, że wiadomości Alicji nie zostaną zmodyfikowane po drodze do Bartka. Wreszcie system poczty powinien zapewniać *uwierzytelnianie odbiorcy*; tzn. Alicja chciałaby mieć pewność, że wysyła wiadomość do Bartka, a nie do kogoś (na przykład Ingi), kto się pod niego podszywa.

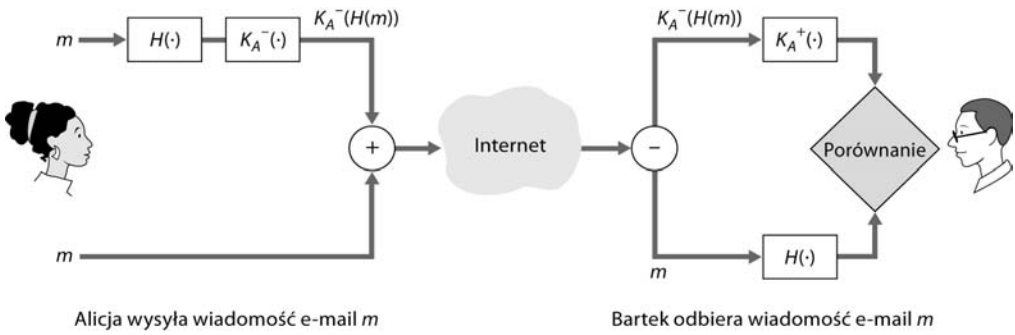
Zacznijmy więc od rozwiązania najważniejszego problemu — poufności. Najprostszym sposobem byłoby szyfrowanie wysyłanej wiadomości za pomocą klucza symetrycznego (DES lub AES) i odszyfrowywanie jej po odbiorze. Jak wyjaśniliśmy w podrozdziale 8.2, jeśli klucz symetryczny jest odpowiednio długi i dostępny tylko Alicji i Bartkowi, osobom postronnym (w tym Indze) będzie niezwykle trudno odczytać wiadomość. Choć ta metoda jest prosta, ma fundamentalną wadę, o której wspomniano w podrozdziale 8.2 — trudno jest przekazać klucz symetryczny w taki sposób, aby tylko Alicja i Bartek mieli jego kopie. Wybieramy więc inne rozwiązanie — kryptografię z kluczem publicznym (na przykład algorytm RSA). W tym przypadku Bartek udostępnia swój klucz publiczny (umieszczając go w serwerze kluczy albo na stronie WWW), Alicja szyfruje wiadomość tym kluczem, a następnie wysyła ją na adres e-mail Bartka. Kiedy Bartek otrzymuje wiadomość, po prostu odszyfrowuje ją za pomocą swojego klucza prywatnego. Jeśli Alicja jest pewna, że klucz publiczny rzeczywiście należy do Bartka, rozwiązanie to jest doskonałą metodą zapewniania żądanej poufności. Jedyny problem w tym, że szyfrowanie z kluczem publicznym jest mało wydajne, zwłaszcza w przypadku długich wiadomości.

Aby rozwiązać problem wydajności, wykorzystajmy klucze sesji (omówione w punkcie 8.2.2). Alicja (1) losowo wybiera symetryczny klucz sesji K_S, (2) szyfruje swoją wiadomość m symetrycznym kluczem K_S, (3) szyfruje klucz symetryczny kluczem publicznym Bartka K_B^+, (4) tworzy „paczkę" z zaszyfrowaną wiadomością oraz zaszyfrowanym kluczem symetrycznym i (5) wysyła paczkę na adres e-mail Bartka. Poszczególne etapy tego procesu zilustrowano na rysunku 8.20 (na tym i na następnych rysunkach otoczony kółkiem plus reprezentuje łączenie, a otoczony kółkiem minus — rozdzielanie). Kiedy Bartek otrzymuje paczkę, (1) używa swojego klucza prywatnego K_B^- do odszyfrowania klucza symetrycznego, po czym (2) używa klucza symetrycznego K_S do odszyfrowania wiadomości m.



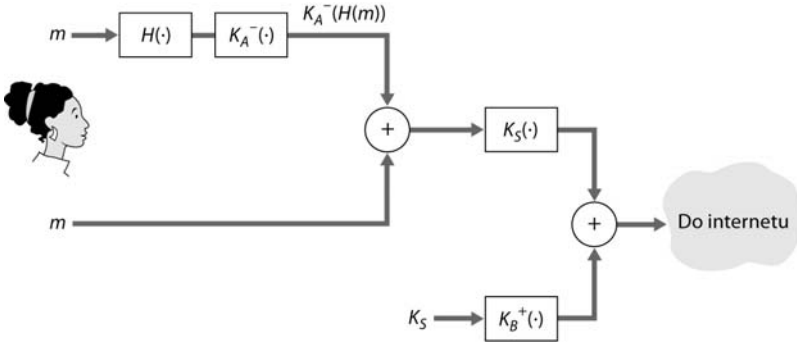
Rysunek 8.20. Alicja używa symetrycznego klucza sesji K_S, aby wysłać Bartkowi tajną wiadomość

Opracowawszy system poczty elektronicznej, który zapewnia poufność, zaprojektujmy teraz kolejny system, który będzie oferował uwierzytelnianie nadawcy oraz ochronę integralności. Załóżmy na chwilę, że Alicja i Bartek nie troszczą się o poufność (chcą podzielić się swoim uczuciem z całym światem!), ale chcą wiedzieć, kto wysłał wiadomość i czy nie została ona zmodyfikowana. Aby osiągnąć ten cel, skorzystamy z podpisów cyfrowych i skrótów wiadomości (opisanych w podrozdziale 8.3). Alicja (1) stosuje funkcję skrótu *H* (na przykład MD5) do swojej wiadomości *m*, aby uzyskać skrót, (2) podpisuje skrót swoim kluczem prywatnym K_A^-, aby utworzyć podpis cyfrowy, (3) tworzy „paczkę" z pierwotnej (niezaszyfrowanej) wiadomości i podpisu, po czym (4) wysyła paczkę na adres e-mail Bartka. Kiedy Bartek otrzymuje paczkę, (1) stosuje klucz publiczny Alicji K_A^+ do podpisanego skrótu i (2) porównuje wynik tej operacji z samodzielnie obliczonym skrótem *H*. Poszczególne etapy tego procesu zilustrowano na rysunku 8.21. Jak wyjaśniono w podrozdziale 8.3, jeśli oba skróty są takie same, Bartek może być pewien, że wiadomość pochodzi od Alicji i nie została zmodyfikowana.



Rysunek 8.21. Wykorzystanie funkcji skrótu i podpisów cyfrowych do uwierzytelniania nadawcy i ochrony integralności

Rozważmy teraz system poczty elektronicznej, który zapewnia poufność, uwierzytelnianie nadawcy *oraz* ochronę integralności. Można to osiągnąć poprzez połączenie procedur pokazanych na rysunkach 8.20 i 8.21. Alicja tworzy najpierw wstępną „paczkę" (dokładnie tak jak pokazano na rysunku 8.21), która składa się z oryginalnej wiadomości i jej cyfrowo podpisanego skrótu. Następnie traktuje tę wstępną paczkę jak nową wiadomość i przetwarza ją w sposób pokazany na rysunku 8.20, tworząc nową paczkę, którą wysyła do Bartka. Czynności wykonywane przez Alicję są zilustrowane na rysunku 8.22. Kiedy Bartek otrzymuje paczkę, najpierw wykonuje czynności ze swojej strony rysunku 8.20, a następnie czynności ze swojej strony rysunku 8.21. Powinno być oczywiste, że projekt ten spełnia wymogi poufności, uwierzytelniania nadawcy i ochrony integralności. Zauważmy, że w tej metodzie Alicja dwukrotnie posługuje się



Rysunek 8.22. Alicja używa kryptografii z kluczem symetrycznym, kryptografii z kluczem publicznym, funkcji skrótu oraz podpisu cyfrowego, aby zapewnić poufność, uwierzytelnianie nadawcy oraz ochronę integralności

kryptografią z kluczem publicznym: raz z wykorzystaniem własnego klucza prywatnego, a raz z wykorzystaniem publicznego klucza Bartka. Podobnie postępuje Bartek, raz wykorzystując swój klucz prywatny, a raz — klucz publiczny Alicji.

KĄCIK HISTORYCZNY

PHIL ZIMMERMAN I PGP

Philip R. Zimmerman napisał program Pretty Good Privacy (PGP). Z tego powodu przez trzy lata toczyło się przeciwko niemu postępowanie sądowe, ponieważ rząd Stanów Zjednoczonych utrzymywał, że opublikowanie bezpłatnej wersji programu w 1991 roku naruszyło ograniczenia eksportu technik kryptograficznych. Ktoś umieścił shareware'ową wersję programu w internecie, skąd mogli ją pobrać obywatele innych krajów. W prawie federalnym Stanów Zjednoczonych oprogramowanie kryptograficzne jest sklasyfikowane jako środek bojowy i objęte zakazem eksportu.

Pomimo braku finansowania, opłacanych pracowników i wspierającej je firmy (a także interwencji rządowej) PGP stało się najpopularniejszym na świecie oprogramowaniem do szyfrowania poczty. Co ciekawe, do sukcesu PGP mógł przyczynić się właśnie rząd Stanów Zjednoczonych i sprawa wytoczona Zimmermanowi.

Rząd oddalił zarzuty na początku 1996 roku, co spotkało się z entuzjazmem internetowych aktywistów. Sprawa Zimmermana zmieniła się w historię niewinnego człowieka walczącego o swoje prawa z bezduszną, biurokratyczną machiną. Ustępstwo rządu było dobrą wiadomością, zwłaszcza w obliczu cenzorskich zapędów Kongresu oraz nacisków FBI na poszerzenie uprawnień w zakresie inwigilacji obywateli.

Po zamknięciu sprawy Zimmerman założył firmę PGP Inc., która została nabyta przez Network Associates w 1997 roku. Obecnie Zimmerman jest niezależnym konsultantem do spraw kryptografii.

Projekt systemu poczty elektronicznej z rysunku 8.22 w większości sytuacji zapewnia wystarczające bezpieczeństwo większości użytkowników. Trzeba jednak rozważyć jeszcze jedną ważną kwestię. Alicja musi zdobyć publiczny klucz Bartka, a Bartek — publiczny klucz Alicji. Dystrybucja tych kluczy publicznych jest poważnym problemem. Inga mogłaby podszyć się pod Bartka i podać Alicji swój klucz publiczny, twierdząc, że jest to klucz Bartka, co pozwoliłoby jej odczytać wiadomość przeznaczoną dla Bartka. Jak dowiedzieliśmy się w podrozdziale 8.3, popularnym sposobem rozpowszechniania kluczy publicznych jest *certyfikowanie* ich przez urząd certyfikacji.

8.4.2. PGP

Pretty Good Privacy (PGP) to oprogramowanie napisane w 1991 roku przez Phila Zimmermana, które stało się faktycznym standardem szyfrowania poczty elektronicznej. Witryna programu wysyła ponad milion stron miesięcznie do użytkowników ze 166 krajów [PGPI 2007]. Istnieje wiele bezpłatnych wersji PGP; oprogramowanie dla różnych platform i sporo ciekawej lektury można znaleźć na przykład w witrynie International PGP Home Page [PGPI 2007] (szczególnie interesujący jest esej autora PGP [Zimmerman 2007]). Działanie PGP jest zasadniczo zgodne z projektem pokazanym na rysunku 8.22. W zależności od wersji PGP używa algorytmów MD5 lub SHA do obliczania skrótów wiadomości, algorytmów CAST, triple-DES lub IDEA do szyfrowania z kluczem symetrycznym oraz algorytmu RSA do szyfrowania z kluczem publicznym.

Po zainstalowaniu PGP tworzy parę kluczy użytkownika. Klucz publiczny można umieścić w osobistej witrynie WWW lub w specjalnym serwerze kluczy. Klucz prywatny jest chroniony hasłem. Użytkownik musi je wprowadzać za każdym razem, kiedy korzysta z klucza prywatnego. PGP oferuje opcje cyfrowego podpisania wiadomości, zaszyfrowania jej albo zarówno podpisania, jak i zaszyfrowania. Na rysunku 8.23 pokazano wiadomość podpisaną przez PGP. Wiadomość ta następuje po nagłówku MIME. Zakodowane dane to $K_A^-(H(m))$, tzn. podpisany cyfrowo skrót wiadomości. Jak już wspomnieliśmy, aby Bartek mógł zweryfikować integralność wiadomości, musi mieć dostęp do publicznego klucza Alicji.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash:  SHA1
Bartku,
Czy zobaczymy się wieczorem?
Żarliwie Twoja, Alicja
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset:  noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

Rysunek 8.23. Wiadomość podpisana przez PGP

Na rysunku 8.24 pokazano tajną wiadomość PGP. Wiadomość ta również jest poprzedzona nagłówkiem MIME. Oczywiście tekst jawny nie jest dołączony do wiadomości. Kiedy nadawca (na przykład Alicja) potrzebuje zarówno poufności, jak i ochrony integralności, PGP osadza wiadomość taką jak ta z rysunku 8.24 w wiadomości z rysunku 8.23.

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=lKhnMikLo0+1/BvcX4T==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE-----
```

Rysunek 8.24. Tajna wiadomość PGP

PGP oferuje też mechanizm certyfikacji kluczy publicznych, który jednak znacznie się różni od tradycyjnych rozwiązań (urzędu certyfikacyjnego). Klucze publiczne PGP są certyfikowane przez *sieć zaufania*. Alicja sama może certyfikować dowolną parę klucz-nazwa użytkownika, jeśli uważa, że klucz rzeczywiście należy do danego użytkownika. Ponadto PGP pozwala Alicji stwierdzić, że darzy zaufaniem pewnego użytkownika, który poręcza za autentyczność innych kluczy. Niektórzy użytkownicy PGP podpisują nawzajem swoje klucze podczas specjalnie organizowanych imprez. Użytkownicy spotykają się, wymieniają dyskietki z kluczami publicznymi i certyfikują nawzajem swoje klucze, podpisując je kluczami prywatnymi.

8.5. Zabezpieczanie połączeń TCP — protokół SSL

W poprzednim podrozdziale pokazaliśmy, w jaki sposób techniki kryptograficzne pozwalają zapewnić poufność, integralność danych i uwierzytelnianie punktów końcowych w konkretnym zastosowaniu — poczcie elektronicznej. W tym podrozdziale przejdziemy o warstwę w dół stosu protokołów i zbadamy, jak za pomocą kryptografii wzbogacić protokół TCP o usługi z zakresu bezpieczeństwa, w tym poufność, integralność danych i uwierzytelnianie punktów końcowych. Ta wzbogacona wersja TCP jest znana pod nazwą **SSL** (ang. *Secure Sockets Layer*). Nieco zmodyfikowana odmiana protokołu SSL w wersji 3., **TLS** (ang. *Transport Layer Security*), została opracowana jako standard przez organizację IETF [RFC 2246].

Protokół SSL opracowała firma Netscape, jednak podstawowe pomysły związane z zabezpieczaniem protokołu TCP są starsze (zobacz na przykład prace Woo [Woo 1994]). Od czasu wprowadzenia protokół SSL znalazł wiele zastosowań. Jest obsługiwany przez wszystkie popularne przeglądarki internetowe i serwery WWW. Korzystają z niego w zasadzie wszystkie witryny z branży handlu

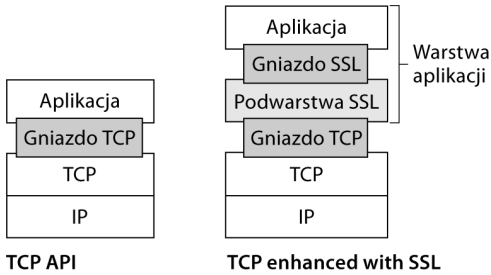
elektronicznego (w tym Amazon, eBay, Yahoo!, MSN itd.). Za pośrednictwem tego protokołu corocznie wydawane są dziesiątki miliardów dolarów. Jeśli Czytelnik kiedykolwiek kupował coś przez internet za pomocą karty kredytowej, to komunikacja między przeglądarką i serwerem odbywała się niemal na pewno przez protokół SSL (kiedy jest on używany przez przeglądarkę, adres URL rozpoczyna się od członu *https*, a nie *http*).

Aby zrozumieć konieczność stosowania protokołu SSL, rozważmy typowy scenariusz handlu internetowego. Bartek surfuje po sieci i trafia do witryny firmy Alicja Sp. z o.o., która sprzedaje perfumy. W witrynie znajduje się formularz, do którego Bartek ma wpisać rodzaj perfum i żądaną ilość, swój adres oraz numer karty kredytowej. Bartek wprowadza te informacje, klika przycisk *Wyślij* i oczekuje na dostawę produktu (na przykład przez listonosza); spodziewa się też znaleźć opłatę za produkt w następnym zestawieniu transakcji wykonanych kartą kredytową. Brzmi świetnie, ale jeśli Bartek nie podejmie odpowiednich środków ostrożności, może go czekać kilka niemiłych niespodzianek.

- Jeśli nie zostanie zachowana poufność (szyfrowanie), intruz może przechwycić zamówienie i uzyskać informacje o karcie kredytowej Bartka. Pozwoli mu to robić zakupy na konto Bartka.
- Jeśli nie zostanie zapewniona integralność danych, intruz może zmodyfikować zamówienie Bartka i doprowadzić do zakupu 10-krotnie większej liczby butelek perfum.
- Jeśli zabraknie uwierzytelniania serwera, serwer może wyświetlić znane logo sklepu Alicja Sp. z o.o., choć właścicielem witryny jest Inga podszywająca się pod firmę Alicji. Po otrzymaniu zamówienia Bartka Inga może ukraść jego pieniądze i zniknąć. Może też skraść tożsamość Bartka po uzyskaniu jego imienia i nazwiska, adresu oraz numeru karty kredytowej.

Protokół SSL pozwala rozwiązać te problemy, ponieważ wzbogaca protokół TCP o funkcje zapewniania poufności, integralności danych oraz uwierzytelnianie serwera i klienta.

Protokołu SSL często używa się do ochrony transakcji realizowanych przez protokół HTTP. Jednak ponieważ SSL zabezpiecza protokół TCP, można go zastosować w dowolnej aplikacji korzystającej z TCP. SSL udostępnia prosty oparty na gniazdach interfejs API, podobny i analogiczny do interfejsu API protokołu TCP. Jeśli programista aplikacji chce zastosować SSL, powinien dołączyć do programu klasy lub biblioteki związane z tym protokołem. Jak przedstawia to rysunek 8.25, choć technicznie protokół SSL działa w warstwie aplikacji, z perspektywy programisty jest to protokół transportowy, świadczący usługi protokołu TCP wzbogacone o usługi z obszaru bezpieczeństwa.



Rysunek 8.25. Choć z technicznego punktu widzenia SSL działa w warstwie aplikacji, to z perspektywy programisty jest to protokół transportowy

8.5.1. Ogólny obraz

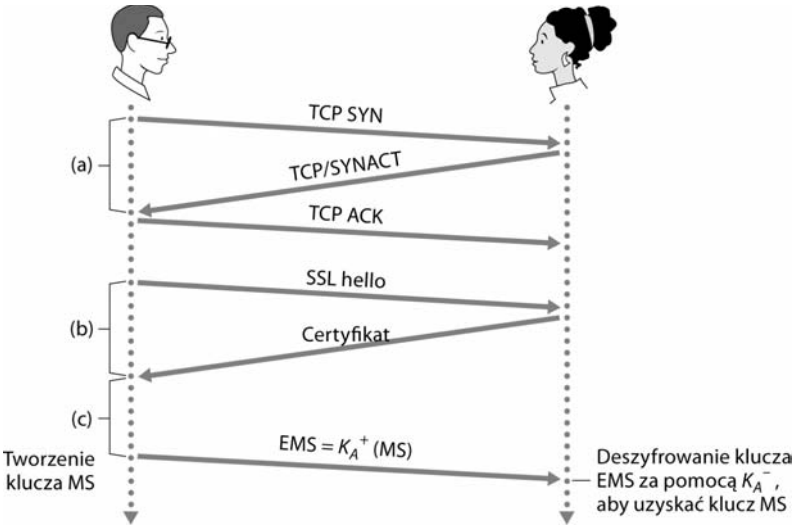
Zaczniemy od opisania uproszczonej wersji protokołu SSL. Umożliwia ona ogólne zrozumienie odpowiedzi na pytania „dlaczego?" i „jak?" związane z tym protokołem. Tę uproszczoną wersję będziemy nazywać „prawie SSL". Po opisaniu „prawie SSL" w następnym punkcie omówimy pełną wersję protokołu. „Prawie SSL" (i SSL) obejmuje trzy etapy: *negocjowanie*, *obliczanie klucza* i *transfer danych*. W tym miejscu opisujemy te trzy fazy sesji komunikacyjnej między klientem (Bartek) i serwerem (Alicja). Alicja ma parę kluczy (prywatny i publiczny) oraz certyfikat wiążący jej tożsamość z kluczem publicznym.

Negocjowanie

W fazie negocjowania Bartek musi (a) nawiązać połączenie TCP z Alicją, (b) zweryfikować, że Alicja to *rzeczywiście* Alicja i (c) przesłać Alicji nadrzędny niejawny klucz, który posłuży obu stronom do wygenerowania wszystkich kluczy symetrycznych potrzebnych w sesji SSL. Te trzy etapy przedstawia rysunek 8.26. Zauważmy, że po nawiązaniu połączenia TCP Bartek wysyła do Alicji komunikat Hello. Następnie Alicja odpowiada przez przesłanie certyfikatu z jej kluczem publicznym. Jak wyjaśniliśmy w podrozdziale 8.3, certyfikat został potwierdzony przez urząd certyfikacyjny, dlatego Bartek jest pewny, że klucz publiczny w tym certyfikacie należy do Alicji. Bartek generuje następnie klucz główny (ang. *Master Secret* — MS; będzie on używany tylko w danej sesji SSL), szyfruje go za pomocą klucza publicznego Alicji, aby utworzyć zaszyfrowany klucz główny (ang. *Encrypted Master Secret* — EMS), i wysyła EMS do Alicji. Alicja odszyfrowuje EMS za pomocą klucza prywatnego, aby uzyskać MS. Po tej fazie Bartek i Alicja (ale nikt inny) znają klucz MS dla sesji SSL.

Obliczanie klucza

Teoretycznie klucz MS, znany teraz Bartkowi i Alicji, można wykorzystać jako symetryczny klucz sesji na potrzeby wszystkich następnych operacji szyfrowania i sprawdzania integralności danych. Jednak ogólnie za bezpieczniejsze dla



Rysunek 8.26. Negocjowanie w protokole „prawie SSL" rozpoczyna się od nawiązania połączenia TCP

Alicji i Bartka uważa się korzystanie przy wykonywaniu tych zadań z różnych kluczy. Dlatego Alicja i Bartek stosują klucz MS do wygenerowania czterech innych kluczy. Są to:

- E_B = szyfrujący klucz sesji dla danych wysyłanych przez Bartka do Alicji;
- M_B = klucz sesji z kodem MAC dla danych wysyłanych przez Bartka do Alicji;
- E_A = szyfrujący klucz sesji dla danych wysyłanych przez Alicję do Bartka;
- M_A = klucz sesji z kodem MAC dla danych wysyłanych przez Alicję do Bartka.

Alicja i Bartek generują te cztery klucze na podstawie klucza MS. Można to zrobić po prostu przez włączenie klucza MS w cztery pozostałe (jednak, jak opisujemy dalej, w *prawdziwym* protokole SSL jest to nieco bardziej skomplikowane). Pod koniec fazy obliczania klucza Alicja i Bartek mają wszystkie cztery klucze. Dwa klucze szyfrujące posłużą do szyfrowania danych, a dwa klucze z kodem MAC — do weryfikowania integralności komunikatów.

Transfer danych

Teraz, kiedy Alicja i Bartek mają te same cztery klucze sesji (E_B, M_B, E_A, M_A), mogą zacząć wysyłać do siebie zabezpieczone dane za pomocą protokołu TCP. Ponieważ TCP to protokół do przesyłania strumienia bajtów, naturalnym podejściem w SSL jest szyfrowanie danych aplikacji „w locie" i przekazywanie w ten sam sposób zaszyfrowanych danych do protokołu TCP. Ale jak przy takim podejściu dodać kod MAC potrzebny do sprawdzenia integralności danych?

Z pewnością niepożądane jest oczekiwanie do końca sesji TCP z weryfikacją integralności wszystkich danych wysyłanych przez Bartka w czasie całej sesji! Aby rozwiązać ten problem, SSL dzieli strumień danych na *rekordy*, dołącza kod MAC do każdego rekordu na potrzeby sprawdzenia integralności, a następnie szyfruje rekord wraz z kodem MAC. W celu utworzenia kodu MAC Bartek przekazuje rekord wraz z kluczem M_B do funkcji skrótu, jak opisano to w podrozdziale 8.3. Aby zaszyfrować rekord wraz z kodem MAC, Bartek stosuje szyfrujący klucz sesji (E_B). Ten zaszyfrowany pakiet jest następnie przekazywany do protokołu TCP w celu przesłania danych przez internet.

Choć to podejście ma wiele zalet, nie zapewnia pełnej ochrony integralności danych w całym strumieniu. Załóżmy, że Inga zajęła pozycję „kobiety w środku" i może wstawiać, usuwać oraz zastępować segmenty w strumieniu segmentów TCP przesyłanym między Alicją i Bartkiem. Inga może na przykład przechwycić dwa segmenty wysłane przez Bartka, zmienić ich kolejność, dostosować do tego numery sekwencyjne TCP (nie są one zaszyfrowane), a następnie przesłać te segmenty do Alicji. Jeśli każdy segment TCP kapsułkuje dokładnie jeden rekord, Alicja przetworzy segmenty w następujący sposób:

1. Protokół TCP po stronie Alicji nie wykryje żadnych problemów i przekaże dwa rekordy do podwarstwy SSL.
2. Protokół SSL po stronie Alicji odszyfruje dwa rekordy.
3. Protokół SSL po stronie Alicji zastosuje kod MAC z obu rekordów do zweryfikowania integralności zapisanych w nich danych.
4. Protokół SSL przekaże następnie zaszyfrowane strumienie bajtów z obu rekordów do warstwy aplikacji. Jednak cały strumień bajtów otrzymany przez Alicję nie będzie poprawny z uwagi na odwrócenie kolejności rekordów!

Zachęcamy Czytelników do zastanowienia się nad podobnymi scenariuszami, w których Inga usuwa lub odtwarza segmenty.

Rozwiązaniem tego problemu — jak Czytelnicy pewnie odgadli — jest zastosowanie numerów sekwencyjnych. W SSL odbywa się to tak: po stronie Bartka działa licznik numerów sekwencyjnych, rozpoczynający odliczanie od zera i zwiększający wartość dla każdego wysłanego rekordu SSL. Bartek nie dołącza numeru sekwencyjnego do samego rekordu, ale uwzględnia ten numer w czasie obliczania kodu MAC. Dlatego kod MAC jest skrótem danych, klucza MAC M_B *i obecnego numeru sekwencyjnego*. Alicja śledzi numery sekwencyjne Bartka, co umożliwia jej zweryfikowanie integralności danych w rekordzie przez uwzględnienie odpowiedniego numeru przy obliczaniu kodu MAC. Wykorzystanie numerów sekwencyjnych protokołu SSL zapobiega przeprowadzeniu przez Ingę ataku „kobieta w środku" polegającego na przykład na zmianie kolejności segmentów lub ich odtworzeniu (dlaczego?).

Rekord SSL

Rysunek 8.27 przedstawia rekord SSL (rekord w protokole „prawie SSL" wygląda tak samo). Rekord ten składa się z pól: typu, wersji, długości, danych i kodu MAC. Zauważmy, że trzy pierwsze pola nie są szyfrowane. Pole typu określa, czy rekord zawiera komunikat negocjowania, czy dane aplikacji. Służy też do zamykania połączenia SSL, co opisujemy dalej. Protokół SSL po stronie odbiorcy używa pola długości do wyodrębnienia rekordów SSL z wejściowego strumienia bajtów TCP. Przeznaczenie pola wersji nie wymaga wyjaśnień.

Rysunek 8.27. Format rekordu SSL

8.5.2. Bardziej kompletny obraz

W poprzednim punkcie omówiliśmy protokół „prawie SSL". Pozwoliło to zrozumieć na podstawowym poziomie odpowiedzi na pytania „dlaczego?" i „jak?" związane z protokołem SSL. Teraz możemy dokładniej zbadać ten protokół i jego kluczowe cechy.

Negocjowanie w protokole SSL

SSL nie wymaga, aby Alicja i Bartek korzystali z określonego algorytmu szyfrowania z kluczem symetrycznym, określonego algorytmu z kluczem publicznym lub określonego algorytmu tworzenia kodu MAC. Protokół ten umożliwia stronom uzgodnienie algorytmów kryptograficznych na początku każdej sesji SSL (na etapie negocjowania). Ponadto w fazie negocjowania Alicja i Bartek przesyłają do siebie identyfikatory jednorazowe służące do tworzenia kluczy sesji (E_B, M_B, E_A, M_A). Oto operacje w fazie negocjowania w prawdziwym SSL:

1. Klient wysyła listę obsługiwanych algorytmów kryptograficznych wraz z identyfikatorem jednorazowym klienta.
2. Z tej listy serwer wybiera algorytm szyfrowania z kluczem symetrycznym (na przykład AES), algorytm szyfrowania z kluczem publicznym (na przykład RSA z kluczem o określonej długości) i algorytm tworzenia kodu MAC. Następnie wysyła do klienta informacje o wybranych algorytmach, certyfikat i identyfikator jednorazowy serwera.
3. Klient weryfikuje certyfikat, wyodrębnia klucz publiczny serwera, generuje klucz PMS (ang. *Pre-Master Secret*), szyfruje klucz PMS za pomocą klucza publicznego serwera i wysyła zaszyfrowany klucz PMS do serwera.

4. Za pomocą tej samej funkcji obliczania klucza (określonej w standardzie SSL) klient i serwer niezależnie od siebie obliczają klucz MS na podstawie klucza PMS i identyfikatorów jednorazowych. Klucz PMS jest następnie dzielony w celu wygenerowania dwóch kluczy szyfrujących i dwóch kodów MAC. Ponadto jeśli wybrany szyfr z kluczem symetrycznym obsługuje wiązanie bloków (umożliwiają to na przykład szyfry 3DES i AES), to na podstawie klucza PMS obliczane są też dwa wektory IV (po jednym dla każdej strony połączenia). Dlatego wszystkie komunikaty przesyłane między klientem i serwerem są szyfrowane oraz uwierzytelniane (za pomocą kodów MAC).
5. Klient wysyła kod MAC obliczony na podstawie wszystkich komunikatów z fazy negocjowania.
6. Serwer wysyła kod MAC obliczony na podstawie wszystkich komunikatów z fazy negocjowania.

Dwa ostatnie kroki chronią fazę negocjowania przed manipulacjami. Aby się o tym przekonać, warto zauważyć, że w kroku 1. klient zwykle przedstawia listę algorytmów. Niektóre z nich są mocne, natomiast inne — słabe. Lista algorytmów jest przesyłana jako tekst jawny, ponieważ jeszcze nie uzgodniono algorytmów i kluczy szyfrujących. Inga („kobieta w środku") może usunąć z listy mocniejsze algorytmy i zmusić w ten sposób klienta do wyboru słabego rozwiązania. Aby zapobiec takim manipulacjom, w kroku 5. klient wysyła kod MAC obliczony na podstawie wszystkich wysłanych i odebranych komunikatów z fazy negocjowania. Serwer może porównać ten kod MAC z kodem MAC wysłanych i otrzymanych wiadomości. Jeśli wystąpi niezgodność, serwer może zerwać połączenie. Podobnie serwer wysyła kod MAC obliczony na podstawie wszystkich zaobserwowanych komunikatów z fazy negocjowania, co umożliwia klientowi wykrycie niezgodności.

Czytelnicy mogą się zastanawiać, do czego służą identyfikatory jednorazowe przesyłane w krokach 1. i 2. Czy numery sekwencyjne nie wystarczą do powstrzymania ataków przez odtwarzanie segmentów? Wystarczą, ale same w sobie nie zapobiegną atakom przez „odtworzenie połączenia". Rozważmy następujący atak tego typu: załóżmy, że Inga podsłuchuje wszystkie wiadomości przesyłane między Alicją i Bartkiem. Następnego dnia Inga podszywa się pod Bartka i wysyła do Alicji dokładnie taką samą sekwencję komunikatów, jaką Bartek wysłał Alicji dzień wcześniej. Jeśli Alicja nie stosuje identyfikatorów jednorazowych, wyśle w odpowiedzi dokładnie taką samą sekwencję komunikatów, co poprzedniego dnia. Alicja nie będzie niczego podejrzewać, ponieważ każda otrzymana wiadomość przejdzie test integralności. Jeśli Alicja to serwer sklepu internetowego, uzna, że Bartek składa drugie zamówienie na ten sam produkt. Natomiast po dodaniu do protokołu identyfikatorów jednorazowych Alicja będzie wysyłać różne identyfikatory w każdej sesji TCP, co spowoduje, że klucze szyfrujące każdego dnia będą inne. Dlatego kiedy Alicja odbierze od Ingi odtworzone rekordy SSL, nie przejdą one testów integralności, a fałszywa

transakcja w sklepie internetowych nie dojdzie do skutku. Podsumujmy te rozważania — w SSL identyfikatory jednorazowe służą do ochrony przed atakami przez „odtwarzanie połączenia", a numery sekwencyjne zabezpieczają przed odtwarzaniem poszczególnych pakietów w bieżącej sesji.

Zamykanie połączenia

W pewnym momencie Bartek lub Alicja zechcą zakończyć sesję SSL. Jednym z rozwiązań jest zezwolenie Bartkowi na zamknięcie sesji przez zakończenie połączenia TCP, czyli wysłanie segmentu TCP FIN do Alicji. Jednak taki naiwny projekt umożliwia przeprowadzenie *ataku przez obcinanie*. Polega on na tym, że Inga ponownie zajmuje pozycję „człowieka w środku" w bieżącej sesji SSL i przedwcześnie kończy ją przez przesłanie segmentu TCP FIN. Jeśli Inga to zrobi, Alicja będzie sądzić, że otrzymała wszystkie dane od Bartka, choć wysłał on dopiero ich część. Rozwiązaniem problemu jest wskazanie w polu typu, czy dany rekord ma kończyć sesję SSL. Choć pole typu jest przesyłane jako tekst jawny, po stronie odbiorcy jest uwierzytelniane za pomocą kodu MAC rekordu. Dzięki temu polu Alicja po otrzymaniu segmentu TCP FIN przed odebraniem zamykającego rekordu SSL wykryje, że dzieje się coś podejrzanego.

To kończy wprowadzenie do protokołu SSL. Czytelnicy dowiedzieli się, że w protokole tym wykorzystano wiele zasad kryptograficznych omówionych w podrozdziałach 8.2 i 8.3. Osoby, które chcą jeszcze lepiej poznać protokół SSL, mogą przeczytać dobrze napisaną książkę na ten temat autorstwa Rescorli [Rescorla 2001].

8.6 Zabezpieczenia w warstwie sieci — IPsec i sieci VPN

Protokół IP security, zwykle nazywany **IPsec**, zapewnia bezpieczeństwo w warstwie sieci. Protokół ten chroni datagramy IP na drodze między dwoma jednostkami z warstwy sieci, na przykład hostami i routerami. Jak wkrótce wyjaśnimy, wiele instytucji (korporacje, jednostki rządowe, organizacje non profit itd.) korzysta z protokołu IPsec do tworzenia **wirtualnych sieci prywatnych** (ang. *Virtual Private Network* — VPN) działających w publicznym internecie.

Zanim zajmiemy się szczegółowo protokołem IPsec, zróbmy krok wstecz i zastanówmy się, co właściwie oznacza poufność w warstwie sieci. Jeśli warstwa sieci ma zapewniać poufność między parą jednostek sieciowych (na przykład między dwoma routerami, dwoma hostami lub routerem i hostem), nadawca musi szyfrować treść wszystkich datagramów przesyłanych do odbiorcy. Zaszyfrowaną treścią może być segment TCP lub UDP, komunikat ICMP itd. Gdyby warstwa sieci świadczyła takie usługi, wszystkie dane wysyłane przez hosty —

w tym poczta, strony WWW, komunikaty kontrolne TCP i komunikaty administracyjne (takie jak ICMP i SNMP) — byłyby ukryte przed intruzami podsłuchującymi sieć. Dlatego mówi się, że warstwa sieci zapewnia pełną ochronę.

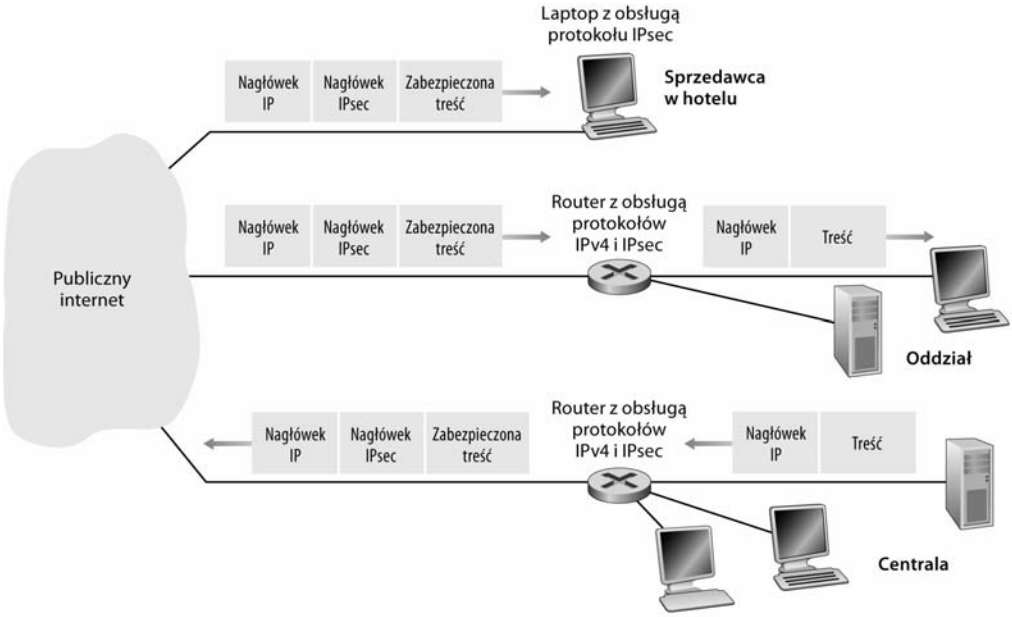
Oprócz poufności warstwa sieci mogłaby zapewniać inne usługi z zakresu bezpieczeństwa, na przykład uwierzytelnianie źródła. Dzięki tej usłudze odbiorca mógłby zweryfikować nadawcę zabezpieczonego datagramu. Bezpieczny protokół warstwy sieci mógłby też zapewniać integralność danych, pozwalając odbiorcy wykryć manipulacje w datagramie, które mogły mieć miejsce w czasie transmisji danych. Następna usługa to zabezpieczanie przed atakami przez odtwarzanie, umożliwiająca Bartkowi wykrycie wszystkich zduplikowanych datagramów wstawionych przez napastnika. Wkrótce pokażemy, że protokół IPsec udostępnia mechanizmy do świadczenia wszystkich tych usług — zapewniania poufności i integralności danych, uwierzytelniania źródła oraz zapobiegania atakom przez odtwarzanie.

8.6.1. IPsec i sieci VPN

Instytucja działająca w wielu miejscach świata często chce mieć własną sieć IP, aby jej hosty i serwery mogły przesyłać dane z zachowaniem bezpieczeństwa i poufności. Aby osiągnąć ten cel, firma może zainstalować niezależną, całkowicie odrębną od publicznego internetu sieć fizyczną z routerami, łączami i infrastrukturą systemu DNS. Takie odłączone sieci przeznaczone do użytku danej organizacji to tak zwane **sieci prywatne**. Nie zaskakuje to, że sieci prywatne bywają bardzo drogie, ponieważ instytucja musi zakupić i zainstalować własną fizyczną infrastrukturę sieciową, a następnie ją konserwować.

Zamiast instalować i konserwować sieć prywatną, obecnie wiele firm tworzy sieci VPN działające w publicznym internecie. W takich sieciach wewnętrzny ruch jest przesyłany przez publiczny internet, a nie przez fizycznie niezależną sieć. Jednak aby zapewnić poufność, ruch wewnętrzny trzeba zaszyfrować przed wysłaniem go do internetu. Prostą przykładową sieć VPN przedstawia rysunek 8.28. Instytucja obejmuje centralę, oddział i mobilnych sprzedawców, którzy zwykle korzystają z internetu w pokojach hotelowych (na rysunku przedstawiono tylko jednego sprzedawcę). Kiedy w ramach tej sieci host z centrali wysyła datagram IP do innego takiego hosta lub hosta z oddziału, korzysta ze zwykłego protokołu IPv4 (bez usług oferowanych przez protokół IPsec). Jednak jeśli dwa hosty komunikują się za pośrednictwem publicznego internetu, dane przed przesłaniem do internetu są szyfrowane.

Aby zrozumieć, jak działa sieć VPN, prześledźmy prosty przykład w kontekście rysunku 8.28. Kiedy host z centrali wysyła datagram IP do sprzedawcy w hotelu, router bramowy w centrali przekształca zwykły datagram IPv4 w datagram IPsec i przekazuje datagram IPsec do internetu. Datagram IPsec ma tradycyjny nagłówek IPv4, aby routery w publicznym internecie przetwarzały



Rysunek 8.28. Sieć VPN

dane tak samo, jak robią to ze zwykłymi datagramami IPv4. Dla routerów zabezpieczony datagram nie różni się niczym od pozostałych. Jednak, jak przedstawia to rysunek 8.28, treść datagramu IPsec obejmuje nagłówek IPsec służący do przetwarzania danych przez omawiany protokół. Ponadto treść datagramu IPsec jest zaszyfrowana. Kiedy datagram IPsec dotrze do laptopa sprzedawcy, system operacyjny odszyfruje treść (i zapewni inne usługi z zakresu bezpieczeństwa, takie jak weryfikacja integralności danych), a następnie przekaże odszyfrowane dane do protokołu wyższej warstwy (na przykład TCP lub UDP).

Właśnie przedstawiliśmy na ogólnym poziomie, jak instytucja może zastosować protokół IPsec do zbudowania sieci VPN. Aby Czytelnicy mogli zobaczyć las spoza drzew, pominęliśmy wiele ważnych szczegółów. Przyjrzyjmy się teraz bliżej temu protokołowi.

8.6.2. Protokoły AH i ESP

IPsec jest dość skomplikowany — jego części są opisane w kilkunastu różnych dokumentach RFC. Dwa ważne dokumenty to RFC 4301, który opisuje ogólną architekturę zabezpieczeń IP, oraz RFC 2411, który zawiera przegląd zestawu protokołów IPsec oraz definiujących go dokumentów. W tej książce nie chcemy powielać treści nudnych i zawiłych dokumentów RFC, ale zastosować bardziej praktyczne i pedagogiczne podejście do opisu protokołów.

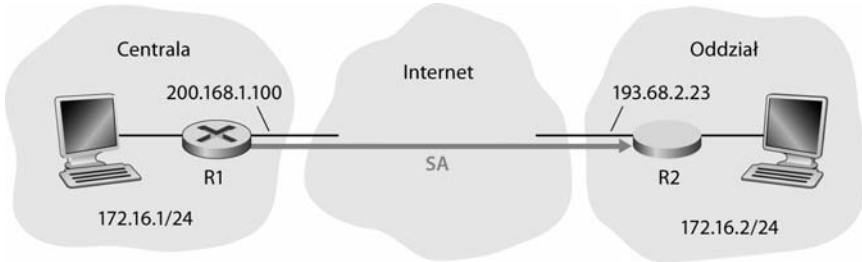
Dwa podstawowe protokoły w zbiorze IPsec to **Authentication Header** (**AH**) oraz **Encapsulation Security Payload** (**ESP**). Kiedy jednostka źródłowa (host lub router) wysyła zabezpieczone datagramy do jednostki docelowej (także hosta lub routera), posługuje się albo protokołem AH, albo ESP. Protokół AH zapewnia uwierzytelnianie źródła i ochronę integralności danych, ale *nie* poufność. Protokół ESP zapewnia uwierzytelnianie, ochronę integralności *oraz* poufność. Ponieważ w sieciach VPN i innych zastosowaniach protokołu IPsec zapewnienie poufności jest często niezbędne, protokół ESP jest stosowany częściej niż protokół AH. Aby objaśnić protokół IPsec i uniknąć wielu jego skomplikowanych aspektów, od tego miejsca skoncentrujemy się wyłącznie na protokole ESP. Czytelników chcących dowiedzieć się więcej o protokole AH zachęcamy do zapoznania się z dokumentami RFC i innymi materiałami dostępnymi w internecie.

8.6.3. Skojarzenia bezpieczeństwa

Datagramy IPsec są przesyłane między parami jednostek sieciowych — między dwoma hostami, dwoma routerami lub hostem i routerem. Przed przesłaniem datagramów IPsec z jednostki źródłowej do docelowej obie strony nawiązują logiczne połączenie w warstwie sieci. To logiczne połączenie to tak zwane **skojarzenie bezpieczeństwa** (ang. *Security Association* — SA). SA to jednokierunkowe połączenie logiczne. Jeśli obie strony chcą przesyłać do siebie datagramy, trzeba nawiązać dwa połączenia logiczne SA (po jednym dla każdego kierunku).

Wróćmy do firmowej sieci VPN z rysunku 8.28. Instytucja obejmuje centralę, oddział i n mobilnych sprzedawców. Na potrzeby przykładu załóżmy, że między centralą i biurem oraz centralą i sprzedawcami w obu kierunkach przesyłane są dane za pomocą protokołu IPsec. Ile połączeń SA istnieje w tej sieci? Aby odpowiedzieć na to pytanie, warto zauważyć, że między routerem bramowym w centrali i routerem bramowym w oddziale działają dwa połączenia SA (po jednym w każdym kierunku). Na każdy laptop sprzedawcy też przypadają dwa połączenia SA między routerem bramowym w centrali i danym laptopem (także po jednym w każdym kierunku). Dlatego łącznie jest $(2+2n)$ połączeń SA. *Warto jednak pamiętać, że nie wszystkie dane przesyłane do internetu z routerów bramowych i laptopów są zabezpieczane za pomocą IPsec.* Host w centrali może chcieć uzyskać dostęp do serwera WWW (na przykład firmy Amazon lub Google) w publicznym internecie. Dlatego routery bramowe (i laptopy) będą wysyłały do internetu zarówno zwykłe datagramy IPv4, jak i zabezpieczone datagramy IPsec.

Zajrzyjmy teraz do „wnętrza" połączenia SA. Aby omówienie było zrozumiałe i konkretne, jako kontekst wykorzystajmy połączenie SA między routerami R1 i R2 (rysunek 8.29). Czytelnicy mogą przyjąć, że router R1 to router bramowy w centrali, a router R2 — router bramowy w oddziale z rysunku 8.28. Router R1 będzie przechowywał informacje o stanie połączenia SA, w tym:



Rysunek 8.29. Połączenie SA między routerami R1 i R2

- 32-bitowy identyfikator połączenia SA, tak zwany **SPI** (ang. *Security Parameter Index*);
- wyjściowy (tu jest to 200.168.1.100) i docelowy interfejs połączenia SA (tu jest to 193.68.2.23);
- typ szyfrowania (na przykład 3DES z wiązaniem bloków);
- klucz szyfrujący;
- rodzaj testu integralności (na przykład HMAC z algorytmem MD5);
- klucz uwierzytelniający.

Kiedy router R1 musi utworzyć datagram IPsec, aby przekazać go przez połączenie SA, sprawdza wymienione wcześniej informacje o stanie w celu określenia, jak powinien uwierzytelnić i zaszyfrować datagram. Router R2 także przechowuje te same informacje o stanie połączenia SA oraz wykorzystuje je do uwierzytelniania i odszyfrowywania datagramów IPsec przesyłanych danym połączeniem.

Jednostka korzystająca z protokołu IPsec (router lub host) często przechowuje informacje o stanie wielu połączeń SA. Na przykład w sieci VPN z rysunku 8.28, w której znajduje się n sprzedawców, router bramowy w centrali przechowuje informacje o stanie $(2+2n)$ połączeń SA. Jednostki korzystające z protokołu IPsec przechowują informacje o stanie wszystkich używanych połączeń SA w **bazie SAD** (ang. *Security Association Database*) — jest to struktura danych w jądrze systemu operacyjnego danej jednostki.

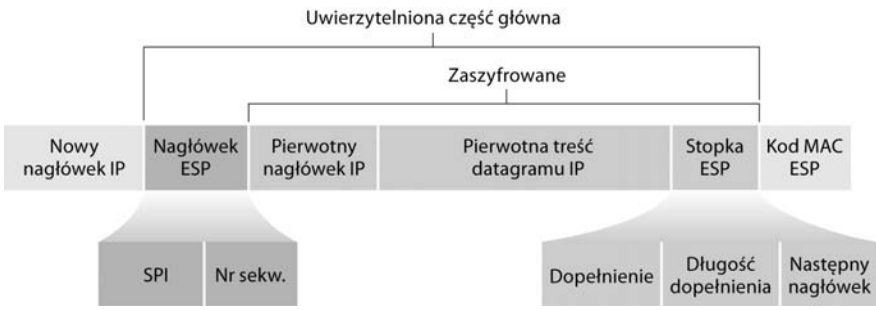
8.6.4. Datagram IPsec

Po omówieniu połączeń SA możemy opisać datagram IPsec. Protokół IPsec ma dwa różne rodzaje pakietów. Jeden z tych typów jest stosowany w **trybie tunelowym**, a drugi — w **trybie transportowym**. Tryb tunelowy, bardziej odpowiedni dla sieci VPN, jest częściej stosowany niż tryb transportowy. Aby dokładniej objaśnić protokół IPsec i uniknąć jego złożonych aspektów, skoncentrujemy

się wyłącznie na trybie tunelowym. Kiedy Czytelnicy dobrze zrozumieją działanie tego trybu, będą mogli łatwo samodzielnie zapoznać się z trybem transportowym.

Format pakietów datagramu IPsec przedstawiono na rysunku 8.30. Czytelnicy mogą sądzić, że formaty pakietów są nudne i mdłe, jednak wkrótce pokażemy, że datagram IPsec może wyglądać i smakować jak specjały z kuchni meksykańskiej! Zbadajmy pola protokołu IPsec w kontekście rysunku 8.29. Załóżmy, że router R1 otrzymuje od hosta 172.16.1.17 (z sieci centrali) zwykły datagram IPv4 przeznaczony dla hosta 172.16.2.48 (z sieci oddziału). Router R1 korzysta z poniższej procedury do przekształcenia „zwykłego datagramu IPv4" w datagram IPsec:

- Dołącza pole „stopki ESP" do końcowej części pierwotnego datagramu IPv4 (obejmuje on pierwotne pola nagłówkowe!).
- Szyfruje efekt poprzedniej operacji za pomocą algorytmu i klucza określonych dla połączenia SA.
- Dołącza z przodu zaszyfrowanych danych pole „nagłówka ESP". Wyjściowy pakiet to tak zwana część główna (ang. *enchilada*).
- Tworzy uwierzytelniający kod MAC dla *całej części głównej* za pomocą algorytmu i klucza określonych przez połączenie SA.
- Dołącza kod MAC za częścią główną, tworząc w ten sposób *treść*.
- W ostatnim kroku tworzy zupełnie nowy nagłówek IP z wszystkimi tradycyjnymi polami nagłówkowymi protokołu IPv4 (zwykle mają one razem 20 bajtów długości), który dołącza przed treścią.



Rysunek 8.30. Format datagramu IPsec

Zauważmy, że wynikowy datagram IPsec to poprawny datagram IPv4 ze zwykłymi polami nagłówkowymi protokołu IPv4, po których następuje treść. Jednak tu treść obejmuje nagłówek ESP, pierwotny datagram IP, stopkę ESP i pole uwierzytelniające ESP (z zaszyfrowanymi pierwotnym datagramem i stopką ESP). Pierwotny datagram IP ma źródłowy adres IP 172.16.1.17 i docelowy adres IP 172.16.2.48. Ponieważ datagram IPsec obejmuje pierwotny datagram IP,

te adresy są dołączone (w zaszyfrowanej postaci) jako fragmenty treści pakietu IPsec. Czym są jednak źródłowy i docelowy adres IP w nowym nagłówku IP, czyli pierwszym od lewej nagłówku datagramu IPsec? Jak można się domyślić, odpowiadają one źródłowemu i docelowemu interfejsowi routera na dwóch końcach tunelu (adresy 200.168.1.100 i 193.68.2.23). Ponadto numer protokołu w nowym polu nagłówkowym datagramu IPv4 nie wskazuje na protokół TCP, UDP lub SMTP, ale ma wartość 50, co oznacza, że jest to datagram IPsec korzystający z protokołu ESP.

Po tym, jak router R1 wyśle datagram IPsec do publicznego internetu, dane zostaną przekazane przez wiele routerów, aż wreszcie dotrą do R2. Każdy z routerów pośrednich przetwarza datagram tak jak zwykłe datagramy, bez świadomości tego, że datagram obejmuje zaszyfrowane dane protokołu IPsec. Dla routerów w publicznym internecie ostatecznym celem datagramu jest R2, ponieważ do tego routera prowadzi docelowy adres IP w nagłówku wyjściowym.

Po omówieniu tworzenia datagramów IPsec przyjrzyjmy się bliżej elementom części głównej. Na rysunku 8.30 widać, że stopka ESP obejmuje trzy pola — dopełnienie, długość dopełnienia i następny nagłówek. Przypomnijmy, że szyfry blokowe wymagają, aby szyfrowana wiadomość miała całkowitą liczbę bloków. Dopełnienie składa się z nieznaczących bitów i jest dodawane do pierwotnego datagramu (wraz z polami długości dopełnienia i następnego nagłówka), aby liczba bloków wynikowej wiadomości była liczbą całkowitą. Pole długości dopełnienia informuje odbiorcę o tym, ile bitów dopełnienia wstawiono (czyli ile trzeba usunąć). Pole następnego nagłówka określa typ danych (na przykład UDP) zapisanych w polu treści. Treść (zwykle pierwotny datagram IP) i stopka ESP są złączane, a następnie szyfrowane.

Przed zaszyfrowanymi danymi znajduje się nagłówek ESP. Jest on wysyłany w formie tekstu jawnego i obejmuje dwa pola — pole z identyfikatorem SPI i pole z numerem sekwencyjnym. SPI informuje odbiorcę o tym, którym kanałem SA jest przesyłany datagram. Odbiorca może wtedy użyć SPI jako indeksu w bazie SAD, aby określić odpowiednie algorytmy i klucze służące do uwierzytelniania oraz odszyfrowywania danych. Pole z numerem sekwencyjnym służy do ochrony przed atakami przez odtwarzanie.

Nadawca dołącza też uwierzytelniający kod MAC. Jak już wspomnieliśmy, nadawca oblicza go dla całej części głównej (składającej się z nagłówka ESP, pierwotnego datagramu IP i stopki ESP, przy czym datagram i stopka są zaszyfrowane). Przypomnijmy, że w celu obliczenia kodu MAC nadawca dołącza do części głównej niejawny klucz MAC, a następnie oblicza skrót o stałej długości na podstawie wynikowej struktury.

Kiedy router R2 odbiera datagram IPsec, wykrywa, że docelowy adres IP to jego adres, dlatego przetwarza datagram. Ponieważ pole protokołu w pierwszym od lewej nagłówku IP ma wartość 50, R2 ustala, że powinien zastosować protokół IPsec ESP do obsługi datagramu. Router po pierwsze: określa na podstawie identyfikatora SPI (po sprawdzeniu zawartości części głównej), do któ-

rego kanału SA datagram należy. Po drugie: oblicza kod MAC części głównej i sprawdza, czy jest zgodny z wartością pola z kodem ESP MAC. Jeśli tak jest, router „wie", że część główna pochodzi od R1 i nie została zmodyfikowana. Po trzecie: R1 sprawdza wartość pola z numerem sekwencyjnym, aby się upewnić, że datagram jest nowy (a nie odtworzony). Po czwarte: odszyfrowuje zaszyfrowane dane za pomocą algorytmu i klucza powiązanych z danym kanałem SA. Po piąte: usuwa dopełnienie i wyodrębnia pierwotny, zwykły datagram IP. Na zakończenie, w szóstym kroku, przekazuje pierwotny datagram do ostatecznej docelowej lokalizacji w sieci oddziału. Nieźle! To naprawdę skomplikowany proces, prawda? Nikt nie mówił, że przygotowywanie i wyodrębnianie części głównej jest proste!

Jest jeszcze jeden niuans, który trzeba opisać. Dotyczy on następującego pytania — kiedy R1 odbiera od hosta z sieci centrali niezabezpieczony datagram przeznaczony dla docelowego adresu IP spoza centrali, to w jaki sposób ustala, czy powinien przekształcić dane na datagram IPsec? Ponadto jeśli dane mają być przetwarzane przez protokół IPsec, to jak R1 określa, którego kanału SA (spośród wielu takich połączeń w bazie SAD routera) należy użyć do utworzenia datagramu IPsec? Oto rozwiązanie tego problemu. Obok bazy SAD jednostka obsługująca protokół IPsec przechowuje inną strukturę danych — **bazę SPD** (ang. *Security Policy Database*). Ta baza określa, które datagramy (ważny jest źródłowy adres IP, docelowy adres IP i typ protokołu) powinny być przetwarzane przez protokół IPsec. Dla takich datagramów podany jest też kanał SA, którego należy użyć. Informacje zapisane w bazie SPD w pewnym sensie określają, „co" zrobić z odebranym datagramem, podczas gdy dane w bazie SAD wskazują, „jak" to zrobić.

Podsumowanie usług protokołu IPsec

Jakie więc dokładnie usługi udostępnia protokół IPsec? Zbadajmy je z perspektywy napastnika, na przykład Ingi. Zajęła ona pozycję „kobiety w środku" na ścieżce między routerami R1 i R2 z rysunku 8.29. Załóżmy, że Inga nie zna kluczy uwierzytelniającego i szyfrującego stosowanych w kanale SA. Co Inga może zrobić, a co się jej nie uda? Po pierwsze, Inga nie może podejrzeć pierwotnego datagramu. Ukryte są przed nią nie tylko dane z tego datagramu, ale też numer protokołu, źródłowy adres IP i docelowy adres IP. Na temat datagramów wysyłanych przez kanał SA Inga może dowiedzieć się tylko tego, że pochodzą od hosta z sieci 172.16.1.0/24 i są przeznaczone dla hosta z sieci 172.16.2.0/24. Napastnik nie może ustalić, czy datagram obejmuje dane TCP, UDP lub ICPM albo dane aplikacji korzystającej z protokołu HTTP, SMTP lub innego. Dlatego poufność jest tu zapewniona na znacznie wyższym poziomie niż w protokole SSL. Po drugie, załóżmy, że Inga chce zmodyfikować datagram w kanale SA przez zamianę niektórych bitów. Kiedy zmodyfikowany datagram dotrze do routera R2, test integralności (oparty na kodzie MAC) zakończy się niepowodzeniem, co ponownie zniweczy złośliwe próby ataku Ingi. Po trzecie,

przyjmijmy, że Inga próbuje podszyć się pod router R1 przez utworzenie datagramu IPsec o adresie źródłowym 200.168.1.100 i adresie docelowym 193.68.2.23. Atak ten będzie nieudany, ponieważ test integralności w routerze R2 ponownie zakończy się niepowodzeniem. Wreszcie z uwagi na to, że datagramy IPsec obejmują numery sekwencyjne, Indze nie uda się przeprowadzić ataku przez odtwarzanie. Podsumujmy te rozważania — jak wspomnieliśmy na początku podrozdziału, protokół IPsec zapewnia poufność, uwierzytelnianie źródła, integralność danych i ochronę przed atakami przez odtwarzanie dowolnej parze urządzeń przekazujących pakiety w warstwie sieci.

8.6.5. IKE — zarządzanie kluczami w protokole IPsec

Jeśli dana sieć VPN składa się z niewielu punktów końcowych (na przykład tylko z dwóch routerów, jak na rysunku 8.29), administrator sieci może ręcznie wprowadzić informacje na temat kanałów SA (algorytmy i klucze szyfrujące oraz uwierzytelniające i identyfikatory SPI) do baz SAD tych punktów. Ręczne przypisywanie kluczy jest oczywiście niepraktyczne w dużych sieciach VPN, które mogą składać się z setek, a nawet tysięcy routerów i hostów obsługujących protokół IPsec. Rozbudowane, rozproszone geograficznie instalacje wymagają automatycznych mechanizmów tworzenia kanałów SA. W IPsec służy do tego protokół IKE (ang. *Internet Key Exchange*) opisany w dokumencie RFC 4306.

Protokół IKE pod niektórymi względami działa podobnie jak proces negocjowania w protokole SSL (zobacz podrozdział 8.5). Każda jednostka korzystająca z IPsec posiada certyfikat obejmujący klucz publiczny. Protokół IKE, podobnie jak SSL, umożliwia dwóm urządzeniom wymianę certyfikatów, negocjowanie algorytmów uwierzytelniania i szyfrowania oraz bezpieczną wymianę materiałów związanych z kluczami w celu utworzenia kluczy sesji dla kanału IPsec SA. W IKE — inaczej niż w SSL — operacje te są wykonywane w dwóch fazach.

Zbadajmy te dwa etapy w kontekście dwóch routerów z rysunku 8.29 (R1 i R2). Pierwsza faza obejmuje dwie wymiany par komunikatów między tymi routerami.

- W czasie pierwszej wymiany wiadomości obie strony stosują algorytm Diffiego-Hellmana (zobacz problemy w końcowej części rozdziału) do utworzenia dwukierunkowego kanału **IKE SA** między routerami. Sytuację komplikuje to, że dwukierunkowe kanały IKE SA działają zupełnie inaczej niż kanały IPsec SA, omówione w punktach 8.6.3 i 8.6.4. IKE SA to uwierzytelniony i szyfrowany kanał łączący dwa routery. W czasie pierwszej wymiany pary komunikatów ustalane są klucze szyfrujący i uwierzytelniający na potrzeby kanału IKE SA. Określany jest też klucz MS, który w drugiej fazie posłuży do obliczenia kluczy dla połączeń IPsec SA. Warto zauważyć, że w pierwszym etapie nie są stosowane publiczne i prywatne klucze RSA.

W szczególności ani R1, ani R2 nie ujawniają swojej tożsamości przez podpisanie komunikatów za pomocą klucza prywatnego.

- W trakcie drugiej wymiany wiadomości obie strony ujawniają przed sobą tożsamość przez podpisywanie komunikatów. Tej tożsamości nie pozna jednak pasywna jednostka podsłuchująca, ponieważ wiadomości są przesyłane przez zabezpieczony kanał IKE SA. Ponadto w tej fazie strony uzgadniają algorytmy szyfrujący i uwierzytelniający na potrzeby kanału IPsec SA.

W drugiej fazie działania protokołu IKE strony tworzą kanały SA w obu kierunkach. Pod koniec tego etapu obie strony kanałów SA znają klucze sesji potrzebne do szyfrowania i uwierzytelniania danych. Następnie jednostki mogą wykorzystać kanały do przesyłania zabezpieczonych datagramów, jak opisaliśmy to w punktach 8.6.3 i 8.6.4. Protokół IKE działa dwuetapowo głównie ze względu na koszty obliczeniowe. Ponieważ druga faza nie wymaga zastosowania kryptografii z kluczem publicznym, IKE pozwala przy stosunkowo niewielkich kosztach obliczeniowych utworzyć dużą liczbę kanałów SA między dwoma jednostkami obsługującymi IPsec.

8.7 Zabezpieczanie bezprzewodowych sieci lokalnych

Bezpieczeństwo jest szczególnie istotną kwestią w sieciach bezprzewodowych, w których fale radiowe przenoszące ramki mogą wydostawać się daleko poza budynek ze stacjami bazowymi i hostami. W tym podrozdziale przedstawiamy krótkie wprowadzenie do zabezpieczeń sieci bezprzewodowych. Bardziej szczegółowe omówienie Czytelnicy znajdą w dobrze napisanej książce Edneya i Arbaugha [Edney 2003].

Kwestia bezpieczeństwa w sieciach 802.11 zyskała spory rozgłos zarówno w kręgach technicznych, jak i w mediach. Dyskusja była gorąca, ale jednostronna — panuje powszechna zgoda, że pierwotna specyfikacja 802.11 ma poważne luki w zabezpieczeniach. Rzeczywiście, obecnie z internetu można pobrać bezpłatne programy, które wykorzystują te luki, przez co ci, którzy używają standardowych mechanizmów bezpieczeństwa 802.11, są równie podatni na atak jak osoby, które nie korzystają z żadnych zabezpieczeń.

W kolejnych podpunktach omówimy mechanizmy bezpieczeństwa opisane w pierwotnej specyfikacji 802.11, zbiorowo określane nazwą **Wired Equivalent Privacy (WEP)**. Jak sama nazwa wskazuje, WEP ma zapewniać poziom bezpieczeństwa zbliżony do tego w sieciach przewodowych. Następnie zbadamy kilka luk w zabezpieczeniach WEP, po czym opiszemy standard 802.11i, znacznie bezpieczniejszą wersję 802.11 przyjętą w 2004 roku.

8.7.1. Wired Equivalent Privacy (WEP)

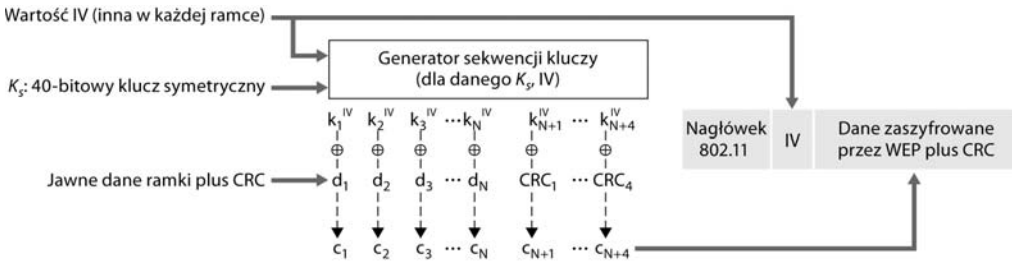
Protokół IEEE 802.11 WEP [IEEE 802.11 2009] zapewnia uwierzytelnianie i szyfrowanie danych między hostem a bezprzewodowym punktem dostępowym (tzn. stacją bazową) z wykorzystaniem wspólnego klucza symetrycznego. WEP nie definiuje algorytmu zarządzania kluczem, a zatem zakłada się, że host i punkt dostępowy uzgodniły klucz jakąś metodą pozapasmową. Uwierzytelnianie odbywa się tak:

1. Host bezprzewodowy żąda uwierzytelnienia przez punkt dostępowy.
2. Punkt dostępowy odpowiada na żądanie uwierzytelniania 128-bitowym identyfikatorem jednorazowym.
3. Host bezprzewodowy szyfruje identyfikator symetrycznym kluczem współdzielonym z punktem dostępowym.
4. Punkt dostępowy odszyfrowuje identyfikator zaszyfrowany przez host.

Jeśli odszyfrowany identyfikator jednorazowy odpowiada wartości uprzednio wysłanej do hosta, host zostaje uwierzytelniony.
Algorytm szyfrowania danych WEP zilustrowano na rysunku 8.31. Tajny, 40-bitowy klucz symetryczny K_S jest znany zarówno hostowi, jak i punktowi dostępowemu. Do 40-bitowego klucza dołączany jest 24-bitowy wektor IV w celu utworzenia 64-bitowego klucza, który zostanie wykorzystany do zaszyfrowania pojedynczej ramki. Wartość IV zmienia się co ramkę, zatem każda ramka jest szyfrowana innym 64-bitowym kluczem. Szyfrowanie przebiega w następujący sposób: najpierw na podstawie treści ramki obliczany jest 4-bajtowy kod CRC (podrozdział 5.2). Treść i cztery bajty kodu CRC są następnie szyfrowane za pomocą strumieniowego algorytmu RC4. Nie będziemy tu szczegółowo omawiać algorytmu RC4 (więcej informacji można znaleźć w [Schneier 1995] i [Edney 2003]). Do naszych celów wystarczy wiedzieć, że na podstawie wartości klucza (w tym przypadku 64-bitowego klucza (K_S, IV)) tworzy on strumień wartości k_1^{IV}, k_2^{IV}, k_3^{IV} … używanych do szyfrowania kolejnych ramek i ich kodów CRC. Dla uproszczenia przyjmijmy, że operacje te są wykonywane na pojedynczych bajtach. Szyfrowanie polega na wykonaniu operacji XOR na i-tym bajcie danych (d_i) oraz i-tym kluczu (k_i^{IV}) ze strumienia wartości generowanych z pary (K_S, IV) w celu uzyskania i-tego bajta szyfrogramu (c_i):

$$c_i = d_i \oplus k_i^{IV}$$

Wartość IV zmienia się co ramkę i jest dołączana *tekstem jawnym* do nagłówka każdej ramki 802.11 zaszyfrowanej przez WEP, jak pokazano na rysunku 8.31. Odbiorca bierze 40-bitowy klucz współdzielony z nadawcą, dołącza do niego wartość IV i wykorzystuje 64-bitowy klucz (identyczny z tym, którego nadawca użył podczas szyfrowania) do odszyfrowania ramki:



Rysunek 8.31. Protokół 802.11 WEP

$$d_i = c_i \oplus k_i^{IV}$$

Prawidłowe użycie algorytmu RC4 wymaga, aby ta sama 64-bitowa wartość klucza *nigdy* nie była wykorzystana więcej niż jeden raz. Przypomnijmy sobie, że klucz WEP zmienia się w każdej ramce. Dla danego klucza K_S (który zmienia się rzadko, jeśli w ogóle) oznacza to, że istnieje tylko 2^{24} niepowtarzalnych kluczy. Jeśli te klucze są wybierane losowo, możemy dowieść [Walker 2000], że prawdopodobieństwo wybrania tej samej wartości IV (a zatem użycia tego samego 64-bitowego klucza) wynosi ponad 99% po zaledwie 12 000 ramek. Jeśli ramka ma rozmiar 1 kilobitu, a szybkość transmisji wynosi 11 Mb/s, przesłanie 12 000 ramek zajmuje zaledwie kilka sekund. Co więcej, wartość IV jest przesyłana tekstem jawnym, więc podsłuchujący będzie wiedział, że użyto zduplikowanej wartości IV.

Aby zobaczyć jeden z kilku problemów związanych ze zduplikowanymi kluczami, rozważmy poniższy atak z wybranym tekstem jawnym przeprowadzany przez Ingę przeciwko Alicji. Przypuśćmy, że Inga (na przykład fałszując adresy IP) wysyła do Alicji żądanie (HTTP lub FTP) przesłania pliku o znanej zawartości d_1, d_2, d_3, d_4, … Inga obserwuje również zaszyfrowane dane c_1, c_2, c_3, c_4, … Ponieważ $d_i = c_i \oplus k_i^{IV}$, to jeśli wykonamy operację XOR c_i na każdej stronie równania, otrzymamy:

$$d_i \oplus c_i = k_i^{IV}$$

Dysponując tym równaniem, Inga może użyć znanych wartości d_i oraz c_i, aby obliczyć k_i^{IV}. Kiedy Inga zaobserwuje kolejne użycie tej samej wartości IV, będzie znała sekwencję k_1^{IV}, k_2^{IV}, k_3^{IV} …, a zatem zdoła odszyfrować wiadomość.

WEP ma jeszcze inne luki w zabezpieczeniach. [Fluhrer 2001] opisuje atak wykorzystujący znaną słabość algorytmu RC4 w razie wyboru pewnych słabych kluczy. [Stubblefield 2002] omawia wydajne sposoby implementowania i realizowania tego ataku. Inny problem z WEP wiąże się z bitami CRC pokazanymi na rysunku 8.31 i przesyłanymi w ramce 802.11 w celu wykrywania zmienionych bitów treści. Napastnik, który chce zmienić zaszyfrowaną treść

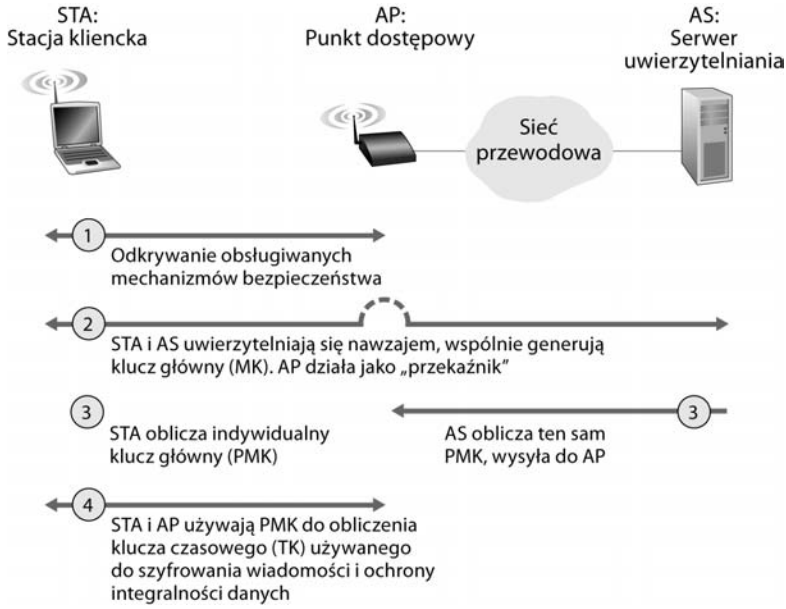
(na przykład wstawić „śmieci" w miejsce oryginalnych, zaszyfrowanych danych), może obliczyć kod CRC podstawionej treści i umieścić go w ramce WEP, aby uzyskać ramkę 802.11, która zostanie przyjęta przez odbiorcę. Do wykrywania modyfikacji i podstawiania treści potrzebne są techniki ochrony integralności podobne do tych, które omówiliśmy w podrozdziale 8.3. Więcej informacji o bezpieczeństwie WEP można znaleźć w [Edney 2003; Walker 2000; Weatherspoon 2000, 802.11 Security 2009] oraz w wymienionych tam źródłach.

8.7.2. IEEE 802.11i

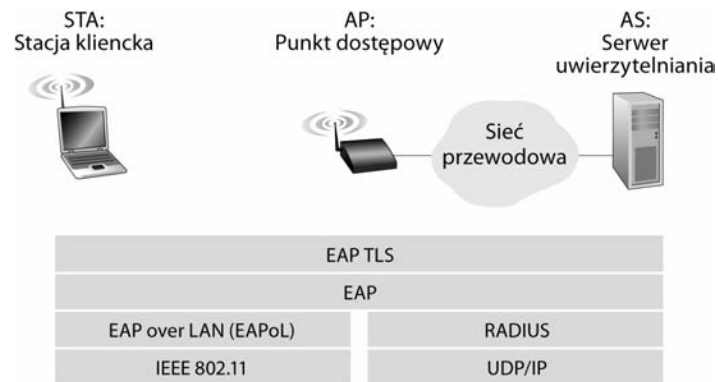
Wkrótce po opublikowaniu standardu IEEE 802.11 w 1999 roku rozpoczęły się prace nad nową, ulepszoną wersją 802.11 z silniejszymi mechanizmami bezpieczeństwa. Nowy standard, znany jako 802.11i, został przyjęty na początku 2004 roku. W przeciwieństwie do protokołu WEP, który zapewnia względnie słabe szyfrowanie, tylko jeden sposób uwierzytelniania i nie jest wyposażony w mechanizm dystrybucji kluczy, IEEE 802.11i oferuje znacznie silniejsze metody szyfrowania, rozszerzalny zbiór mechanizmów uwierzytelniania oraz mechanizm dystrybucji kluczy. Poniżej przedstawiamy krótki przegląd 802.11i; doskonałym omówieniem technicznym (w postaci dźwiękowej) jest [TechOnline 2004].

Na rysunku 8.32 pokazano architekturę standardu 802.11i. Oprócz klienta bezprzewodowego i punktu dostępowego standard 802.11i definiuje serwer uwierzytelniania, z którym może komunikować się punkt dostępowy. Dzięki oddzieleniu serwera uwierzytelniania od punktu dostępowego jeden serwer może obsługiwać wiele punktów, centralizując decyzje (często poufne) dotyczące uwierzytelniania i dostępu oraz obniżając koszt i złożoność punktów dostępowych. Protokół 802.11i składa się z czterech etapów:

1. *Odkrywanie.* W fazie odkrywania punkt dostępowy ogłasza swoją obecność oraz formy uwierzytelniania i szyfrowania, które może zaoferować klientom bezprzewodowym. Następnie klient żąda konkretnej formy uwierzytelniania i szyfrowania. Choć klient i punkt dostępowy wymieniają już komunikaty, klient nie jest jeszcze uwierzytelniony i nie ma klucza szyfrowania. Potrzebnych będzie jeszcze kilka etapów, zanim klient będzie mógł porozumiewać się z dowolnym hostem przez kanał bezprzewodowy.
2. *Wzajemne uwierzytelnianie i generowanie klucza głównego.* Uwierzytelnianie zachodzi między klientem bezprzewodowym a serwerem uwierzytelniania. W tej fazie punkt dostępowy pełni funkcję przekaźnika komunikatów przesyłanych między klientem a serwerem. Protokół **EAP** (ang. *Extensible Authentication Protocol*) [RFC 2284] określa formaty komunikatów międzywęzłowych stosowane w prostym trybie interakcji żądanie-odpowiedź między klientem i serwerem uwierzytelniania. Jak pokazano na rysunku 8.33, komunikaty EAP są kapsułkowane z wykorzystaniem **EAPoL** (EAP over LAN, [IEEE 802.IX]) i wysyłane



Rysunek 8.32. Cztery fazy protokołu 802.11i



Rysunek 8.33. EAP to protokół używany przez węzły końcowe. Komunikaty EAP są kapsułkowane w EAPoL na łączu bezprzewodowym między klientem a punktem dostępowym oraz w UDP/IP między punktem dostępowym a serwerem uwierzytelniania

przez łącze bezprzewodowe 802.11. Komunikaty te są odkapsułkowywane w punkcie dostępowym i kapsułkowane ponownie przy użyciu protokołu **RADIUS** w celu przesłania ich przez UDP/IP do serwera uwierzytelniania. Choć serwer i protokół RADIUS [RFC 2865] nie są wymagane przez protokół 802.11i, *de facto* są jego standardowymi komponentami. Niedawno ustandaryzowany protokół **DIAMETER** [RFC 3588] prawdopodobnie niedługo zastąpi protokół RADIUS.

W protokole EAP serwer uwierzytelniania może wybrać jedną z kilku metod uwierzytelniania. Choć standard 802.11i nie wymusza konkretnej metody, często używa się uwierzytelniania EAP-TLS [RFC 2716]. EAP-TLS korzysta z technik klucza publicznego (w tym z szyfrowanych identyfikatorów jednorazowych oraz skrótów wiadomości) podobnych do tych, które przestudiowaliśmy w podrozdziale 8.3. Pozwalają one na wzajemne uwierzytelnienie klienta i serwera oraz wygenerowanie znanego im obu klucza głównego (ang. *Master Key* — MK).

3. *Generacja indywidualnego klucza głównego.* Klucz MK to wspólny sekret klienta i serwera uwierzytelniania, które na jego podstawie generują drugi klucz, zwany indywidualnym kluczem głównym (ang. *Pairwise Master Key* — PMK). Następnie serwer uwierzytelniania wysyła klucz PMK do punktu dostępowego. I właśnie o to chodziło! Teraz klient i punkt dostępowy mają wspólny klucz (jak pamiętamy, w WEP w ogóle nie uwzględniono dystrybucji kluczy) i są wzajemnie uwierzytelnione. Są niemal gotowe, aby zabrać się do pracy.
4. *Generacja klucza czasowego.* Dysponując kluczem PMK, klient bezprzewodowy i punkt dostępowy mogą generować dodatkowe klucze, które zostaną wykorzystane do komunikacji. Szczególnie godny uwagi jest klucz czasowy (ang. *Temporal Key* — TK), który posłuży do szyfrowania danych wysyłanych przez łącze bezprzewodowe do dowolnego zdalnego hosta.

802.11i zapewnia kilka postaci szyfrowania, w tym szyfrowanie oparte na algorytmie AES oraz wzmocnioną wersję szyfrowania WEP.

8.8. Bezpieczeństwo operacyjne — zapory i systemy wykrywania włamań

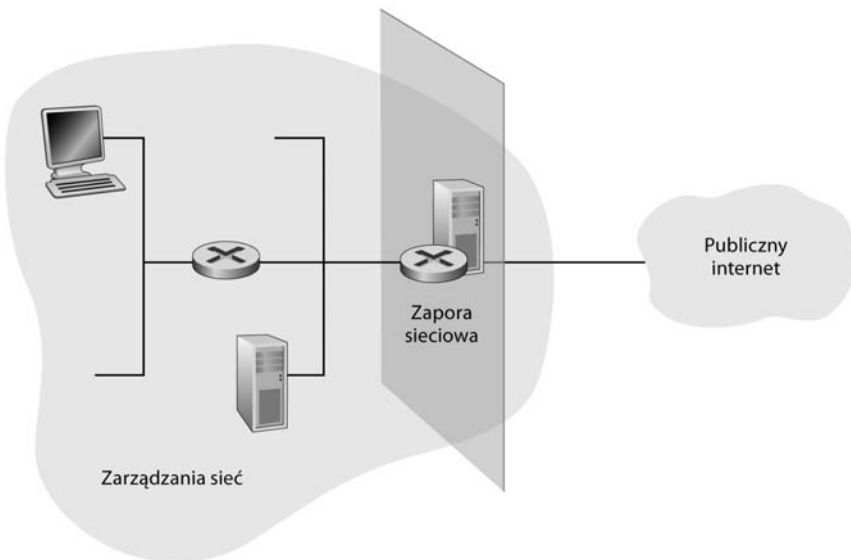
W tym rozdziale wielokrotnie podkreślaliśmy, że internet nie jest bezpiecznym miejscem — roi się w nim od „złych facetów", którzy nieustannie sieją spustoszenie. Ponieważ internet jest z natury wrogi, warto zastanowić się nad sieciami firm i zarządzającymi nimi administratorami. Z punktu widzenia administratora sieci świat dzieli się na dwa obozy: dobrych (którzy należą do organizacji zarządzającej siecią i powinni mieć względnie nieskrępowany dostęp do zasobów sieci) oraz złych (wszystkich pozostałych, których dostęp powinien być ściśle nadzorowany). W wielu organizacjach, zaczynając od średniowiecznych zamków, a kończąc na współczesnych budynkach biurowych, istnieje jeden punkt wejścia i wyjścia, w którym kontrolowane są wszystkie osoby przybywające do organizacji lub opuszczające ją. W zamku robi się to w bramie na końcu mostu zwodzonego; w biurowcu — na portierni. W sieci komputerowej, w której ruch

przychodzący i wychodzący jest kontrolowany, rejestrowany, odrzucany lub przepuszczany, zajmują się tym zapory sieciowe, systemy wykrywania włamań (ang. *Intrusion Detection System* — IDS) i systemy zapobiegania włamaniom (ang. *Intrusion Prevention System* — IPS).

8.8.1. Zapory sieciowe

Zapora sieciowa to kombinacja sprzętu i oprogramowania, która izoluje wewnętrzną sieć organizacji od internetu, przepuszczając niektóre pakiety i blokując inne. Zapora sieciowa pozwala administratorowi kontrolować dostęp do zasobów sieci poprzez zarządzanie ruchem płynącym do nich i w przeciwnym kierunku. Zapory mają trzy cechy:

- *Cały ruch płynący spoza sieci do niej i w przeciwnym kierunku przechodzi przez zaporę.* Na rysunku 8.34 przedstawiono zaporę umieszczoną na granicy między zarządzaną siecią a resztą internetu. Choć duże organizacje mogą używać zapór wielopoziomowych lub rozproszonych [Skoudis 2006], umieszczenie zapory w pojedynczym punkcie dostępu (jak na rysunku 8.34) ułatwia zarządzanie nią i egzekwowanie polityki bezpieczeństwa.



Rysunek 8.34. Zapora między zarządzaną siecią a resztą świata

- *Przez zaporę może przejść tylko autoryzowany ruch (zgodnie z lokalną polityką bezpieczeństwa).* Ponieważ cały ruch kierowany do sieci instytucji i wysyłany z niej przechodzi przez zaporę, urządzenie to może ograniczać dostęp i przepuszczać tylko autoryzowany ruch.

- *Sama zapora jest odporna na atak.* Sama zapora to urządzenie podłączone do internetu. Jeśli urządzenie nie jest prawidłowo zaprojektowane lub zainstalowane, jego zabezpieczenia mogą zostać złamane. Zapora daje wtedy fałszywe poczucie bezpieczeństwa (jest to gorsze, niż kiedy firma w ogóle nie korzysta z takiego urządzenia!).

Obecnie czołowi producenci zapór to Cisco i Check Point. Można też łatwo utworzyć zaporę (filtr pakietów) na komputerze z systemem Linux, używając narzędzia iptables (jest to bezpłatne oprogramowanie powszechnie udostępniane wraz z systemem Linux).

Zapory sieciowe można podzielić na trzy kategorie: **tradycyjne filtry pakietów**, **filtry stanowe** i **bramy aplikacyjne**. Opiszemy je w kolejnych podpunktach.

Tradycyjne filtry pakietów

Jak pokazano na rysunku 8.34, organizacja zazwyczaj ma router, który łączy jej wewnętrzną sieć z dostawcą ISP (a przez niego z publicznym internetem). Cały ruch przychodzący i wychodzący przechodzi przez ten router i właśnie w nim odbywa się **filtrowanie pakietów**. Filtry pakietów analizują nagłówki datagramów, a następnie korzystają z określonych przez administratora reguł filtrowania, aby ustalić, czy należy odrzucić datagram, czy też go przepuścić. Decyzję zwykle podejmuje się na podstawie:

- źródłowego lub docelowego adresu IP;
- typu protokołu określonego w polu datagramu IP — TCP, UDP, ICMP, OSPF i tak dalej;
- źródłowego lub docelowego portu TCP albo UDP;
- bitów flag protokołu TCP — SYN, ACK i tak dalej;
- typu komunikatu ICMP;
- różnych reguł dla datagramów opuszczających sieć i przychodzących do niej;
- różnych reguł dla poszczególnych interfejsów routera.

Administrator sieci konfiguruje zaporę na podstawie polityki organizacji. Może ona uwzględniać produktywność użytkowników i wykorzystanie przepustowości, a także kwestie związane z bezpieczeństwem firmy. W tabeli 8.5 wymieniono kilka przykładowych zasad, a także sposoby ich realizowania za pomocą filtru pakietów. Na przykład jeśli firma nie chce akceptować żadnych przychodzących połączeń TCP oprócz tych skierowanych do jej publicznego serwera WWW, może zablokować wszystkie otrzymane segmenty TCP SYN z wyjątkiem tych, które mają docelowy port 80 i docelowy adres IP serwera WWW firmy. Jeśli organizacja nie chce, aby użytkownicy wykorzystywali dostępną przepustowość na słuchanie radia internetowego, może zablokować

Zasada	Ustawienia zapory
Blokowanie dostępu do stron WWW.	Usuwanie wszystkich wychodzących pakietów skierowanych pod dowolny adres IP i port 80.
Blokowanie przychodzących połączeń TCP oprócz tych skierowanych do publicznego serwera WWW firmy.	Usuwanie wszystkich przychodzących pakietów TCP SYN skierowanych pod dowolny adres IP oprócz adresu 130.207.244.203, port 80.
Zapobieganie wykorzystaniu dostępnej przepustowości na słuchanie radia internetowego.	Usuwanie wszystkich przychodzących pakietów UDP oprócz pakietów systemu DNS.
Zapobieganie wykorzystaniu sieci do „ataku smerfów".	Usuwanie wszystkich pakietów ping protokołu ICMP, które mają „rozgłaszać" adres (na przykład 130.207.255.255).
Zapobieganie badaniu sieci za pomocą narzędzia traceroute.	Usuwanie wszystkich wychodzących wygasłych komunikatów ICMP TTL.

Tabela 8.5. Zasady i odpowiadające im reguły filtrowania dla firmowej sieci 130.27/16 z serwerem WWW dostępnym pod adresem 130.207.244.203

cały niekrytyczny ruch UDP (ponieważ radia internetowe zwykle prowadzą transmisję za pomocą protokołu UDP). Jeśli firma nie chce, aby osoby z zewnętrz mapowały jej wewnętrzną sieć (za pomocą narzędzia traceroute), może zablokować wszystkie wygasłe komunikaty ICMP TTL wychodzące z sieci organizacji.

Polityka filtrowania może również opierać się na kombinacji adresów i numerów portów. Przykładowo router może blokować wszystkie datagramy Telnetu (z numerem portu równym 23) oprócz tych, które zmierzają pod określone adresy lub spod nich przybywają. Taka polityka dopuszcza połączenia Telnetu z niektórymi hostami. Niestety, oparcie polityki na zewnętrznych adresach nie zabezpiecza przed datagramami ze sfałszowanym adresem źródłowym.

Można też filtrować pakiety na podstawie bitu TCP ACK. Jest to bardzo przydatne, jeśli organizacja chce pozwolić wewnętrznym klientom na łączenie się z zewnętrznymi serwerami, ale zapobiec łączeniu się zewnętrznych klientów z wewnętrznymi serwerami. Jak podano w podrozdziale 3.5, pierwszy segment w każdym połączeniu TCP ma wyzerowany bit ACK, a we wszystkich pozostałych segmentach bit ten jest ustawiony na 1. Jeśli zatem organizacja chce uniemożliwić zewnętrznym klientom inicjowanie połączeń z wewnętrznymi serwerami, wystarczy, że odfiltruje wszystkie segmenty przychodzące z wyzerowanym bitem ACK. Polityka ta blokuje wszystkie połączenia TCP inicjowane z zewnątrz, ale pozwala na inicjowanie ich od wewnątrz.

Reguły zapory są stosowane w routerach za pomocą list kontroli dostępu. Każdy interfejs routera ma odrębną listę tego typu. Przykładową listę kontroli dostępu dla firmowej sieci 222.22/16 przedstawia tabela 8.6. Ta lista dotyczy interfejsu łączącego router z zewnętrznym dostawcą ISP. Reguły są stosowane

po kolei (od góry do dołu) do każdego datagramu przechodzącego przez interfejs. Pierwsze dwie zasady umożliwiają użytkownikom wewnętrznym przeglądanie zasobów sieci WWW. Pierwsza reguła zezwala na opuszczanie sieci organizacji przez pakiety TCP o porcie docelowym 80. Druga zasada umożliwia wejście do sieci pakietom TCP o porcie źródłowym 80 i bicie ACK ustawionym na jeden. Zauważmy, że jeśli zewnętrzna jednostka spróbuje nawiązać połączenie TCP z wewnętrznym hostem, zapora je zablokuje, nawet jeśli źródłowy lub docelowy port to 80. Dwie następne reguły umożliwiają wchodzenie do firmowej sieci i wychodzenie z niej pakietom DNS. Podsumujmy te rozważania — ta dość restrykcyjna lista kontroli dostępu blokuje cały ruch oprócz ruchu związanego z siecią WWW zainicjowanego z wewnątrz firmy i ruchu DNS. W [CERT Filtering 2009] można znaleźć listę zalecanych filtrów portów i protokołów, które pozwalają uniknąć kilku dobrze znanych luk w zabezpieczeniach istniejących aplikacji sieciowych.

Operacja	Adres źródłowy	Adres docelowy	Protokół	Port źródłowy	Port docelowy	Bit flagi
Zezwól	222.22/16	Spoza 222.22/16	TCP	> 1023	80	Dowolny
Zezwól	Spoza 222.22/16	222.22/16	TCP	80	> 1023	ACK
Zezwól	222.22/16	Spoza 222.22/16	UDP	> 1023	53	-
Zezwól	Spoza 222.22/16	222.22/16	UDP	53	> 1023	-
Odrzuć	Wszystkie	Wszystkie	Wszystkie	Wszystkie	Wszystkie	Wszystkie

Tabela 8.6. Lista kontroli dostępu interfejsu routera

Stanowe filtry pakietów

W tradycyjnych filtrach pakietów decyzje są podejmowane dla każdego pakietu z osobna. Filtry stanowe śledzą połączenia TCP i podejmują decyzje na podstawie informacji na temat tych połączeń.

Aby zrozumieć działanie filtrów stanowych, ponownie zbadajmy listę kontroli dostępu z tabeli 8.6. Choć lista ta jest dość restrykcyjna, zezwala na przejście przez filtry dowolnym pakietom spoza sieci, mającym bit ACK = 1 i port równy 80. Napastnicy mogą wykorzystać takie pakiety do: próby wywołania awarii wewnętrznych systemów przez przesłanie błędnie zbudowanych pakietów, przeprowadzenia ataku DoS lub zmapowania wewnętrznej sieci. Naiwne rozwiązanie polega na zablokowaniu także pakietów TCP ACK, jednak uniemożliwi to wewnętrznym użytkownikom korzystanie z sieci WWW.

Filtry stanowe rozwiązują problem przez śledzenie wszystkich bieżących połączeń TCP w tabeli połączeń. Jest to możliwe, ponieważ zapora potrafi wykryć nawiązanie nowego połączenia przez obserwację 3-etapowego procesu negocjowania (pakiety SYN, SYNACK i ACK). Może też wykryć zakończenie połączenia, kiedy odbierze powiązany z nim pakiet FIN. Zapora może też konserwatywnie przyjąć, że połączenie zostało zakończone, jeśli jest nieaktywne przez na przykład 60 sekund. Przykładową tabelę połączeń zapory przedstawia tabela 8.7. Zapisane w niej dane informują, że obecnie aktywne są trzy połączenia TCP. Każde z nich zainicjowano wewnątrz organizacji. Ponadto listy kontroli dostępu filtrów stanowych mają nową kolumnę, „sprawdź połączenie", co ilustruje tabela 8.8. Warto zauważyć, że lista w tabeli 8.8 jest prawie identyczna jak lista w tabeli 8.6 i różni się tylko tym, że w dwóch regułach wskazuje na konieczność sprawdzenia połączenia.

Adres źródłowy	Adres docelowy	Port źródłowy	Port docelowy
222.22.1.7	37.96.87.123	12699	80
222.22.93.2	199.1.205.23	37654	80
222.22.65.143	203.77.240.43	48712	80

Tabela 8.7. Tabela połączeń filtru stanowego

Operacja	Adres źródłowy	Adres docelowy	Protokół	Port źródłowy	Port docelowy	Bit flagi	Sprawdź połączenie
Zezwól	222.22/16	Spoza 222.22/16	TCP	> 1023	80	Dowolny	
Zezwól	Spoza 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
Zezwól	222.22/16	Spoza 222.22/16	UDP	> 1023	53	-	
Zezwól	Spoza 222.22/16	222.22/16	UDP	53	> 1023	-	X
Odrzuć	Wszystkie	Wszystkie	Wszystkie	Wszystkie	Wszystkie	Wszystkie	

Tabela 8.8. Lista kontroli dostępu w filtrze stanowym

Prześledźmy kilka przykładów, aby zobaczyć, jak tabele połączeń i wzbogacone listy kontroli dostępu współdziałają ze sobą. Załóżmy, że napastnik próbuje wysłać błędnie zbudowany pakiet do sieci organizacji przez przesłanie datagramu o źródłowym porcie TCP 80 i z ustawionym bitem ACK. Przyjmijmy też,

że ten pakiet ma źródłowy numer portu 12543 i źródłowy adres IP 150.23.23.155. Kiedy ten pakiet dotrze do zapory, sprawdzi ona listę kontroli dostępu z tabeli 8.8, wedle której trzeba skontrolować tabelę połączeń przed wpuszczeniem pakietu do sieci organizacji. Zapora sprawdzi tę tabelę, wykryje, że pakiet nie jest przesyłany w ramach bieżącego połączenia TCP i odrzuci go. W drugim przykładzie przypuśćmy, że wewnętrzny użytkownik chce odwiedzić zewnętrzną witrynę. Ponieważ najpierw prześle segment TCP SYN, zapora zapisze połączenie TCP w tabeli połączeń. Kiedy serwer WWW odeśle pakiety (koniecznie z ustawionym bitem ACK), zapora sprawdzi zawartość tabeli i wykryje, że powiązane z nimi połączenie jest aktywne. Dlatego zapora przepuści pakiety i nie będzie zakłócać wewnętrznemu użytkownikowi przeglądania zasobów po sieci WWW.

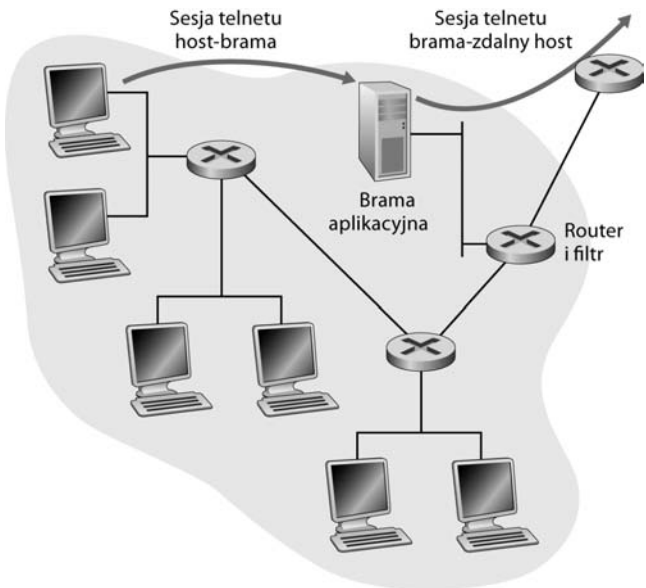
Brama aplikacyjna

Jak dowodzą powyższe przykłady, filtrowanie na poziomie pakietów pozwala organizacjom na ogólne badanie nagłówków IP oraz TCP/UDP, w tym adresów IP, numerów portów i bitów potwierdzenia. Co ma jednak zrobić organizacja, aby zaoferować usługę Telnetu tylko wybranym użytkownikom wewnętrznym (a nie adresom IP)? A jeśli organizacja chce, żeby ci uprzywilejowani użytkownicy uwierzytelniali się, zanim nawiążą połączenie z zewnętrznym serwerem Telnetu? Takie zadania przekraczają możliwości filtrów tradycyjnych i stanowych. Informacje o tożsamości wewnętrznych użytkowników nie są umieszczone w nagłówkach IP, TCP i UDP, ale w danych warstwy aplikacji.

Aby uzyskać ściślejszą kontrolę nad ruchem, zapory sieciowe muszą łączyć filtry pakietów z bramami aplikacyjnymi. Takie bramy podejmują decyzje na podstawie danych aplikacji, dlatego analizują nie tylko nagłówki IP, TCP i UDP. **Brama aplikacyjna** to serwer, przez który muszą przepływać wszystkie dane określonej aplikacji (przychodzące i wychodzące). W jednym komputerze może działać wiele bram aplikacyjnych, ale każda z nich jest oddzielnym serwerem z własnym procesem.

Aby zrozumieć działanie bram aplikacyjnych, zaprojektujmy zaporę, która zezwoli na nawiązywanie połączeń Telnetu ograniczonej grupie wewnętrznych użytkowników, a zapobiegnie nawiązywaniu jakichkolwiek połączeń z zewnątrz. Cel ten można osiągnąć za pomocą kombinacji filtra pakietów (w routerze) i bramy aplikacyjnej Telnetu (rysunek 8.35). Filtr routera jest skonfigurowany tak, aby blokował wszystkie połączenia Telnetu z wyjątkiem tych, które przychodzą spod adresu IP bramy aplikacyjnej. Taki filtr zmusza do nawiązywania wszystkich wychodzących połączeń Telnetu za pośrednictwem bramy aplikacyjnej. Rozważmy użytkownika wewnętrznego, który chce połączyć się z zewnętrznym serwerem Telnetu. Użytkownik musi najpierw nawiązać sesję Telnetu z bramą. Aplikacja działająca w bramie, która czeka na przychodzące połączenia Telnetu, pyta użytkownika o identyfikator i hasło. Kiedy użytkownik poda te

informacje, brama sprawdza, czy ma on prawo do łączenia się z zewnętrznymi serwerami. Jeśli nie, brama przerywa połączenie z użytkownikiem. Jeśli użytkownik ma odpowiednie uprawnienia, brama (1) pyta użytkownika o nazwę zewnętrznego hosta, z którym chce on nawiązać połączenie, (2) nawiązuje sesję Telnetu między sobą a zewnętrznym hostem i (3) przekazuje zewnętrznemu hostowi wszystkie dane przychodzące od użytkownika, a użytkownikowi — wszystkie dane przychodzące od zewnętrznego hosta. Zatem brama aplikacyjna Telnetu nie tylko przeprowadza autoryzację użytkownika, ale również pełni funkcję serwera i klienta Telnetu, przekazując informacje między użytkownikiem a zdalnym serwerem. Zwróćmy uwagę, że filtr zezwala na wykonanie drugiego etapu powyższej procedury, ponieważ to brama inicjuje połączenie Telnetu z zewnętrznym hostem.



Rysunek 8.35. Zapora składająca się z bramy aplikacyjnej i filtra

Sieci wewnętrzne często zawierają wiele bram aplikacyjnych, na przykład do obsługi Telnetu, HTTP, FTP i poczty. W rzeczywistości serwery poczty (podrozdział 2.4) i serwery buforujące są bramami aplikacyjnymi.

Bramy aplikacyjne nie są bez wad. Po pierwsze, każda aplikacja wymaga własnej bramy. Po drugie, obniża się wydajność, ponieważ wszystkie dane są przekazywane przez bramę. Problem ten jest szczególnie dotkliwy, kiedy wielu użytkowników lub aplikacji korzysta z tej samej bramy. Po trzecie, oprogramowanie klienckie musi „wiedzieć", jak skontaktować się z bramą aplikacyjną i jak poinformować bramę, z którym serwerem chce połączyć się użytkownik.

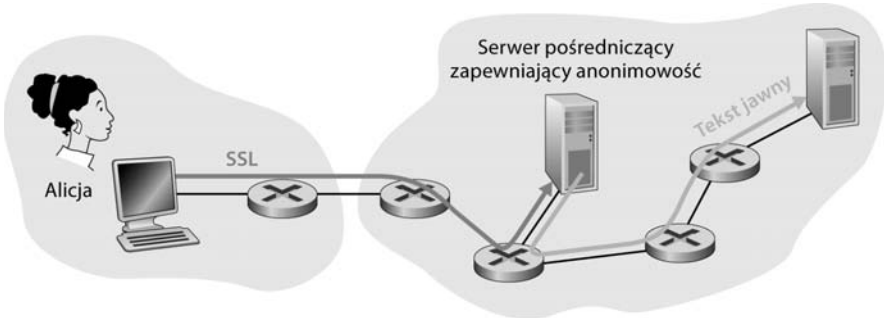
KĄCIK HISTORYCZNY

ANONIMOWOŚĆ I PRYWATNOŚĆ

Załóżmy, że Czytelnik zamierza odwiedzić kontrowersyjny serwis internetowy (prowadzony na przykład przez politycznego aktywistę), ale (1) nie chce ujawniać swojego adresu IP witrynie, (2) nie chce, aby lokalny dostawca ISP (dostarczający internet do domu lub biura) wiedział o odwiedzinach w tej witrynie i (3) nie chce, aby lokalny dostawca ISP miał wgląd w dane wymieniane z witryną. Przy tradycyjnym podejściu, czyli bezpośrednim połączeniu się z serwisem bez stosowania szyfrowania, żaden z tych warunków nie będzie spełniony. Nawet jeśli Czytelnik użyje protokołu SSL, dwa pierwsze wymogi zostaną naruszone — źródłowy adres IP będzie widoczny dla witryny w każdym wysyłanym datagramie, a lokalny dostawca ISP będzie mógł łatwo podejrzeć adres docelowy każdego pakietu.

Aby zapewnić sobie prywatność i anonimowość, można zastosować zaufany serwer pośredniczący i protokół SSL, jak ilustruje to rysunek 8.36. W tym podejściu najpierw należy nawiązać połączenie SSL z zaufanym serwerem pośredniczącym. Następnie trzeba przesłać tym połączeniem żądanie HTTP dotyczące pobrania strony z określonej witryny. Kiedy serwer pośredniczący otrzyma zaszyfrowane żądanie HTTP, odszyfruje je i przekaże jako tekst jawny do witryny. Następnie witryna prześle odpowiedź do serwera pośredniczącego, który przekaże ją do użytkownika przez połączenie SSL. Ponieważ witryna ma dostęp tylko do adresu IP serwera pośredniczącego, a nie do adresu klienta, można w ten sposób uzyskać anonimowy dostęp do witryny. Ponieważ cały ruch między użytkownikiem i serwerem pośredniczącym jest szyfrowany, lokalny dostawca ISP nie może naruszyć prywatności przez rejestrowanie odwiedzonych witryn lub wymienianych danych. Obecnie wiele firm (na przykład proxify.com) udostępnia usługi pośrednictwa tego typu.

Oczywiście w tym modelu serwer pośredniczący ma wszystkie informacje. Zna adres IP użytkownika i adres IP odwiedzanej witryny, a także ma dostęp do danych wymienianych jako tekst jawny między nadawcą i witryną. Dlatego to rozwiązanie jest tylko tak dobre, jak wiarygodny jest serwer. Bardziej niezawodne podejście, stosowane w usłudze TOR (zapewnia ona anonimowość i prywatność), polega na kierowaniu ruchu przez kilka współdziałających serwerów pośredniczących [TOR 2009]. TOR umożliwia niezależnym jednostkom dodawanie serwerów pośredniczących do ich puli. Kiedy użytkownik połączy się z serwerem pośredniczącym obsługującym TOR, usługa ta losowo wybierze z puli łańcuch trzech serwerów pośredniczących i skieruje cały ruch między klientem i serwerem do tego łańcucha. W ten sposób (przy założeniu, że właściciele serwerów nie współpracują ze sobą) nikt nie wie, iż miała miejsce komunikacja między danym adresem IP i docelową witryną. Ponadto choć między ostatnim serwerem pośredniczącym i serwerem WWW przesyłany jest tekst jawny, ostatni serwer pośredniczący nie zna adresu IP hosta wysyłającego i odbierającego te dane.



Rysunek 8.36. Zapewnianie anonimowości i prywatności za pomocą serwera pośredniczącego

8.8.2. Systemy wykrywania włamań

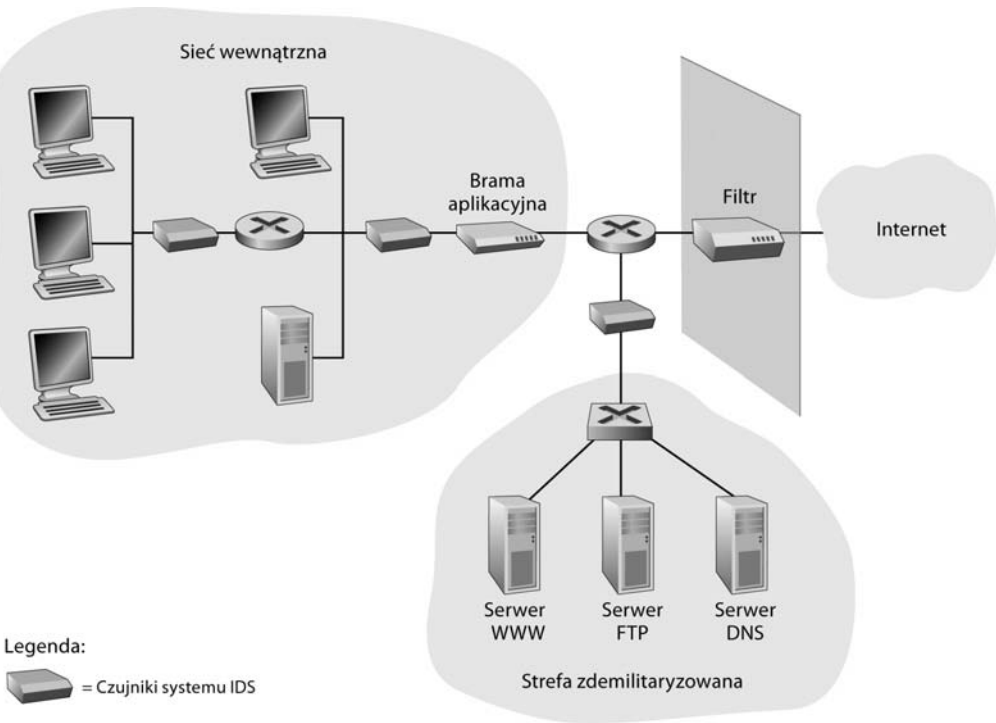
Właśnie pokazaliśmy, że filtry pakietów (tradycyjne i stanowe) badają pola nagłówków IP, TCP, UDP i ICMP przy określaniu, które pakiety należy przepuścić przez zaporę. Jednak do wykrycia ataków wielu typów potrzebna jest **pogłębiona analiza pakietów**, czyli zbadanie nie tylko pół nagłówka, ale też danych aplikacji przenoszonych w pakiecie. W punkcie 8.8.1 wyjaśniliśmy, że bramy aplikacyjne często przeprowadzają pogłębioną analizę pakietów. Jednak bramy robią to tylko dla określonej aplikacji.

Widać więc, że potrzebne jest jeszcze jedno urządzenie. Powinno ono nie tylko badać nagłówki wszystkich przesyłanych pakietów (jak robią to filtry), ale też — w odróżnieniu od filtrów — przeprowadzać pogłębioną analizę pakietów. Kiedy takie urządzenie wykryje podejrzany pakiet lub ich serię, może zablokować danym dostęp do sieci organizacji. Z uwagi na to, że pakiety są tylko podejrzane, narzędzie może też je przepuścić, a jednocześnie zaalarmować administratora sieci, który następnie przyjrzy się dokładniej danym i podejmie odpowiednie działania. Urządzenie ogłaszające alarm po wykryciu potencjalnie szkodliwych danych to **system wykrywania włamań** (ang. *Intrusion Detection System* — IDS). Narzędzie filtrujące podejrzane pakiety to **system zapobiegania włamaniom** (ang. *Intrusion Prevention System* — IPS). W tym punkcie omawiamy razem oba rodzaje systemów (IDS i IPS), ponieważ ich najciekawsze techniczne aspekty związane są z wykrywaniem podejrzanego ruchu, a nie z tym, czy narzędzie ogłasza alarm lub odrzuca pakiety. Od tego miejsca będziemy określać systemy IDS i IPS wspólną nazwą IDS.

System IDS może służyć do wykrywania różnorodnych ataków — mapowania sieci (przeprowadzonego na przykład za pomocą narzędzia nmap), skanowania portów, skanowania stosu TCP, ataków DoS przez wykorzystanie całej dostępnej przepustowości, robaków i wirusów, ataków z wykorzystaniem luk systemu operacyjnego i ataków z wykorzystaniem luk aplikacji (przegląd ataków sieciowych przedstawiliśmy w podrozdziale 1.6). Obecnie systemy IDS są

stosowane przez tysiące organizacji. Wiele z tych systemów to rozwiązania zastrzeżone (oferują je na przykład firmy Cisco, Check Point i inni producenci sprzętu zabezpieczającego). Jednak liczne z zainstalowanych systemów IDS (na przykład omówiony dalej system Snort) to narzędzia bezpłatne.

Organizacja może zainstalować w sieci jeden lub kilka czujników systemu IDS. Rysunek 8.37 przedstawia sieć firmy korzystającej z trzech takich sensorów. Jeśli zainstalowanych jest kilka czujników, zwykle współdziałają one ze sobą i przesyłają informacje o podejrzanym ruchu do centralnego procesora systemu IDS, który zbiera i integruje dane oraz, kiedy uzna to za właściwe, alarmuje administratorów sieci. Sieć z rysunku 8.37 jest podzielona na dwa obszary. Pierwszy z nich to mocno zabezpieczona część chroniona za pomocą filtra pakietów i bramy aplikacyjnej oraz monitorowana przy użyciu czujników systemu IDS. Drugi obszar to część ze słabszymi zabezpieczeniami — tak zwana **strefa zdemilitaryzowana**. Jest ona chroniona tylko przez filtr pakietów, a ponadto monitorowana za pomocą czujników IDS. Warto zauważyć, że w strefie zdemilitaryzowanej znajdują się serwery potrzebujące możliwości komunikowania się ze światem zewnętrznym (na przykład publiczny serwer WWW i autorytarny serwer DNS).



Rysunek 8.37. Sieć organizacji stosującej filtr, bramę aplikacyjną i czujniki systemu IDS

Czytelnicy mogą się w tym momencie zastanawiać, po co instalować wiele czujników systemu IDS. Dlaczego w sieci z rysunku 8.37 nie wystarczy umieścić jednego sensora bezpośrednio za filtrem pakietów (a nawet zintegrować czujnik z tym urządzeniem)? Wkrótce pokażemy, że system IDS musi nie tylko przeprowadzać pogłębioną analizę pakietów, ale też porównywać każdy przekazywany pakiet z dziesiątkami tysięcy sygnatur. Może to wymagać dużej mocy obliczeniowej, zwłaszcza jeśli firma odbiera z internetu wiele gigabitów danych na sekundę. Umieszczenie czujników systemu IDS bliżej stacji odbiorczych powoduje, że każdy sensor obsługuje tylko część danych, co ułatwia ich sprawne przetwarzanie. Jednak obecnie dostępne są wydajne systemy IDS i IPS, dlatego wielu firmom wystarcza jeden czujnik zlokalizowany blisko routera dostępowego.

Systemy IDS można ogólnie podzielić na **systemy oparte na sygnaturach** i **systemy oparte na anomaliach**. Te pierwsze przechowują rozbudowane bazy danych sygnatur specyficznych dla ataków. Każda sygnatura to zestaw reguł powiązanych z włamaniami. Sygnatura może być po prostu listą cech pojedynczego pakietu (może obejmować na przykład numery portów źródłowego i docelowego, typ protokołu i określony łańcuch bitów w treści pakietu) lub dotyczyć serii pakietów. Za tworzenie sygnatur odpowiadają zwykle doświadczeni inżynierowie zabezpieczeń sieci badający znane ataki. Administrator firmowej sieci może zmodyfikować sygnatury lub dodać do bazy ich nowe egzemplarze.

Pod względem operacyjnym systemy IDS oparte na sygnaturach odczytują każdy przekazywany pakiet i porównują go z sygnaturami z bazy danych. Jeśli pakiet (lub ich seria) pasuje do sygnatury z bazy, system IDS ogłasza alarm. Może on zostać wysłany w wiadomości e-mail do administratora sieci, przekazany do systemu zarządzania siecią lub po prostu zarejestrowany w celu późniejszej analizy.

Systemy IDS oparte na sygnaturach — choć są powszechnie stosowane — mają wiele ograniczeń. Najważniejsze jest to, że przygotowanie odpowiedniej sygnatury wymaga wiedzy na temat ataku. Oznacza to, że takie systemy są zupełnie nieodporne na nowe ataki, które dopiero trzeba zarejestrować. Inna wada polega na tym, że dopasowanie sygnatury do pakietu nie musi wcale oznaczać ataku, dlatego system będzie zgłaszał fałszywe alarmy. Ponadto z uwagi na to, że każdy pakiet trzeba porównać z długą listą sygnatur, system IDS może szybko zostać przeciążony przetwarzaniem i nie wykryć wielu szkodliwych pakietów.

Systemy IDS oparte na anomaliach tworzą profil ruchu w czasie jego obserwacji w normalnych warunkach. Następnie system szuka statystycznie nietypowych strumieni pakietów, na przykład niezwykłego wzrostu odsetka pakietów ICMP lub nagłego wykładniczego wzrostu liczby operacji skanowania portów i skanowania ping sweep. Doskonałą cechą systemów IDS opartych na anomaliach jest to, że nie wymagają uprzedniej wiedzy na temat znanych ataków. Pozwala to wykryć nowe, nieudokumentowane ataki. Z drugiej strony niezwykle trudne jest odróżnienie zwykłego ruchu od statystycznie nietypowych

danych. Obecnie większość zainstalowanych systemów IDS jest oparta na sygnaturach, choć niektóre rozwiązania mają wybrane funkcje systemów opartych na anomaliach.

Snort

Snort to bezpłatny system IDS o otwartym dostępie do kodu źródłowego zainstalowany w setkach tysięcy komputerów [Snort 2007; Koziol 2003]. Można go uruchomić w systemach operacyjnych Linux, UNIX i Windows. W narzędziu tym wykorzystano uniwersalny interfejs do analizy ruchu, libpcap, stosowany również w aplikacji Wireshark i innych programach do analizy pakietów. System Snort sprawnie obsługuje dane przesyłane z szybkością 100 Mb/s. W sieciach, w których ruch ma natężenie gigabitu na sekundę lub większe, potrzebnych może być kilka czujników.

Aby lepiej zrozumieć system Snort, przyjrzyjmy się przykładowej sygnaturze:

```
alert ixmp $EXTERNAL_NET any -> $HOME_NET any
(msg:"ICMP PING NMAP"; dsize: 0; itype: 8;)
```

Ta sygnatura pasuje do każdego pakietu ICMP, który jest przesyłany do sieci firmy (`$HOME_NET`) spoza niej (`$EXTERNAL_NET`), ma typ równy 8 (komunikat ICMP ping) i nie ma treści (`dsize = 0`). Ponieważ pakiety ping o tych cechach generuje narzędzie nmap (zobacz podrozdział 1.6), omawiana sygnatura służy do wykrywania skanowania ping sweep przeprowadzanego za pomocą tego narzędzia. Jeśli pakiet pasuje do tej sygnatury, system Snort ogłosi alarm obejmujący komunikat `ICMP PING NMAP`.

Prawdopodobnie najbardziej zdumiewającym aspektem związanym z systemem Snort jest duża społeczność użytkowników i ekspertów do spraw bezpieczeństwa tworzących bazę sygnatur. Zwykle w przeciągu kilku godzin od wykrycia nowego ataku członkowie społeczności piszą i udostępniają odpowiednią sygnaturę pobieraną następnie przez setki tysięcy instalacji systemu Snort rozproszonych po całym świecie. Ponadto administratorzy sieci mogą za pomocą składni sygnatur systemu Snort dostosowywać je do potrzeb organizacji poprzez modyfikację istniejących sygnatur lub tworzenie nowych.

8.9. Podsumowanie

W tym rozdziale zbadaliśmy różne mechanizmy, których nasi kochankowie, Bartek i Alicja, mogą użyć do bezpiecznej komunikacji. Stwierdziliśmy, że Bartek i Alicja są zainteresowani poufnością (aby tylko oni mogli zrozumieć treść przesłanej wiadomości), uwierzytelnianiem (aby byli pewni, że porozumiewają się ze sobą) i ochroną integralności (aby byli pewni, że ich wiadomości nie zostały zmodyfikowane podczas transmisji). Oczywiście bezpieczna komunikacja nie

jest wyłączną domeną sekretnych kochanków. Jak napisaliśmy w podrozdziałach od 8.4 do 8.7, zabezpieczenia są potrzebne we wszystkich warstwach architektury sieciowej, ponieważ napastnicy dysponują bardzo zróżnicowanym arsenałem ataków.

W pierwszej części rozdziału omówiliśmy zasady bezpiecznej komunikacji. W podrozdziale 8.2 opisaliśmy techniki szyfrowania i odszyfrowywania danych, w tym kryptografię z kluczem symetrycznym oraz kryptografię z kluczem publicznym. Te dwie ogólne klasy technik szyfrowania zbadaliśmy na przykładzie algorytmów DSA i RSA, które są powszechnie używane we współczesnych sieciach.

W podrozdziale 8.3 zbadaliśmy dwa modele zapewniania integralności wiadomości — kody uwierzytelniające (MAC) i podpisy cyfrowe. Te dwa podejścia mają wiele cech wspólnych. W obu stosowane są kryptograficzne funkcje skrótu i obie umożliwiają zweryfikowanie źródła wiadomości, a także sprawdzenie integralności samego komunikatu. Ważna różnica między nimi związana jest z tym, że przy stosowaniu kodów MAC szyfrowanie nie jest potrzebne, natomiast podpisy cyfrowe wymagają infrastruktury do obsługi kluczy publicznych. Obie techniki — jak pokazaliśmy w podrozdziałach od 8.4 do 8.7 — są powszechnie używane w praktyce. Ponadto podpisy cyfrowe służą do tworzenia certyfikatów cyfrowych istotnych przy weryfikowaniu poprawności kluczy publicznych. Zbadaliśmy też uwierzytelnianie punktów końcowych i wyjaśniliśmy, jak można wykorzystać identyfikatory jednorazowe do zapobiegania atakom przez odtwarzanie.

W podrozdziałach od 8.4 do 8.7 przeanalizowaliśmy kilka powszechnie stosowanych bezpiecznych protokołów sieciowych. Pokazaliśmy, że podstawą protokołów PGP, SSL, IPsec i bezpiecznej komunikacji bezprzewodowej jest kryptografia z kluczem symetrycznym. Wyjaśniliśmy, że kryptografia z kluczem publicznym jest niezbędna w protokołach PGP i SSL. Ponadto Czytelnicy dowiedzieli się, że protokół PGP zapewnia integralność wiadomości za pomocą podpisów cyfrowych, natomiast protokoły SSL i IPsec wykorzystują do tego kody MAC. Po zrozumieniu podstawowych zasad kryptografii i przestudiowaniu ich zastosowań Czytelnicy są gotowi do projektowania własnych bezpiecznych protokołów sieciowych!

Uzbrojeni w techniki opisane w podrozdziałach od 8.2 do 8.7 Alicja i Bartek mogą bezpiecznie się porozumiewać (miejmy nadzieję, że studiują łączność sieciową i dobrze nauczyli się tej partii materiału, bo w przeciwnym razie Inga łatwo odkryje ich romans!). Okazało się jednak, że poufność jest tylko jednym z elementów bezpieczeństwa sieci. Jak wyjaśniliśmy w podrozdziale 8.8, coraz większy nacisk kładzie się na zabezpieczenie infrastruktury sieciowej przed zakusami potencjalnych napastników. W drugiej części rozdziału omówiliśmy więc zapory sieciowe i systemy IDS, analizujące pakiety przychodzące do sieci firmy i wychodzące z niej.

W tym rozdziale omówiliśmy wiele materiału, przy czym skoncentrowaliśmy się na najważniejszych aspektach zabezpieczeń we współczesnych sieciach.

Czytelników chcących lepiej poznać tę dziedzinę zachęcamy do zapoznania się ze źródłami wymienionymi w tym rozdziale. Szczególnie polecamy następujące pozycje: [Skoudis 2006] (ataki i bezpieczeństwo operacyjne), [Kaufman 1995] (kryptografia i jej zastosowania w zabezpieczaniu sieci), [Rescorla 2001] (szczegółowe, ale przystępne omówienie protokołu SSL) i [Edney 2003] (wyczerpujący przegląd zabezpieczeń sieci 802.11, w tym wnikliwa analiza protokołu WEP i jego wad). Czytelnicy mogą też zapoznać się ze źródłem [Ross 2009]. Jest to bogaty zestaw ponad 400 slajdów z programu PowerPoint dotyczących zabezpieczania sieci (obejmują one też kilka ćwiczeń związany z systemem Linux).

Pytania i problemy do rozwiązania

Rozdział 8. Pytania kontrolne

PODROZDZIAŁ 8.1

1. Jakie są różnice między poufnością a integralnością wiadomości? Czy można mieć jedno bez drugiego? Uzasadnij odpowiedź.
2. Jednostki działające w internecie (routery, przełączniki, serwery DNS, serwery WWW, systemy końcowe itd.) często muszą się komunikować w bezpieczny sposób. Podaj trzy przykładowe pary takich jednostek, które mogą potrzebować bezpiecznej komunikacji.

PODROZDZIAŁ 8.2

3. Jaka jest ważna różnica między systemem z kluczami symetrycznymi a systemem z kluczami publicznymi?
4. Przypuśćmy, że intruz ma zaszyfrowaną wiadomość oraz jej odszyfrowaną wersję. Czy intruz może przeprowadzić atak ze znanym szyfrogramem, atak ze znanym tekstem jawnym, czy też atak z wybranym tekstem jawnym?
5. Rozważmy 8-bitowy szyfr blokowy. Ile możliwych bloków wejściowych ma ten szyfr? Ile odwzorowań w nim występuje? Jeśli potraktujemy każde odwzorowanie jako klucz, to ile kluczy pozwala wygenerować ten szyfr?
6. Przypuśćmy, że N osób chce porozumiewać się z N–1 pozostałych osób, używając kryptografii z kluczem symetrycznym. Cała komunikacja między dowolnymi dwiema osobami i oraz j jest widoczna dla wszystkich pozostałych N osób, a żadna inna osoba nie powinna mieć możliwości odkodowania tej komunikacji. Ile kluczy jest potrzebnych w całym systemie? Załóżmy teraz, że używana jest kryptografia z kluczem publicznym. Ile kluczy jest potrzebnych w tym przypadku?
7. Załóżmy, że $n = 10000$, $a = 10023$ i $b = 10004$. Wykorzystaj tożsamość z arytmetyki modularnej do rozwiązania w pamięci równania $(a \cdot b) \bmod n$.

8. Przyjmijmy, że chcesz zaszyfrować wiadomość 10101111 przez zakodowanie odpowiadającej jej liczby dziesiętnej. Podaj tę liczbę.

PODROZDZIAŁ 8.3

9. Dlaczego skrót wiadomości zapewnia lepszą ochronę integralności niż suma kontrolna (na przykład internetowa suma kontrolna)?
10. Czy potrafisz „odszyfrować" skrót wiadomości, aby uzyskać jej pierwotną postać? Wyjaśnij odpowiedź.
11. Rozważmy odmianę algorytmu do obliczania kodu MAC (rysunek 8.9), w której nadawca wysyła parę $(m, H(m)+s)$, gdzie $H(m)+s$ to złączenie $H(m)$ i s. Czy ta wersja algorytmu ma lukę? Dlaczego?
12. Co to znaczy, że dokument podpisany cyfrowo jest weryfikowalny i niepodrabialny?
13. Dlaczego skrót wiadomości zaszyfrowany kluczem publicznym jest lepszym podpisem cyfrowym niż cała wiadomość zaszyfrowana kluczem publicznym?
14. Załóżmy, że firma certifier.com tworzy certyfikat dla organizacji foo.com. Zwykle cały certyfikat jest szyfrowany za pomocą klucza publicznego firmy certifier.com. Prawda czy fałsz?
15. Załóżmy, że Alicja wysyła wiadomość do każdego, kto o nią poprosi. Tysiące osób zamierza pobrać ten komunikat, przy czym każda z nich chce mieć pewność co do integralności wiadomości. Czy w tym kontekście lepszy będzie system sprawdzania integralności na podstawie kodów MAC, czy sygnatur cyfrowych? Dlaczego?
16. Do czego służy identyfikator jednorazowy w protokole uwierzytelniania?
17. Co oznacza powiedzenie, że identyfikator jednorazowy jest wartością wykorzystywaną „raz w życiu"? W czyim „życiu"?
18. Na czym polega atak typu „człowiek w środku"? Czy można przeprowadzić ten atak, jeśli używane są klucze symetryczne?

PODROZDZIAŁY 8.4 – 8.7

19. Załóżmy, że Bartek otrzymuje komunikat PGP od Alicji. Skąd Bartek wie, że to Alicja utworzyła wiadomość (a nie na przykład Inga)? Czy w protokole PGP do sprawdzania integralności komunikatów używane są kody MAC?
20. Prawda czy fałsz: w rekordzie SSL znajduje się pole z numerem sekwencyjnym protokołu SSL.
21. Do czego służą losowe identyfikatory jednorazowe w procesie negocjowania w protokole SSL?
22. Załóżmy, że w sesji SSL stosowany jest szyfr blokowy z wiązaniem bloków. Prawda czy fałsz: serwer wysyła do klienta wektor IV jako tekst jawny.

23. Przypuśćmy, że Bartek inicjuje połączenie TCP z podszywającą się pod Alicję Ingą. W czasie negocjowania Inga wysyła do Bartka certyfikat Alicji. W której fazie działania algorytmu negocjowania protokołu SSL Bartek odkryje, że nie komunikuje się z Alicją?
24. Rozważmy wysyłanie strumienia pakietów z hosta A do hosta B za pomocą protokołu IPsec. Prawda czy fałsz: zwykle dla każdego pakietu w strumieniu tworzony jest nowy kanał SA.
25. Załóżmy, że protokół TCP korzysta z kanału IPsec między centralą i oddziałem z rysunku 8.29. Prawda czy fałsz: jeśli protokół TCP przeprowadzi retransmisję pakietu, dwa odpowiadające sobie pakiety wysyłane przez router R1 będą miały ten sam numer sekwencyjny w nagłówku ESP.
26. Prawda czy fałsz: kanały IKE SA i IPsec SA są identyczne.
27. Rozważmy protokół WEP w sieciach 802.11. Załóżmy, że dane to 10101100, a strumień szyfrujący to 1111000. Podaj wynikowy szyfrogram.
28. Prawda czy fałsz: w protokole WEP wektor IV jest wysyłany w każdej ramce jako tekst jawny.

PODROZDZIAŁ 8.8

29. Filtry stanowe przechowują dwie struktury danych. Nazwij je i krótko opisz, do czego służą.
30. Rozważmy tradycyjny (bezstanowy) filtr pakietów. Prawda czy fałsz: takie urządzenie może filtrować pakiety na podstawie bitów flag protokołu TCP, a także innych pól nagłówka.
31. Prawda czy fałsz: w tradycyjnych filtrach pakietów każdy interfejs ma odrębną listę kontroli dostępu.
32. Dlaczego brama aplikacyjna musi współdziałać z filtrem routera, aby była skuteczna?
33. Prawda czy fałsz: systemy IDS i IPS oparte na sygnaturach analizują treść segmentów TCP i UDP.

Problemy

1. Za pomocą szyfru monoalfabetycznego z rysunku 8.3 zakoduj wiadomość „`To jest łatwy problem`”. Zdekoduj wiadomość „`aicćw owcwów, fćwóaw?`”.
2. Udowodnij, że atak ze znanym tekstem jawnym, w którym Inga zna pary translacji (tekst zaszyfrowany, tekst jawny) dla siedmiu liter, w przykładzie z podrozdziału 8.2.1 zmniejsza liczbę kombinacji do sprawdzenia o około 10^9.

3. Rozważ system polialfabetyczny pokazany na rysunku 8.4. Czy atak z wybranym tekstem jawnym, w którym udałoby się uzyskać szyfrogram tekstu „`Pchnąć w tę łódź jeża lub ośm skrzyń fig`". wystarczyłby do odszyfrowania wszystkich wiadomości? Dlaczego (albo dlaczego nie)?
4. Rozważmy szyfr blokowy z rysunku 8.5. Załóżmy, że każdy szyfr blokowy T_i po prostu odwraca kolejność ośmiu bitów wejściowych (na przykład zamiast 11110000 otrzymujemy 00001111). Przyjmijmy też, że 64-bitowy szyfrator nie modyfikuje żadnych bitów (wartość wyjściowa m-tego bitu jest równa jego wartości wejściowej). (a) Jaka będzie wartość wyjściowa dla $n = 3$ i 64-bitowych danych wejściowych w postaci ośmiu powtórzeń sekwencji 10100000? (b) Odpowiedz na pytanie z punktu (a) przy założeniu, że ostatni bit 64-bitowych danych wejściowych zamieniono z 0 na 1. (c) Ponownie rozwiąż zadania z punktów (a) i (b), jednak tym razem przyjmij, że 64-bitowy szyfrator odwraca kolejność 64 bitów.
5. Rozważmy szyfr blokowy z rysunku 8.5. Dla danego „klucza" Alicja i Bartek muszą przechowywać osiem tabel o wielkości osiem na osiem bitów. Ile bitów pamięci potrzebuje Alicja (lub Bartek) na przechowywanie wszystkich ośmiu tabel? Jak ma się to do liczby bitów potrzebnych do przechowywania pełnej tabeli 64-bitowego szyfru blokowego?
6. Zastanówmy się nad 3-bitowym szyfrem blokowym z tabeli 8.1. Załóżmy, że tekst jawny to 100100100. (a) Najpierw przyjmijmy, że wiązanie bloków nie jest stosowane. Podaj wyjściowy szyfrogram. (b) Przypuśćmy, że Inga przechwyciła szyfrogram. Co Inga może wykryć, jeśli wie, że wykorzystano 3-bitowy szyfr blokowy bez wiązania bloków (nie zna jednak konkretnego szyfru)? (c) Teraz załóżmy, że zastosowano wiązanie bloków z wektorem IV = 111. Podaj wyjściowy szyfrogram.
7. (a) Użyj algorytmu RSA i wartości $p = 3$ oraz $q = 11$ do zakodowania poszczególnych liter słowa „`kot`". Zastosuj algorytm deszyfrowania, aby odtworzyć pierwotny tekst jawny. (b) Wykonaj zadanie z punktu (a), jednak tym razem zaszyfruj słowo „kot" jako jedną wiadomość m.
8. Rozważmy algorytm RSA o wartościach $p = 5$ i $q = 11$.
   a. Podaj wartości n i z.
   b. Niech e wynosi 3. Dlaczego jest to rozsądna wartość?
   c. Znajdź d takie, aby $de = 1 \pmod z$ i $d < 160$.
   d. Zaszyfruj wiadomość $m = 8$ za pomocą klucza (n, e). Niech c oznacza uzyskany szyfrogram. Przedstaw obliczenia. *Wskazówka:* aby je uprościć, wykorzystaj poniższą zależność:

$$[(a \bmod n)\cdot(b \bmod n)] \bmod n = (a\cdot b) \bmod n$$

9. W tym problemie rozważymy algorytm szyfrowania z kluczem publicznym Diffiego-Hellmana (DH). Umożliwia on dwóm jednostkom uzgodnienie

wspólnego klucza. Algorytm DH wykorzystuje dużą liczbę pierwszą p i inną dużą liczbę g mniejszą od p. Liczby p i g są publicznie dostępne (dlatego napastnik może je poznać). W algorytmie DH Alicja i Bartek niezależnie wybierają niejawne klucze S_A i S_B. Alicja oblicza następnie klucz publiczny T_A przez podniesienie g do potęgi S_A i wykonanie dzielenia modulo p. Bartek w podobny sposób oblicza klucz publiczny T_B — podnosi g do potęgi S_B i wykonuje dzielenie modulo p. Alicja i Bartek wymieniają klucze publiczne przez internet. Alicja oblicza następnie wspólny niejawny klucz S przez podniesienie wartości T_B do potęgi S_A i wykonanie dzielenia modulo p. Bartek w podobny sposób oblicza wspólny klucz S' — podnosi wartości T_A do potęgi S_B i wykonuje dzielenia modulo p.

a. Udowodnij, że Alicja i Bartek uzyskają ten sam klucz symetryczny (czyli że $S = S'$).

b. Załóżmy, że $p = 11$ i $g = 2$, a Alicja i Bartek wybrali klucze prywatne $S_A = 5$ i $S_B = 12$. Ustal klucze publiczne Alicji i Bartka (T_A i T_B). Przedstaw wszystkie obliczenia.

c. Kontynuując punkt (b), określ wspólny klucz symetryczny S. Przedstaw wszystkie obliczenia.

d. Przedstaw diagram czasowy pokazujący, że algorytm DH jest podatny na atak metodą „człowiek w środku". Schemat powinien mieć trzy poziome linie — po jednej dla Alicji, Bartka i Ingi (napastnika).

10. Przypuśćmy, że Alicja chce komunikować się z Bartkiem za pomocą szyfru z kluczem symetrycznym i klucza sesji K_S. W podrozdziale 8.2 wyjaśniliśmy, jak wykorzystać szyfr z kluczem publicznym do przesłania klucza sesji od Alicji do Bartka. W tym problemie zbadamy możliwość dystrybucji klucza sesji za pomocą centrum dystrybucji kluczy (bez korzystania z szyfrów z kluczem publicznym). Takie centrum to serwer mający wspólny niepowtarzalny niejawny klucz symetryczny z każdym zarejestrowanym użytkownikiem. Klucze należące do Alicji i Bartka oznaczmy jako $K_{A\text{-}CDK}$ i $K_{B\text{-}CDK}$. Zaprojektuj system, który korzysta z centrum dystrybucji kluczy do przekazywania klucza K_S od Alicji do Bartka. Dystrybucja klucza sesji w tym systemie powinna wymagać przesłania trzech komunikatów: od Alicji do centrum dystrybucji, z centrum dystrybucji do Alicji i od Alicji do Bartka. Pierwsza wiadomość to $K_{A\text{-}CDK}(A, B)$. Przy użyciu oznaczeń $K_{A\text{-}CDK}$, $K_{B\text{-}CDK}$, S, A i B odpowiedz na następujące pytania:

a. Jak będzie wyglądać druga wiadomość?

b. Jak będzie wyglądać trzecia wiadomość?

11. Skonstruuj trzecią wiadomość, inną niż dwie pokazane na rysunku 8.8, która będzie miała tę samą sumę kontrolną.

12. Załóżmy, że Alicja i Bartek mają dwa wspólne niejawne klucze — klucz uwierzytelniający S_1 i symetryczny klucz szyfrujący S_2. Uzupełnij rysunek 8.9 w taki sposób, aby zapewnić integralność i poufność.
13. W protokole dystrybucji plików BitTorrent (zobacz rozdział 2.) serwer dzieli plik na bloki, a następnie węzły rozsyłają te bloki między sobą. Jeśli nie zostaną zastosowane żadne zabezpieczenia, napastnik będzie mógł łatwo wyrządzić szkody w grupie hostów przez podanie się za niegroźny węzeł i przesłanie sfałszowanych bloków małej podgrupie hostów. Następnie te nieświadome zagrożenia hosty prześlą sfałszowane bloki kolejnym węzłom. Dlatego w protokole BitTorrent niezbędny jest mechanizm umożliwiający węzłom zweryfikowanie integralności bloku, co zapobiegnie dalszemu przesyłaniu sfałszowanych danych. Załóżmy, że kiedy węzeł dołącza do grupy, otrzymuje plik *.torrent* z *w pełni* zaufanego źródła. Opisz prosty system, który umożliwi węzłom weryfikowanie integralności bloków.
14. W protokole routingu OSPF integralność komunikatów jest zapewniana za pomocą kodów MAC, a nie podpisów cyfrowych. Jak sądzisz, dlaczego wybrano to rozwiązanie?
15. Rozważmy protokół uwierzytelniania z rysunku 8.16, w którym Alicja uwierzytelnia się wobec Bartka i który działa poprawnie (tzn. nie znaleźliśmy w nim żadnych usterek). Przypuśćmy teraz, że kiedy Alicja uwierzytelnia się wobec Bartka, Bartek musi uwierzytelnić się wobec Alicji. Podaj scenariusz, w którym Inga, udając Alicję, może uwierzytelnić się wobec Bartka jako Alicja (*wskazówka:* zauważ, że sekwencje operacji w tym protokole — taka, w której inicjatorem jest Inga, i taka, w której inicjatorem jest Bartek — mogą być dowolnie przeplecione. Szczególną uwagę zwróć na fakt, że zarówno Bartek, jak i Alicja używają identyfikatora jednorazowego, a jeśli nie zachowują należytej ostrożności, ten sam identyfikator może zostać wykorzystany w złych intencjach).
16. W ataku typu „człowiek w środku" pokazanym na rysunku 8.19 Alicja nie uwierzytelniła się wobec Bartka. Gdyby zrobiła to za pomocą protokołu uwierzytelniania za pomocą klucza publicznego, czy udałoby się uniknąć ataku? Wyjaśnij swoje rozumowanie.
17. Na rysunku 8.20 pokazano operacje, które musi wykonać Alicja w czasie korzystania z protokołu PGP, aby zapewnić poufność, uwierzytelnianie i integralność. Narysuj podobny diagram operacji, które musi wykonać Bartek na „paczce" otrzymanej od Alicji.
18. Załóżmy, że Alicja chce wysłać e-mail do Bartka. Bartek ma klucze publiczny i prywatny (K_B^+, K_B^-), a Alicja posiada certyfikat Bartka, natomiast nie ma kluczy publicznego i prywatnego. Alicja i Bartek (podobnie jak cały świat) korzystają z tej samej funkcji skrótu H(·).

c. Czy przy tych założeniach można zaprojektować system, który umożliwi Bartkowi zweryfikowanie, że to Alicja utworzyła wiadomość? Jeśli tak, narysuj ten system w formie schematu blokowego z postaciami Alicji i Bartka.

d. Czy przy tych założeniach można zaprojektować system, który zapewni poufność wiadomości wysyłanej od Alicja do Bartka? Jeśli tak, narysuj ten system w formie schematu blokowego z postaciami Alicji i Bartka.

19. Rozważmy poniższe dane wyjściowe programu Wireshark. Ilustrują one fragment sesji SSL.

a. Czy pakiet 112 widoczny w programie Wireshark został wysłany przez klienta, czy przez serwer?

b. Jaki jest adres IP i numer portu serwera?

c. Załóżmy, że nie występuje utrata danych ani ich retransmisja. Jaki będzie numer sekwencyjny następnego segmentu TCP wysłanego przez klienta?

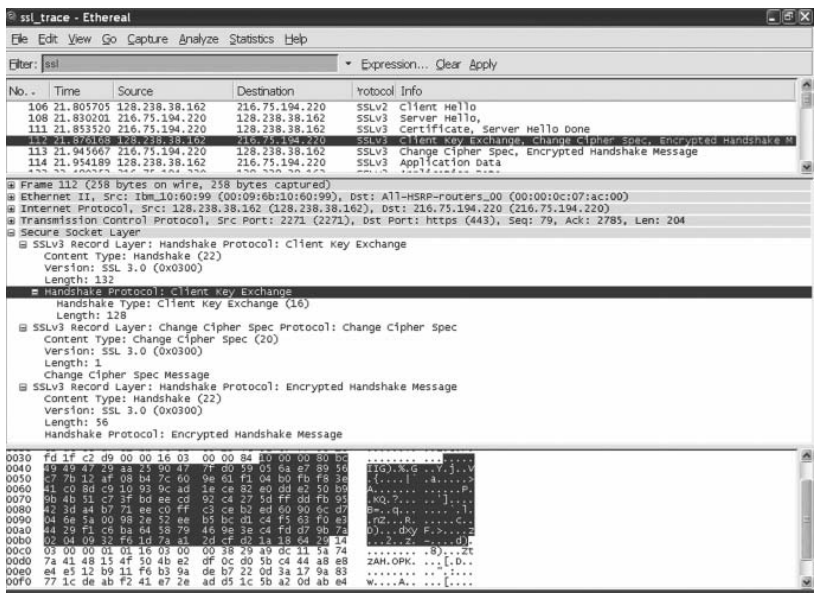
d. Ile rekordów SSL obejmuje pakiet 112 widoczny w programie Wireshark?

e. Czy pakiet 112 obejmuje klucz MS lub klucz EMS?

f. Przyjmijmy, że pole typu używane do negocjowania ma jeden bajt, a każde pole długości ma trzy bajty. Jakie są wartości pierwszego i ostatniego bajta klucza MS (lub EMS)?

g. Ile rekordów SSL jest uwzględnianych po stronie klienta przy tworzeniu zaszyfrowanej wiadomości w fazie negocjacji?

h. Ile rekordów SSL jest uwzględnianych po stronie serwera przy tworzeniu zaszyfrowanej wiadomości w fazie negocjacji?



20. W punkcie 8.5.1 pokazano, że jeśli nie zostaną zastosowane numery sekwencyjne, Inga („kobieta w środku") może wyrządzić szkody w sesji SSL przez zmianę kolejności segmentów TCP. Czy Inga może osiągnąć podobny efekt przez usunięcie segmentu TCP? Czego potrzebuje, aby z powodzeniem przeprowadzić atak przez usunięcie danych? Jakie skutki przyniesie taki atak?
21. Załóżmy, że Alicja i Bartek komunikują się z wykorzystaniem sesji SSL. Przyjmijmy, że napastnik nieznający żadnego ze wspólnych kluczy wstawia do strumienia pakietów sfałszowany segment TCP o prawidłowej sumie kontrolnej i numerze sekwencyjnym TCP (oraz właściwym adresie IP i numerze portu). Czy protokół SSL po stronie odbiorcy zaakceptuje sfałszowany pakiet i przekaże jego treść do aplikacji? Dlaczego (lub dlaczego nie)?
22. Poniższe stwierdzenia dotyczą rysunku 8.29. Prawda czy fałsz?
    a. Kiedy host z sieci 172.16.1/24 wyśle datagram do serwera Amazon.com, router R1 zaszyfruje datagram za pomocą protokołu IPsec.
    b. Kiedy host z sieci 172.16.1/24 wyśle datagram do hosta z sieci 172.16.2/24, router R1 zmieni źródłowy i docelowy adres datagramu IP.
    c. Załóżmy, że host z sieci 172.16.1/24 zainicjuje połączenie TCP z serwerem WWW z sieci 172.16.2/24. W ramach tego połączenia wszystkie datagramy wysyłane przez router R1 będą miały numer protokołu 50 w pierwszym od lewej polu nagłówka protokołu IPv4.
    d. Rozważ wysyłanie segmentu TCP z hosta z sieci 172.16.1/24 do hosta z sieci 172.16.2/24. Załóżmy, że potwierdzenie dla tego segmentu zaginęło, dlatego protokół TCP ponownie wysyła ten sam segment. Ponieważ w protokole IPsec stosowane są numery sekwencyjne, router R1 nie wyśle ponownie segmentu.
23. Zastanówmy się nad przykładem z rysunku 8.29. Załóżmy, że Inga zajmuje pozycję „kobiety w środku" i może wstawiać datagramy do strumienia danych przesyłanych z routera R1 do R2. W ramach ataku przez odtwarzanie Inga wysyła duplikat jednego z datagramów przesłanych z routera R1 do R2. Czy router R2 odszyfruje powielony datagram i przekaże go do sieci oddziału firmy? Jeśli nie, opisz szczegółowo, w jaki sposób router R2 wykryje zduplikowany datagram.
24. Rozważmy opisany dalej protokół pseudo-WEP. Klucz ma cztery bity, a wektor IV — dwa bity. Wektor IV jest dołączany za kluczem w czasie generowania strumienia szyfrującego. Załóżmy, że wspólny niejawny klucz to 1010. Strumienie szyfrujące dla czterech możliwych danych wejściowych to:

    101000: 0010101101010101001011010100100. . .
    101001: 1010011011001010110100100101101. . .
    101010: 0001101000111100010100101001111. . .
    101011: 1111101010000000101010100010111. . .

Przypuśćmy, że wszystkie wiadomości mają długość ośmiu bitów. Załóżmy, że suma kontrolna ICV (ang. *Integrity Check*) ma długość czterech bitów i jest obliczana przez wykonanie operacji XOR na czterech pierwszych i czterech ostatnich bitach danych. Przyjmijmy, że pakiet protokołu pseudo-WEP składa się z trzech pól — pola wektora IV, pola komunikatu i pola sumy kontrolnej ICV, przy czym niektóre z nich są zaszyfrowane.

a. Chcemy wysłać wiadomość m = 10100000 przy użyciu wektora IV = 11 i protokołu WEP. Jakie będą wartości w trzech polach tego protokołu?

b. Wykaż, że kiedy odbiorca odszyfruje pakiet WEP, odtworzy wiadomość i sumę kontrolną ICV.

c. Załóżmy, że Inga przechwyciła pakiet WEP (niekoniecznie z wektorem IV = 11) i chce go zmodyfikować przed przekazaniem odbiorcy. Przyjmijmy, że Inga zmienia pierwszy bit sumy kontrolnej ICV. Jakie jeszcze bity Inga musi zmodyfikować, aby otrzymany pakiet miał odpowiednią wartość ICV (zakładamy, że Inga nie zna strumieni szyfrujących dla żadnego wektora IV)?

d. Uzasadnij odpowiedź przez zmodyfikowanie bitów w pakiecie WEP z punktu (a), zaszyfrowanie wynikowego pakietu i zweryfikowanie integralności.

25. Przedstaw tabelę filtra i tabelę połączeń zapory stanowej, tak aby była ona jak najbardziej restrykcyjna, a przy tym spełniała poniższe warunki:

a. Umożliwiała wszystkim wewnętrznym użytkownikom nawiązywanie sesji telnetowych z zewnętrznymi hostami.

b. Umożliwiała zewnętrznym użytkownikom przeglądanie witryny firmy dostępnej pod adresem 222.22.0.12.

c. Blokowała cały pozostały ruch przychodzących i wychodzący.

Adres sieci wewnętrznej to 222.22/16. W rozwiązaniu przyjmij, że tabela połączeń przechowuje obecnie informacje o trzech połączeniach (wszystkie one prowadzą z sieci poza nią). Musisz wymyślić odpowiednie adresy IP i numery portów.

26. Załóżmy, że Alicja chce odwiedzić witrynę activist.com za pomocą usługi podobnej do TOR. Usługa ta oparta jest na dwóch niezależnych serwerach pośredniczących — Proxy1 i Proxy2. Alicja najpierw uzyskuje certyfikaty (każdy z nich obejmuje klucz publiczny) tych serwerów z serwera centralnego. Jako $K_1^+()$, $K_2^+()$, $K_1^-()$ i $K_2^-()$ oznaczmy operacje szyfrowania i deszyfrowania za pomocą publicznego oraz prywatnego klucza RSA.

a. Za pomocą diagramu czasowego przedstaw jak najprostszy protokół, który umożliwi Alicji określenie klucza sesji S_1 wspólnego z serwerem Proxy1. Oznacz jako $S_1(m)$ operację szyfrowania i deszyfrowania danych m za pomocą wspólnego klucza S_1.

b. Przedstaw na diagramie czasowym jak najprostszy protokół, który umożliwi Alicji określenie klucza sesji S_2 wspólnego z serwerem Proxy2 *bez ujawniania temu serwerowi swojego adresu IP*.

c. Załóżmy teraz, że wspólne klucze S_1 i S_2 są już ustalone. Przedstaw na diagramie czasowym jak najprostszy protokół, który *nie korzysta z szyfrowania z kluczem publicznym*, ale umożliwia Alicji zażądanie strony HTML z witryny activist.com *bez ujawniania swojego adresu serwerowi Proxy1* i *bez ujawniania odwiedzanej witryny serwerowi Proxy2*. Diagram powinien kończyć się przesłaniem żądania HTTP do witryny activist.com

Zagadnienia do dyskusji

1. Przypuśćmy, że intruz może wstawiać do sieci i usuwać z sieci komunikaty DNS. Podaj trzy scenariusze problemów, które mógłby spowodować taki intruz.
2. Czym jest Kerberos? Jak działa? W jaki sposób związany jest z problemem 10. z tego rozdziału?
3. Skoro IPsec zapewnia bezpieczeństwo w warstwie sieci, dlaczego potrzebne są dodatkowe zabezpieczenia w wyższych warstwach?
4. Poszukaj informacji o botnetach. Jakich protokołów i systemów używają obecnie napastnicy do kontrolowania botnetów i ich aktualizowania?

Ćwiczenie realizowane za pomocą narzędzia Wireshark

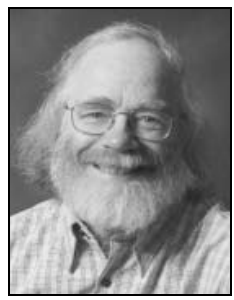
W tym ćwiczeniu badamy protokół SSL. W podrozdziale 8.5 wyjaśniliśmy, że SSL służy do zabezpieczania połączeń TCP i jest powszechnie stosowany do chronienia transakcji w internecie. W tym ćwiczeniu skoncentrujemy się na rekordach SSL wysyłanych przez połączenie TCP. Spróbujemy opisać i sklasyfikować poszczególne rekordy, aby zrozumieć ich funkcje oraz mechanizmy działania. Zbadamy różne typy rekordów SSL, a także pola komunikatów przesyłanych za pomocą tego protokołu. Aby to zrobić, przeanalizujemy zapisy rekordów SSL przesyłanych między hostem Czytelnika i serwerem sklepu internetowego.

Ćwiczenie z wykorzystaniem protokołu IPsec

W tym ćwiczeniu (jest ono dostępne w witrynie poświęconej książce) pokażemy, jak utworzyć kanały IPsec SA między komputerami z systemem Linux. Pierwszą część zadania można wykonać za pomocą dwóch zwykłych komputerów tego typu mających po jednej karcie ethernetowej. Jednak drugi fragment ćwiczenia wymaga dostępu do czterech maszyn z systemem Linux, a każda z nich musi być wyposażona w dwie karty ethernetowe. W tej części utworzysz kanały IPsec SA za pomocą protokołu ESP w trybie tunelowym. Najpierw utworzysz takie kanały ręcznie, a następnie za pomocą protokołu IKE.

WYWIAD Z…

Steven M. Bellovin



Steven M. Bellovin po wielu latach pracy w Laboratorium Badań Usług Sieciowych przy Laboratoriach Badawczych AT&T we Florham Park w stanie New Jersey został wykładowcą Uniwersytetu Columbia. Specjalizuje się w sieciach, bezpieczeństwie oraz tym, co sprawia, że jedno kłóci się z drugim. W 1995 roku odebrał nagrodę Usenix Liftime Achievement Award za pracę nad stworzeniem Usenetu, pierwszej sieci wymiany wiadomości, która mogła łączyć dwa lub więcej komputerów i pozwalała użytkownikom na dzielenie się informacjami i prowadzenie dyskusji. Steve został również wybrany na członka Narodowej Akademii Inżynierii. Uzyskał licencjat na Uniwersytecie Columbia i doktorat na Uniwersytecie Północnej Karoliny w Chapel Hill.

Dlaczego postanowił Pan specjalizować się w bezpieczeństwie sieci?

Zabrzmi to dziwnie, ale odpowiedź jest prosta: bo mnie to bawiło. Odebrałem wykształcenie w programowaniu i administracji systemów, co dość naturalnie doprowadziło do bezpieczeństwa. Zawsze też byłem zainteresowany komunikacją, już kiedy pracowałem dorywczo nad programowaniem systemów, kiedy uczyłem się w szkole średniej.

Zajmuję się bezpieczeństwem z dwóch powodów — chcę, żeby komputery były użyteczne, co oznacza, że ich funkcje nie mogą być zakłócane przez napastników, i pragnę chronić prywatność.

Jak wyobrażał Pan sobie Usenet, gdy go Pan projektował? A jak teraz?

Początkowo miał to być sposób na ogólnokrajową debatę o informatyce i programowaniu systemów, z licznymi zastosowaniami lokalnymi do spraw administracyjnych, ogłoszeń o sprzedaży itd. W rzeczywistości moje wstępne prognozy zakładały jedną, dwie wiadomości dziennie z najwyżej 50 – 100 ośrodków. Największy rozwój nastąpił jednak w tematach związanych z ludźmi, w tym — choć nie tylko — ludzkimi interakcjami z komputerami. Najbardziej lubię grupy w rodzaju rec.woodworking, a także sci.crypt.

Do pewnego stopnia grupy dyskusyjne zostały wyparte przez sieć WWW. Gdybym zaczął je projektować dzisiaj, wyglądałyby zupełnie inaczej. Pomimo to nadal pozwalają dotrzeć do szerokiego audytorium bez zdawania się na konkretne witryny WWW.

Czy ktoś zainspirował Pana w sprawach zawodowych? W jaki sposób?

Ogromny wpływ na moją karierę wywarł profesor Fred Brooks — założyciel i pierwszy przewodniczący wydziału informatyki na Uniwersytecie Północnej Kalifornii w Chapel Hill, kierownik zespołu, który zaprojektował IBM S/360 i OS/360, oraz autor *Mitycznego*

osobomiesiąca. Bardziej niż czegokolwiek innego uczył praktyczności i pragmatyczności — jak rozpatrywać problemy w kontekście rzeczywistego świata (który jest znacznie bardziej zagmatwany, niż życzyłby sobie teoretyk) i jak godzić sprzeczne interesy w projekcie rozwiązania. Większość pracy z komputerami to inżynieria — sztuka znajdowania kompromisów, które pozwalają osiągnąć wiele przeciwstawnych celów.

Jak wyobraża Pan sobie przyszłość łączności sieciowej i bezpieczeństwa?

Dotychczas większość zabezpieczeń opierała się na izolacji. Na przykład działanie zapory sieciowej polega na odcinaniu dostępu do określonych komputerów i usług. Żyjemy jednak w erze coraz intensywniejszej łączności — coraz trudniej jest pozostawać w izolacji. Co gorsza, współczesne systemy składają się z wielu odrębnych części połączonych sieciami. Zabezpieczenie tego wszystkiego jest jednym z największych wyzwań.

Jakie według Pana były największe postępy w dziedzinie bezpieczeństwa? Jak daleko jesteśmy od celu?

Z naukowego punktu widzenia wiemy już, o co chodzi w kryptografii. To bardzo pomaga. Ale większość problemów z bezpieczeństwem bierze się ze źle napisanego kodu, a to jest znacznie trudniejszy problem. Prawdę mówiąc, jest to najstarszy nierozwiązany problem w całej informatyce i myślę, że tak już zostanie. Musimy wymyślić, jak budować bezpieczne systemy z niezabezpieczonych komponentów. Potrafiliśmy rozwiązać problem niezawodności mimo awarii sprzętu; czy zdołamy zrobić to samo z bezpieczeństwem?

Czy ma Pan jakieś rady dla osób, które studiują internet i bezpieczeństwo sieci?

Nauka mechanizmów jest łatwa. Trudniej jest nauczyć się „myśleć paranoicznie". Trzeba pamiętać, że krzywa prawdopodobieństwa nie ma tu zastosowania — napastnicy potrafią znaleźć i wykorzystać najmniej prawdopodobny zbieg okoliczności. Poza tym ważne — bardzo ważne — są szczegóły.