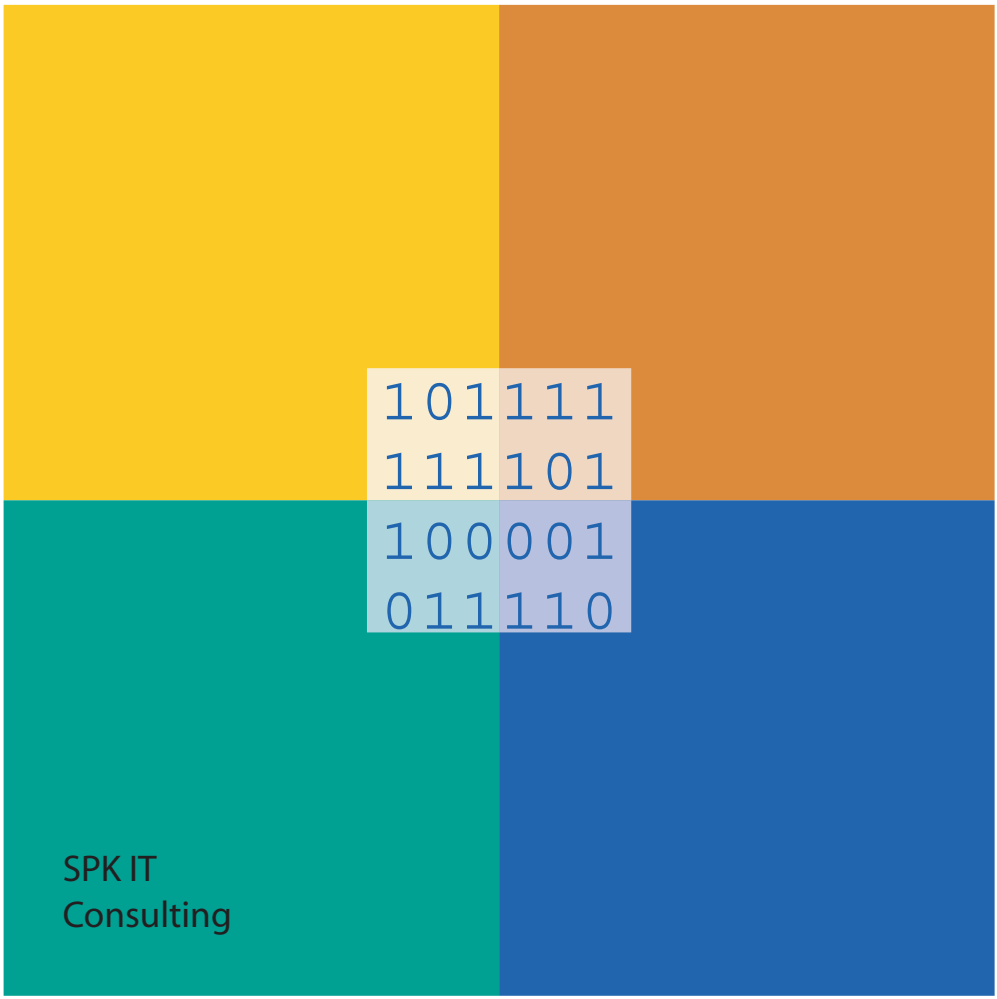
TEORI & APLIKASI KRIPTOGRAFI

Sentot Kromodimoeljo



SPK IT
Consulting

Teori dan Aplikasi Kriptografi

Sentot Kromodimoeljo

Januari 2010

Teori dan Aplikasi Kriptografi
Penulis: Sentot Kromodimoeljo
Januari 2010
ISBN 978-602-96233-0-7
453 halaman, 14.85 x 21 cm

Penerbit: SPK IT Consulting

Typesetting menggunakan LaTeX.

Sentot Kromodimoeljo
Consultant, SPK IT Consulting
Information Technology, Cryptography, Mathematical Logic
sentotk2000@yahoo.ca

Untuk anak-anakku yang tercinta, Kelsey dan Zakrie.

Kata Pengantar

Sangat banyak buku dalam bahasa Indonesia mengenai ilmu dan teknologi komputer yang telah diterbitkan, namun hampir semua bersifat kontemporer. Buku kontemporer memang berharga dan diperlukan, tetapi untuk pembaca yang ingin belajar teori ilmu komputer secara mendalam, diperlukan buku yang dapat memberi motivasi dan arahan untuk studi lanjut.

Buku ini mencoba menjelaskan teori dan praktek kriptografi dan ditujukan terutama kepada pembaca yang ingin memperdalam pengetahuannya mengenai kriptografi. Banyak orang yang enggan membaca buku yang berisi matematika karena presentasi matematika biasanya hambar tanpa motivasi. Kriptografi tidak bisa dipisahkan dari matematika, jadi buku ini juga berisi matematika, akan tetapi penulis mencoba menggunakan bahasa yang sederhana dan memberi motivasi dan penjelasan untuk setiap rumus matematika agar mudah dipahami dan tidak membosankan. Memang untuk mendalami ilmu kriptografi tidak ada jalan pintas dan diperlukan ketekunan yang luar biasa. Tetapi buku ini dapat juga dibaca untuk mendapatkan pengetahuan *superficial* mengenai ilmu kriptografi, dengan melewatkan atau membaca secara sepintas saja bagian-bagian yang sukar matematikanya. Tentunya teori tanpa aplikasi adalah sia-sia, oleh sebab itu aplikasi kriptografi juga dibahas.

Mudah-mudahan buku ini berguna bagi pembaca yang ingin memperdalam pengetahuan di bidang kriptografi. Kalau ada kesalahan dalam buku ini, mohon maaf sebelumnya. Selamat membaca!

Sentot Kromodimoeljo.

Terima kasih

Penulis ingin mengucapkan terima kasih yang sebesarnya kepada Dr. Laksana Tri Handoko dari Pusat Penelitian Fisika LIPI, Prof. Chan Basaruddin, Dr. Lim Yohanes Stefanus dan Dr. Setiadi Yazid dari Fakultas Ilmu Komputer UI, dan Dr. Budi Susilo Soepandji dari Departemen Pertahanan, atas komentar, saran dan dukungannya.

Penulis juga ingin mengucapkan terima kasih yang sebesarnya kepada semua teman di KINDO 21 atas komentar, saran dan dukungan yang telah diberikan, terutama kepada Kemal Surianegara, Neni Sintawardani, Arnold Soetrisnanto, E.T. Sari, Ardi Sutedja, Hartojo Wignjowijoto, Hidayat Brata dan Yandri Susanto.

Daftar Isi

Daftar Gambar

Daftar Tabel

Bab 1

Pendahuluan

Di jaman Romawi kuno, Julius Caesar telah menggunakan teknik kriptografi yang dijuluki *Caesar cipher* untuk mengirim pesan secara rahasia, meskipun teknik yang digunakannya sangat tidak memadai untuk ukuran kini. Casanova menggunakan pengetahuan mengenai kriptografi untuk mengelabui Madame d'Urfé (ia mengatakan kepada Madame d'Urfé bahwa sesosok jin memberi tahu kunci rahasia Madame d'Urfé kepadanya, padahal ia berhasil memecahkan kunci rahasia berdasarkan pengetahuannya mengenai kriptografi), sehingga ia mampu mengontrol kehidupan Madame d'Urfé secara total. Sewaktu perang dunia kedua, pihak sekutu berhasil memecahkan kode mesin kriptografi Jerman, Enigma; keberhasilan yang sangat membantu pihak sekutu dalam memenangkan perang[1]. Sejarah kriptografi penuh dengan intrik dan banyak orang melihat kriptografi sebagai sesuatu yang penuh dengan misteri.

Juga banyak orang yang menganggap pengetahuan dan penggunaan kriptografi sebagai perbuatan subversif, bahkan Amerika Serikat pernah memperlakukan kriptografi sebagai senjata yang tidak dapat diekspor secara bebas, meskipun sekarang peraturan sudah diperlunak. Di satu sisi pemerintah AS tidak menginginkan warganya dan pihak lain menggunakan kriptografi untuk berbagai macam keperluan, di sisi lain kriptografi sangat diperlukan dalam aplikasi pengamanan teknologi informasi seperti transaksi perbankan. Ada beberapa negara, baik negara belum maju yang memiliki pemerintahan represif maupun negara barat yang "liberal", yang melarang atau membatasi penggunaan kriptografi.

Pro dan kontra penggunaan kriptografi tidak akan dibahas dalam buku ini. Tujuan buku ini adalah untuk mengungkap tabir misteri yang menutupi ilmu kriptografi. Ini dilakukan dengan memperkenalkan berbagai konsep dasar krip-

[1]Setelah perang berahir, konon pihak sekutu menjual mesin Enigma ke beberapa negara berkembang tanpa memberi tahu bahwa kode sudah dipecahkan.

tografi, penjelasan mengenai teori matematika dan teknik-teknik kriptografi, contoh-contoh aplikasi kriptografi, diskusi mengenai kendala aplikasi kriptografi, dan diskusi mengenai masa depan kriptografi.

Konsep-konsep dasar kriptografi sangat perlu untuk dipahami, karena tanpa konsep dasar, segala macam teori matematika tidak ada gunanya. Berbagai konsep yang akan dibahas antara lain: konsep acak, *one-time pad*, *cryptanalysis*, berbagai operasi dasar untuk kriptografi, dan manajemen kunci.

Berbagai teknik enkripsi klasik akan dibahas, mulai dari yang sederhana seperti transformasi linear dan transformasi *affine*, sampai dengan *stream cipher* dan *block cipher*. Teknik enkripsi yang lemah dibahas bukan untuk diimplementasi, tetapi agar pembaca dapat lebih memahami apa kelemahannya dan bagaimana teknik enkripsi yang lebih tangguh menanggulanginya. Teknik-teknik yang digunakan untuk mencari atau mengeksploitasi kelemahan enkripsi antara lain analisa frekuensi, *differential cryptanalysis* dan *linear cryptanalysis*. Matematika yang digunakan untuk transformasi enkripsi juga akan dibahas secara rinci sebelum pembahasan teknik enkripsi, akan tetapi pembahasan teori probabilitas dan statistika hanya sebatas penggunaan.

Buku ini tidak merekomendasikan penggunaan *stream cipher* karena sangat mudah untuk menggunakannya secara tidak aman. Algoritma RC4 jelas merupakan algoritma yang lemah. Semua ini akan dibahas di bab 6.

Cryptographically secure hashing juga akan dibahas di buku ini, termasuk MD5 dan SHA. Sebagai contoh analisa kekuatan algoritma *hashing*, *differential cryptanalysis* terhadap MD5 akan dibahas.

Teknik-teknik kriptografi *public key* yang dibahas termasuk yang berbasis pada sukarnya penguraian bilangan yang sangat besar (contohnya RSA), yang berbasis pada sukarnya komputasi logaritma diskrit (contohnya ElGamal), dan juga yang menggunakan *elliptic curve* menggantikan *finite field*. Teknik yang bersifat probabilistik, yaitu *zero-knowledge protocol*, dan algoritma Merkle-Hellman untuk enkripsi *knapsack* dengan kelemahannya, juga akan dibahas. Untuk memberi gambaran mengenai sukarnya penguraian dan komputasi logaritma diskrit, berbagai teknik penguraian dan komputasi logaritma diskrit dibahas.

Algoritma Euclid, *Fermat's Little Theorem* dan *Chinese Remainder Theorem* adalah 3 topik sangat penting dan umum yang wajib dikuasai oleh setiap murid kriptografi atau ilmu komputer. Sebaliknya, bab mengenai *algebraic number* dan bagian yang membahas metode *number field sieve* berisi bahan tingkat sangat lanjut dan khusus, jadi mungkin sebaiknya tidak digunakan dalam mata kuliah tingkat awal.

Quantum key distribution juga akan dibahas. Jika *quantum key distribution* menjadi sesuatu yang praktis, maka enkripsi *one-time pad* juga akan menjadi sesuatu yang praktis. Namun *quantum key distribution* bersifat *point-to-point* secara fisik, jadi aplikasinya terbatas.

Berbagai aplikasi kriptografi akan dibahas, termasuk aplikasi untuk pengamanan sesi, aplikasi pengamanan *email*, aplikasi *authentication*, *public key infrastructure* dan *cryptographic library*.

Kendala aplikasi kriptografi terutama disebabkan oleh kurangnya pemahaman mengenai kriptografi sehingga banyak terjadi penggunaan kriptografi dengan cara yang salah, terutama dalam hal *key management*. Penggunaan kriptografi harus dengan cara yang benar, dan implementasi harus dengan teliti dan hati-hati. Kalau tidak, akibatnya adalah kriptografi yang mudah untuk dipecahkan. Penggunaan kriptografi secara benar bukan sesuatu yang mudah dan sejarah penggunaan kriptografi menunjukkan bahwa kesalahan tidak hanya dilakukan oleh mereka yang "gagap teknologi." Perusahaan besar dibidang teknologi informasi seperti Microsoft dan Netscape juga pernah melakukan implementasi yang tidak aman seperti enkripsi *password* yang sangat mudah untuk dipecahkan. Pakar-pakar protokol komunikasi juga telah sering membuat kesalahan, sebagai contoh versi pertama dari pengamanan Wi-Fi (protokol komunikasi nirkabel) yaitu WEP menggunakan *stream cipher* RC4 secara tidak aman (lihat bab 6). Selain masalah sulitnya manajemen kunci, rumitnya sistem dan sistem yang tidak sesuai kebutuhan menjadi kendala penggunaan kriptografi.

Cara kerja protokol yang digunakan untuk sistem pengamanan dengan kriptografi sangat menentukan keamanan sistem. Oleh sebab itu suatu metode untuk menganalisa protokol kriptografi akan dibahas. Metode ini menggunakan pendekatan logika klasik.

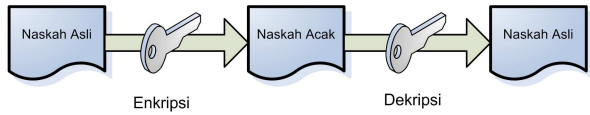
Masa depan kriptografi akan dipengaruhi oleh perkembangan matematika, terutama dalam hal algoritma, dan juga akan dipengaruhi oleh perkembangan di bidang *hardware*. Potensi *quantum computing* juga berperan sangat besar: jika *quantum computer* dapat direalisir maka teknik kriptografi yang digunakan sekarang tidak akan berfungsi efektif karena kunci akan dapat dengan mudah dipecahkan.

Bab 2

Konsep-konsep Dasar

Apakah sebenarnya kriptografi itu? Kriptografi adalah ilmu mengenai teknik enkripsi dimana data diacak menggunakan suatu kunci enkripsi menjadi sesuatu yang sulit dibaca oleh seseorang yang tidak memiliki kunci dekripsi. Dekripsi menggunakan kunci dekripsi mendapatkan kembali data asli. Proses enkripsi dilakukan menggunakan suatu algoritma dengan beberapa parameter. Biasanya algoritma tidak dirahasiakan, bahkan enkripsi yang mengandalkan kerahasiaan algoritma dianggap sesuatu yang tidak baik. Rahasia terletak di beberapa parameter yang digunakan, jadi kunci ditentukan oleh parameter. Parameter yang menentukan kunci dekripsi itulah yang harus dirahasiakan (parameter menjadi ekuivalen dengan kunci).

Dalam kriptografi klasik, teknik enkripsi yang digunakan adalah enkripsi simetris dimana kunci dekripsi sama dengan kunci enkripsi. Untuk *public key cryptography*, diperlukan teknik enkripsi asimetris dimana kunci dekripsi tidak sama dengan kunci enkripsi. Enkripsi, dekripsi dan pembuatan kunci untuk teknik enkripsi asimetris memerlukan komputasi yang lebih intensif dibandingkan enkripsi simetris, karena enkripsi asimetris menggunakan bilangan-bilangan yang sangat besar. Namun, walaupun enkripsi asimetris lebih "mahal" dibandingkan enkripsi simetris, *public key cryptography* sangat berguna untuk *key management* dan *digital signature*.



Gambar 2.1: Proses enkripsi dan dekripsi

Gambar diatas menunjukkan efek dari proses enkripsi dan proses dekripsi.

Secara garis besar, proses enkripsi adalah proses pengacakan "naskah asli" (*plaintext*) menjadi "naskah acak" (*ciphertext*) yang "sulit untuk dibaca" oleh seseorang yang tidak mempunyai kunci dekripsi. Yang dimaksud dengan "sulit untuk dibaca" disini adalah probabilitas mendapat kembali naskah asli oleh seseorang yang tidak mempunyai kunci dekripsi dalam waktu yang tidak terlalu lama adalah sangat kecil. Jadi suatu proses enkripsi yang baik menghasilkan naskah acak yang memerlukan waktu yang lama (contohnya satu juta tahun)[1] untuk didekripsi oleh seseorang yang tidak mempunyai kunci dekripsi. Satu cara untuk mendapatkan kembali naskah asli tentunya dengan menerka kunci dekripsi, jadi proses menerka kunci dekripsi harus menjadi sesuatu yang sulit. Tentunya naskah acak harus dapat didekripsi oleh seseorang yang mempunyai kunci dekripsi untuk mendapatkan kembali naskah asli.

Walaupun awalnya kriptografi digunakan untuk merahasiakan naskah teks, kini kriptografi digunakan untuk data apa saja yang berbentuk digital.

2.1 Konsep Acak

Yang dimaksud dengan sifat acak (*randomness*) dalam kriptografi adalah sifat bebas dari kecenderungan sehingga tidak mudah untuk diterka. Dari segi matematika, jika suatu variabel dianggap bersifat acak, maka teori probabilitas dapat digunakan untuk memprediksi "kelakuan" dari variabel tersebut, antara lain variabel akan memenuhi beberapa kriteria statistik. Metode statistika dapat digunakan, berdasarkan apa yang sudah terjadi, untuk menilai apakah variabel memenuhi kriteria statistik untuk variabel acak. Akan tetapi jika kriteria statistik terpenuhi, belum tentu variabel benar acak, karena sesuatu yang deterministik seperti *pseudo-random number generator* dapat memenuhi kriteria statistik untuk variabel acak. Jadi kriteria statistik bukan merupakan definisi untuk variabel acak.

Sifat acak memang tidak dapat didefinisikan secara matematis, sebab sesuatu yang mempunyai definisi matematis sudah tidak bersifat acak. Apalagi jika definisi berupa rumus yang dapat digunakan untuk kalkulasi, yang didefinisikan bukan saja mudah diterka tetapi tidak perlu diterka.

Sifat acak dapat dikaitkan dengan urutan *events*, dimana *event* berikutnya dalam suatu urutan tidak mudah untuk diterka berdasarkan apa yang sudah lalu. Sifat ini diperlukan dalam pembuatan kunci (*key generation*) supaya kunci dekripsi tidak mudah untuk diterka.

Sifat acak juga dikaitkan dengan tidak adanya korelasi (atau korelasi yang mendekati nol). Dalam kriptografi, tidak diinginkan adanya korelasi antara naskah asli dengan naskah acak atau kunci dengan naskah acak. Ini untuk

[1]Dalam kriptografi, waktu yang dimaksud adalah rerata waktu, ada kemungkinan waktu lebih singkat dan ada kemungkinan waktu lebih lama.

mempersulit analisa seperti analisa frekuensi (*frequency analysis*) atau analisa lebih canggih seperti *linear cryptanalysis* atau *differential cryptanalysis*.

Meskipun tidak sebenarnya acak, sesuatu yang *pseudo-random* berguna dan digunakan dalam kriptografi, tetapi harus dikombinasikan dengan sesuatu yang benar acak. Sebagai contoh, *pseudo-random number generator* dikombinasikan dengan sumber entropi yang benar acak sebagai *seed*, untuk mendapatkan sesuatu yang praktis bersifat *random number generator*.

2.2 One-Time Pad

Secara teoritis, teknik *one-time pad* merupakan teknik enkripsi yang sempurna (*perfect encryption*) asalkan proses pembuatan kunci benar acak.

10010111001011101001	naskah asli
01001110001101001101	kunci
11011001000110100100	naskah acak

Tabel 2.1: Proses enkripsi one-time pad

Dengan *one-time pad*, operasi *exclusive or* (xor) dilakukan pada naskah asli dan kunci secara *bitwise* seperti dalam tabel 2.1. Operasi xor menghasilkan 0 jika argumen sama (0 dengan 0 atau 1 dengan 1) dan menghasilkan 1 jika argumen berbeda (0 dengan 1 atau 1 dengan 0). Jadi bit pertama naskah asli (1) dengan bit pertama kunci (0) menghasilkan bit pertama naskah acak (1), bit kedua naskah asli (0) dengan bit kedua kunci (1) menghasilkan bit kedua naskah acak (1), bit ketiga naskah asli (0) dengan bit ketiga kunci (0) menghasilkan bit ketiga naskah acak (0), dan seterusnya.

11011001000110100100	naskah acak
01001110001101001101	kunci
10010111001011101001	naskah asli

Tabel 2.2: Proses dekripsi one-time pad

Proses dekripsi sama dengan enkripsi tetapi xor dilakukan pada naskah acak, dengan kunci yang sama (kunci dekripsi sama dengan kunci enkripsi). Setiap bit dalam kunci jika dioperasikan terhadap bit dalam naskah asli (seperti dalam proses enkripsi) kemudian dioperasikan lagi terhadap hasil operasi pertama (seperti dalam proses dekripsi) akan mendapatkan kembali bit naskah asli. Ini adalah konsekuensi sifat aljabar operasi xor.

bit naskah	bit kunci	bit hasil operasi
0	0	0
1	0	1
0	1	1
1	1	0

Tabel 2.3: Tabel operasi xor

Tabel 2.3 memperlihatkan operasi xor yang digunakan oleh *one-time pad*. Nilai 0 untuk bit kunci mempertahankan nilai bit naskah yang dioperasikan, jadi dua kali mempertahankan akan tetap mempertahankan nilai bit naskah asli. Nilai 1 untuk bit kunci menghasilkan negasi bit naskah yang dioperasikan, jadi dua kali negasi akan mendapatkan nilai bit semula yaitu nilai bit naskah asli. Alhasil, nilai apapun untuk bit kunci akan mendapatkan kembali nilai bit naskah asli jika dioperasikan dua kali. Operasi *exclusive or* sangat berperan dalam kriptografi: semua algoritma enkripsi simetris modern menggunakan operasi *exclusive or*. Simbol $\oplus$ kerap digunakan sebagai notasi untuk *exclusive or*.

Ketangguhan enkripsi *one-time pad* tergantung pada keacakan kunci yang digunakan. *Key management* menjadi sangat penting dan merupakan kelemahan *one-time pad* yang membuatnya tidak layak untuk penggunaan skala besar[2]. Besar kunci harus menyamai besar naskah asli, dan kunci tidak boleh diguna-ulang. Selain masalah *key generation*, masalah *key distribution* menjadi kendala penggunaan skala besar enkripsi *one-time pad*.

Secara historis, enkripsi *one-time pad* digunakan oleh misi diplomatik berbagai negara di masa lalu untuk komunikasi rahasia. Semacam buku kode yang dibuat secara acak dan tidak boleh diguna-ulang harus dibawa oleh kurir yang dipercaya untuk didistribusikan ke perwakilan diplomatik negara yang bersangkutan. Setiap pengiriman naskah rahasia, kunci sebesar naskah rahasia diambil dari buku kode untuk mengenkripsi naskah. Kunci yang sudah digunakan untuk enkripsi tidak boleh digunakan lagi untuk enkripsi selanjutnya.

One-time pad dapat digunakan untuk komunikasi sangat rahasia dengan volume yang tidak terlalu besar, namun untuk penggunaan skala besar dalam suatu sistem teknologi informasi, *one-time pad* tidak praktis. Walaupun tidak digunakan secara langsung, konsep *one-time pad* "ditiru" dalam teknik enkripsi *stream cipher* (lihat bab 6).

[2]Situasi ini dapat berubah jika *quantum key distribution* menjadi sesuatu yang praktis.

2.3 Cryptanalysis

Cryptanalysis adalah teknik untuk mencoba memecahkan enkripsi, biasanya dengan mencari kunci enkripsi. Ada tiga kategori teknik pencarian kunci yang biasanya digunakan untuk kriptografi klasik yaitu

- *known plaintext attack*,
- analisa statistik, dan
- *brute force search*.

Kombinasi dari teknik-teknik diatas juga kerap digunakan. Biasanya minimal pemecah mempunyai akses ke naskah acak, dan kadang juga mengetahui naskah aslinya. Kita akan bahas ketiga teknik tersebut.

Algoritma enkripsi klasik yang baik adalah algoritma yang tahan terhadap *known plaintext attack* dan analisa statistik sehingga pencarian kunci harus dilakukan dengan *brute force search*. Tentunya kunci enkripsi harus cukup besar agar *brute force search* tidak efektif.

Berbeda dengan kriptografi klasik, kriptografi *public key* mengandalkan keamanannya pada sukarnya komputasi untuk mendapatkan kunci rahasia, yaitu

- penguraian bilangan bulat yang besar, atau
- komputasi logaritma diskrit untuk *finite field* yang besar.

Jadi pemecahan kunci untuk kriptografi *public key* difokuskan pada teknik-teknik untuk mempercepat kedua komputasi tersebut. Kita akan bahas penguraian bilangan bulat di bab 14 dan logaritma diskrit di bab 15.

2.3.1 Known Plaintext Attack

.

Known plaintext attack adalah teknik pencarian kunci enkripsi berdasarkan pengetahuan mengenai pasangan naskah asli - naskah acak. Kita akan gunakan *Caesar cipher* sebagai contoh dari enkripsi yang rentan terhadap *known plaintext attack*.

Julius Caesar menukar setiap huruf dalam naskah asli dengan huruf lain dalam naskah acak. Besar atau kecil huruf dipertahankan dalam naskah acak (huruf besar ditukar dengan huruf besar, huruf kecil ditukar dengan huruf kecil). Spasi, titik, koma dan tanda lainnya tidak ditukar.

Caesar cipher adalah jenis enkripsi yang disebut *simple substitution cipher* dimana setiap huruf dalam naskah asli ditukar dengan huruf lain dalam naskah

acak. Julius Caesar menukar huruf dengan cara *shift transformation*. Rumus umum untuk *shift transformation* adalah:

$$C = \begin{cases} P + b & \text{jika } P + b < n \\ P + b - n & \text{jika } P + b \geq n \end{cases} \tag{2.1}$$

dimana:
C adalah kode bilangan karakter acak,
P adalah kode bilangan karakter asli,
b adalah besarnya *shift*,
n adalah besarnya perbendaharaan karakter (dengan kode 0 sampai $n - 1$).

Jadi rumus untuk enkripsi sesuai dengan relasi ekuivalen:

$$C \equiv P + b \pmod{n}. \tag{2.2}$$

Rumus untuk dekripsi juga sesuai dengan relasi ekuivalen 2.2:

$$P = \begin{cases} C - b & \text{jika } C \geq b \\ C - b + n & \text{jika } C < b. \end{cases} \tag{2.3}$$

Julius Caesar sendiri menggunakan huruf "A" sampai "Z" (dengan kode 0 sampai 25) sebagai perbendaharaan karakter untuk enkripsi (karakter selain huruf tidak dienkripsi), dan menggunakan parameter $b = 3$ menghasilkan rumus enkripsi

$$C = \begin{cases} P + 3 & \text{jika } P < 23 \\ P - 23 & \text{jika } P \geq 23. \end{cases} \tag{2.4}$$

Jadi untuk enkripsi, 0 ("A") ditukar dengan 3 ("D"), 1 ("B") dengan 4 ("E"), ..., 24 ("Y") dengan 1 ("B"), dan 25 ("Z") dengan 2 ("C").

Rumus dekripsi menjadi

$$P = \begin{cases} C - 3 & \text{jika } C \geq 3 \\ C + 23 & \text{jika } C < 3. \end{cases} \tag{2.5}$$

Naskah Asli	`Jangan rahasiakan pesan ini!`
Naskah Acak	`Mdqjdq udkdvldndq shvdq lql!`

Tabel 2.4: Enkripsi dengan *Caesar cipher*

Enkripsi yang menggunakan *shift transformation* seperti *Caesar cipher* sangat rentan terhadap *known plaintext attack*. Jika pasangan naskah asli - naskah acak diketahui, parameter b dapat ditemukan dengan mudah. Sebagai

contoh, jika pasangan dalam tabel 2.4 diketahui, kita dapat menggunakan pasangan huruf acak - huruf asli berdasarkan posisi, misalnya "M" dengan "J". Ini akan segera mendapatkan $b = (12 - 9) \bmod 26 = 3$.

Known plaintext attack terhadap enkripsi *Caesar cipher* adalah contoh *attack* yang bersifat deterministik dimana jika pasangan (atau beberapa pasangan) naskah asli - naskah acak diketahui maka kunci dapat ditemukan dengan pasti. Jika tidak hati-hati, enkripsi *one-time pad* juga sangat rentan terhadap *known plaintext attack* yang bersifat deterministik. Operasi *exclusive or* terhadap naskah asli dan naskah acak langsung mendapatkan kunci. Oleh karena itu kunci untuk *one-time pad* tidak boleh diguna-ulang (itulah maksud nama *one-time pad*: hanya digunakan satu kali). Efek serupa berlaku untuk enkripsi yang "meniru" *one-time pad* seperti *stream cipher* (dimana operasi *exclusive or* terhadap naskah acak dan naskah asli langsung mendapatkan *keystream*), jadi penggunaan *stream cipher* harus dengan sangat hati-hati.

Tidak semua *known plaintext attack* bersifat deterministik. *Linear cryptanalysis* (lihat bagian 8.2) menggunakan *known plaintext attack* tetapi secara probabilistik.

Tingkat kesukaran *known-plaintext attack* tergantung pada rumus yang digunakan untuk enkripsi. Semakin rumit rumus yang digunakan, semakin sukar untuk melakukan *known-plaintext attack*. Rumus yang bersifat linear masih tergolong sederhana dan dianggap rentan terhadap *known-plaintext attack*. Semakin non-linear dan semakin banyak parameter yang digunakan untuk rumus, semakin sulit untuk mengkalkulasi kunci berdasarkan rumus. Ini akan semakin jelas dengan pembahasan berbagai macam enkripsi di bab-bab selanjutnya.

2.3.2 Analisa Statistik

Kecuali *one-time pad*, semua algoritma enkripsi sebelum *Data Encryption Standard* (DES) rentan terhadap analisa statistik. Sebagai contoh, mari kita lihat bagaimana enkripsi dengan cara *shift transformation* seperti *Caesar cipher* rentan terhadap analisa statistik yang sederhana yaitu analisa frekuensi.

Enkripsi dengan cara *shift transformation* sangat rentan terhadap analisa frekuensi sebagai berikut: dengan rumus enkripsi

$$C \equiv P + b \pmod{n},$$

jika n diketahui dan sepasang C dan P dapat diterka dengan akurat, maka parameter b (kunci) dapat dicari. Setiap "pencarian" b dapat dicoba cukup dengan sepasang nilai untuk C dan P. Pasangan nilai yang patut dicoba adalah pasangan yang sesuai dengan statistik frekuensi penggunaan. Sebagai contoh, menggunakan naskah acak dalam tabel 2.4, huruf "D" dan "Q" adalah yang terbanyak digunakan dalam naskah acak. Karena dalam bahasa Indonesia, huruf "A" adalah huruf dengan statistik penggunaan terbesar, jika naskah

asli dalam bahasa Indonesia, maka besar kemungkinan huruf "D" atau "Q" merupakan huruf acak untuk "A". Jadi besar kemungkinan, jika kita menggunakan kode untuk "D" atau "Q" sebagai nilai C dan kode untuk "A" sebagai nilai P, rumus enkripsi akan menghasilkan nilai b yang benar. Jadi kita coba dua kemungkinan: pasangan "D-A" (yang menghasilkan $b = 3$) dan pasangan "Q-A" (yang menghasilkan $b = 16$). Hasil yang dicari adalah nilai b yang jika digunakan untuk mendekripsi naskah acak akan menghasilkan naskah asli yang "masuk akal."

Pasangan	Kode Acak	Kode Asli	Nilai b	Hasil Dekripsi
D-A	3	0	$b = 3$	`Jangan rahasiakan pesan ini!`
Q-A	16	0	$b = 16$	`Wnatna enunfvnxna crfna vav!`

Tabel 2.5: Hasil analisa frekuensi

Berdasarkan hasil analisa frekuensi, yang "masuk akal" hanya $b = 3$ dengan hasil dekripsi "Jangan rahasiakan pesan ini!", jadi kita dapat cukup yakin bahwa parameter $b = 3$.

Analisa frekuensi diatas didasarkan pada pengetahuan bahwa naskah asli adalah dalam bahasa Indonesia dimana huruf "A" mempunyai statistik frekuensi penggunaan terbesar[3]. Tentunya naskah asli tidak akan selalu mempunyai statistik yang mirip dengan data empiris. Tetapi secara umum, semakin panjang naskah yang digunakan untuk analisa, semakin besar kemungkinan statistik penggunaan akan mirip dengan data empiris, yang berarti semakin besar kemungkinan analisa frekuensi akan sukses. Dalam contoh diatas, statistik penggunaan huruf "A" cukup mirip dengan data empiris, jadi analisa frekuensi berhasil.

Untuk *shift transformation*, karena rumus transformasi sangat sederhana hanya dengan satu parameter, setiap percobaan cukup dengan menggunakan satu persamaan. Strategi pencarian yang baik adalah dengan mencoba pasangan yang mempunyai frekuensi penggunaan yang besar (huruf acak yang frekuensinya besar dalam naskah acak dipasangkan dengan huruf asli yang frekuensinya juga besar menurut data empiris). Semakin besar frekuensi terbesar dalam data empiris, secara umum berarti semakin besar *redundancy* dari segi teori informasi, yang akan mempermudah analisa frekuensi.

Jika rumus transformasi lebih rumit dengan lebih dari satu parameter, maka setiap percobaan harus dilakukan dengan lebih dari satu persamaan, dengan banyaknya persamaan yang dibutuhkan sedikitnya sama dengan banyaknya parameter yang harus dicari.

[3]Untuk bahasa Inggris, frekuensi terbesar adalah untuk huruf "E" dan statistik penggunaan untuk semua huruf cukup diketahui berdasarkan pengamatan empiris.

Untuk sukses dalam analisa frekuensi, dibutuhkan pengetahuan empiris mengenai statistik penggunaan huruf, naskah acak yang dapat dianalisa harus cukup besar, rumus atau seminimnya jenis enkripsi harus diketahui (jika rumus tidak diketahui tetapi jenis enkripsi diketahui berupa *simple substitution*, setiap huruf acak harus dipasangkan dengan huruf asli). Untuk analisa frekuensi yang rumit, penggunaan komputer sangat membantu.

Pada umumnya, semakin besar data yang digunakan sebagai dasar, baik untuk probabilitas *a priori* seperti frekuensi penggunaan huruf, maupun yang bersifat *a posteriori* yaitu naskah acak, semakin pasti (tidak tentatif) hasil percobaan kalkulasi. Sebagai contoh, untuk *shift transformation* dengan perbendaharaan 26 huruf dan naskah dalam bahasa Inggris, analisa frekuensi dengan naskah acak sebesar 50 karakter atau lebih akan mendapatkan hasil dengan kepastian mendekati 100 persen, berdasarkan pengetahuan *a priori* mengenai penggunaan huruf dalam bahasa Inggris.

Secara umum, enkripsi yang rentan terhadap *known plaintext attack* yang deterministik juga rentan terhadap analisa frekuensi, jika data empiris mengenai statistik naskah asli diketahui. Ini karena analisa frekuensi dapat dipandang sebagai suatu *known plaintext attack* yang probabilistik, dimana pasangan naskah asli - naskah acak diterka atau diperkirakan berdasarkan data empiris.

Analisa frekuensi lebih mudah dilakukan pada *substitution cipher* dibandingkan dengan *block cipher*. Enigma berhasil dipecahkan oleh pihak sekutu menggunakan analisa frekuensi karena enkripsi yang digunakan Enigma adalah *substitution cipher*, meskipun jenisnya adalah *polyalphabetic* (pertukaran huruf berubah terus). Analisa frekuensi terhadap *polyalphabetic substitution cipher* memang jauh lebih sukar dibandingkan dengan analisa frekuensi terhadap *simple substitution cipher*, tetapi jauh lebih mudah dibandingkan analisa frekuensi terhadap *block cipher*. Fakta ini serta beberapa kelemahan Enigma lainnya, seperti tidak pernah menukar huruf dengan huruf yang sama, berhasil dieksploitasi oleh pihak sekutu dibawah pimpinan Alan Turing.

Analisa frekuensi merupakan contoh dari analisa statistik yang tergolong sederhana. Kita akan bahas analisa statistik yang lebih canggih yaitu *linear cryptanalysis* dan *differential cryptanalysis* di bab 8 setelah kita bahas *block cipher*.

2.3.3 Brute Force Search

Satu dari kriteria sistem enkripsi yang baik adalah bahwa dekripsi tanpa kunci hanya dapat dipecahkan dengan cara *brute force search* dimana semua kemungkinan kunci harus dicoba. Tentunya jumlah kemungkinan kunci harus cukup besar sehingga diperlukan waktu yang sangat lama untuk mencoba semua kunci. Jika jumlah kunci yang harus dicoba kurang besar, maka sistem enkripsi rentan terhadap analisa *brute force search*.

Besarnya kunci enkripsi (jumlah bit dalam kunci enkripsi) menentukan jumlah kemungkinan kunci yang harus dicoba dalam *brute force search*. Untuk kunci sebesar n *bits*, jumlah kemungkinan kunci adalah 2^n dan rerata, kunci akan ditemukan setelah kita mencoba 2^{n-1} kemungkinan (setengah dari semua kemungkinan). Jadi enkripsi rentan terhadap *brute force search* jika 2^n kemungkinan kunci dapat dicoba dalam waktu yang tidak terlalu lama. Tentunya selain tergantung pada jumlah kemungkinan kunci yang harus dicoba, waktu yang diperlukan juga tergantung pada kemampuan hardware yang digunakan. Juga batas "waktu yang tidak terlalu lama" tergantung pada aplikasi, apakah kurang dari 1 bulan, kurang dari 5 tahun, kurang dari 1000 tahun, atau ada batas waktu lain.

Caesar cipher, selain dapat dipecahkan dengan analisa frekuensi atau *known plaintext attack*, dapat juga dipecahkan dengan *brute force search* karena semua kemungkinan kunci ($b = 1$ sampai dengan $b = 25$) dapat dicoba dalam waktu yang tidak terlalu lama. Kita gunakan contoh naskah acak yang digunakan dalam analisa frekuensi (lihat 2.3.2), yaitu "`Mdqjdq udkdvldndq shvdq lql!`".

Tabel 2.6 memperlihatkan hasil *brute force search* terhadap naskah acak *Caesar cipher*. Dari semua kemungkinan kunci, hanya $b = 3$ yang "masuk akal." Karena jumlah kemungkinan kunci cukup kecil, *brute force search* dapat dilakukan tanpa menggunakan komputer. Jika jumlah kunci yang harus dicoba sangat besar, komputer dapat digunakan untuk membantu analisa.

Besar kunci enkripsi ikut menentukan sukses dari *brute force search*. Dengan kemajuan dibidang hardware untuk melakukan *brute force search* (hardware khusus dapat dibuat untuk melakukan *brute force search* terhadap kunci *block cipher*), enkripsi *block cipher* dengan besar kunci 56 bit (jadi ada 2^{56} kemungkinan) kini dapat dipecahkan dalam waktu yang tidak terlalu lama (kira-kira puluhan menit) dengan hardware seharga 1 juta USD. Dengan hardware yang lebih banyak, waktu yang diperlukan menjadi semakin singkat dengan perbandingan waktu *inverse proportional* terhadap harga. Untuk keamanan, sebaiknya enkripsi *block cipher* menggunakan kunci minimum 128 bit. *Data Encryption Standard* (DES) hanya menggunakan 56 bit untuk kunci, jadi sebaiknya diganti dengan 3DES atau *block cipher* lain seperti CAST, AES dan Blowfish.

2.4 Manajemen Kunci

Aspek manajemen kunci sangat penting dalam aplikasi kriptografi. Manajemen kunci yang tidak baik dapat berakibat fatal. Konon[4], di masa perang dingin, pihak Uni Soviet "kecurian" rahasia penting oleh pihak Amerika Serikat

[4]Tidak diverifikasi oleh penulis.

Nilai b	Hasil Dekripsi
$b = 1$	`Lcpicp tcjcukcmcp rgucp kpk!`
$b = 2$	`Kbohbo sbibtjblbo qftbo joj!`
$b = 3$	`Jangan rahasiakan pesan ini!`
$b = 4$	`Izmfzm qzgzrhzjzm odrzm hmh!`
$b = 5$	`Hyleyl pyfyqgyiyl ncqyl glg!`
$b = 6$	`Gxkdxk oxexpfxhxk mbpxk fkf!`
$b = 7$	`Fwjcwj nwdwoewgwj laowj eje!`
$b = 8$	`Evibvi mvcvndvfvi kznvi did!`
$b = 9$	`Duhauh lubumcueuh jymuh chc!`
$b = 10$	`Ctgztg ktatlbtdtg ixltg bgb!`
$b = 11$	`Bsfysf jszskascsf hwksf afa!`
$b = 12$	`Arexre iryrjzrbre gvjre zez!`
$b = 13$	`Zqdwqd hqxqiyqaqd fuiqd ydy!`
$b = 14$	`Ypcvpc gpwphxpzpc ethpc xcx!`
$b = 15$	`Xobuob fovogwoyob dsgob wbw!`
$b = 16$	`Wnatna enunfvnxna crfna vav!`
$b = 17$	`Vmzsmz dmtmeumwmz bqemz uzu!`
$b = 18$	`Ulyrly clsldtlvly apdly tyt!`
$b = 19$	`Tkxqkx bkrkcskukx zockx sxs!`
$b = 20$	`Sjwpjw ajqjbrjtjw ynbjw rwr!`
$b = 21$	`Rivoiv zipiaqisiv xmaiv qvq!`
$b = 22$	`Qhunhu yhohzphrhu wlzhu pup!`
$b = 23$	`Pgtmgt xgngyogqgt vkygt oto!`
$b = 24$	`Ofslfs wfmfxnfpfs ujxfs nsn!`
$b = 25$	`Nerker velewmeoer tiwer mrm!`

Tabel 2.6: Mencoba semua kemungkinan kunci *Caesar cipher*

akibat masalah dalam manajemen kunci. Ternyata pihak Uni Soviet, karena masalah kurangnya dana, mengguna-ulang kode untuk enkripsi *one-time pad*. Pihak Amerika Serikat berhasil menggunakan kesalahan ini untuk memecahkan sebagian komunikasi rahasia pihak Uni Soviet.

Proses pembuatan kunci sangat penting dan sebaiknya proses ini benar acak. Sumber acak (entropi) dapat diambil dari proses fisika acak seperti proses radio-aktif. Sumber acak dapat juga diambil dari berbagai kejadian (*events*) yang muncul secara acak. *Operating system* seperti unix menggunakan kombinasi *system events* termasuk *interrupts* sebagai sumber entropi[5], yang

[5]Dalam suatu *operating system* biasanya pembuatan entropi menggunakan kombinasi dari banyak sumber agar entropi yang cukup besar didapatkan dalam jangka waktu yang cukup

kemudian dikombinasikan dengan algoritma *pseudo-random number generator* menjadi *random number generator*. Aplikasi kriptografi dapat menggunakan *random number generator* yang disediakan *operating system* untuk pembuatan kunci, akan tetapi sebaiknya ini dilakukan hanya jika *random number generator* yang disediakan cukup acak.

Distribusi kunci secara aman juga penting untuk keperluan pengamanan komunikasi. Sebagai contoh, untuk komunikasi yang diamankan dengan enkripsi simetris, tentunya kedua mitra dalam komunikasi harus menggunakan kunci yang sama. Kunci ini dapat dibuat oleh satu pihak dan dikirim secara aman ke mitra komunikasi. Pengiriman kunci dapat dilakukan *out-of-band* yaitu menggunakan jalur khusus diluar jalur normal komunikasi, atau dilakukan *in-band* melalui jalur normal menggunakan sarana *public key cryptography*. Alternatif dari pengiriman kunci adalah *key agreement*, dimana kedua mitra berpartisipasi membuat kunci tanpa dapat diketahui oleh pihak ketiga. *Key agreement* juga menggunakan sarana *public key cryptography*.

Penyimpanan kunci jelas sangat penting untuk pengamanan sistem enkripsi secara menyeluruh. Kunci yang disimpan secara sembrono akan mudah untuk "dicuri" oleh pihak yang tidak diinginkan. Solusi untuk penyimpanan kunci beraneka ragam, mulai dari penggunaan hardware khusus dimana semua proses kriptografi dilakukan didalam hardware khusus dan kunci enkripsi disimpan dan tidak dapat keluar dari hardware, sampai dengan penyimpanan dalam *file* yang dienkripsi menggunakan password atau *passphrase*. Karena praktis, metode terahir sangat populer, yang berarti pengamanan password menjadi penting.

Pengamanan password juga mempunyai beberapa masalah, dari masalah manusia seperti menulis password di secarik kertas yang ditempelkan ke meja kerja, sampai dengan masalah sistem seperti program yang menyimpan password dalam bentuk teks.

Pada dasarnya masalah akses terhadap sesuatu yang penting seperti kunci enkripsi menjadi masalah *authentication* dan tren saat ini mengarah pada *multiple factor authentication*. Kebenaran identititas seseorang atau sesuatu dinilai dari gabungan berbagai atribut yang cukup unik seperti sidik jari, pengetahuan password, dan kepemilikan sesuatu yang unik lainnya.

2.5 Operasi dasar

Operasi terpenting terhadap unit data dalam kriptografi adalah *exclusive or* (xor), seperti yang digunakan dalam enkripsi *one-time pad*. Operasi xor sangat mudah implementasinya dalam hardware, dan prosesor komputer biasanya memiliki instruksi untuk melakukan *bitwise* xor.

singkat.

Jika *one-time pad* dapat digunakan dalam skala besar, maka enkripsi hanya memerlukan operasi xor. Akan tetapi, *one-time pad* tidak praktis untuk penggunaan skala besar, oleh sebab itu diperlukan operasi lainnya yaitu substitusi dan permutasi.

Substitusi adalah proses penukaran unit data secara utuh, seperti yang dilakukan dalam *Caesar cipher* dimana huruf ditukar dengan huruf. Operasi ini membuat efek *confusion* (pembingungan) terhadap analisa statistik. Spesifikasi dan implementasi operasi ini dapat dilakukan menggunakan tabel jika tabel tidak terlalu besar, dengan nilai yang hendak ditukar digunakan sebagai indeks tabel, dan nilai dalam tabel digunakan sebagai nilai penukar. Contoh dari operasi substitusi menggunakan tabel adalah operasi substitusi S1 yang digunakan algoritma DES, seperti terlihat pada tabel 2.7. Tabel mempunyai 4 baris (dengan indeks 0, 1, 2, 3) dan 16 kolom (dengan indeks 0 sampai dengan 15). Input 6 bit digunakan sebagai indeks baris dan kolom, dengan bit 1 dan bit 6 menentukan indeks baris dan bit 2 sampai dengan bit 5 menentukan indeks kolom. Sebagai contoh, jika input 6 bit adalah 011011, maka indeks baris adalah 1 (karena bit 1, 6 mempunyai nilai 01), dan indeks kolom adalah 13 (karena bit 2, 3, 4, 5 mempunyai nilai 1101). Dengan input 011011, S1 akan menghasilkan 4 bit 0101 karena baris 1 kolom 13 dalam tabel S1 mempunyai nilai 5, yang dalam notasi biner 4 bit adalah 0101.

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Tabel 2.7: Tabel untuk S1

Jika tabel terlalu besar, biasanya operasi substitusi mempunyai rumus untuk mengkalkulasi nilai tukar, seperti rumus enkripsi *Caesar cipher* dan rumus untuk S-box AES. *Caesar cipher* dan S-box untuk AES, meskipun mempunyai rumus yang elegan, dapat juga diimplementasikan menggunakan tabel karena tabel tidak terlalu besar. Akan tetapi, S-boxes untuk DES tidak memiliki rumus yang elegan, jadi biasanya diimplementasikan menggunakan tabel.

Permutasi adalah proses penukaran posisi dalam unit data. Operasi ini membuat efek *diffusion* (pembauran) yang mempersulit analisa statistik. Operasi ini dapat diimplementasikan dalam hardware secara efisien. Spesifikasi operasi permutasi dan implementasi dengan software biasanya dilakukan menggunakan tabel. Posisi awal komponen data digunakan sebagai indeks tabel, dan nilai dalam tabel digunakan sebagai posisi ahir komponen data. Operasi permutasi bersifat *bijective map* dimana tidak ada komponen data yang hilang

dan tidak ada komponen data yang digandakan. Dalam kriptografi, komponen data dalam operasi permutasi biasanya berupa bit. Contoh operasi permutasi adalah *Initial Permutation* yang digunakan algoritma DES, seperti terlihat dalam tabel 2.8. Posisi ahir bit digunakan sebagai indeks tabel, jadi nilai output bit n sama dengan nilai input bit $T(n)$ dimana $T(n)$ adalah nilai komponen n dalam tabel. Sebagai contoh, nilai output bit 1 sama dengan nilai input bit 58, karena komponen pertama dalam tabel mempunyai nilai 58. Permutasi bit dapat diimplementasikan dalam hardware secara efisien karena hanya dibutuhkan koneksi antara posisi awal bit dengan posisi ahir bit. Jadi untuk *Initial Permutation* hanya diperlukan 64 koneksi, dengan bit 58 input menjadi bit 1 output, bit 50 input menjadi bit 2 output, dan seterusnya.

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabel 2.8: Tabel *Initial Permutation*

2.6 Ringkasan

Dalam bab ini, pembaca telah diperkenalkan dengan konsep enkripsi dan dekripsi meskipun hanya secara garis besar. Pembaca juga diperkenalkan dengan konsep acak, suatu konsep yang sangat penting dalam kriptografi. Teknik enkripsi *one-time pad* merupakan teknik enkripsi "sempurna" dari segi teoritis, meskipun dalam prakteknya banyak kendala penggunaannya. Kita melihat bagaimana analisa statistik dapat dipergunakan untuk memecahkan enkripsi yang kurang kuat. Pemecahan enkripsi juga dapat dipermudah jika kita mengetahui naskah asli atau bagian dari naskah asli. Selain analisa statistik, pemecahan enkripsi juga dapat dilakukan dengan *brute force search* jika ruang pencarian relatif tidak terlalu besar. Tidak kalah pentingnya dalam kriptografi adalah masalah manajemen kunci, karena jika tidak hati-hati kunci dapat jatuh ke tangan pihak yang tidak diinginkan. Yang terahir, pembaca diperkenalkan dengan operasi dasar yang banyak digunakan dalam kriptografi. Konsep-konsep yang diperkenalkan dalam bab ini akan digunakan pada bab-bab selanjutnya.

Bab 3

Matematika I - Aritmatika Modular

Aritmatika modular sangat berperan dalam kriptografi karena banyak digunakan dalam algoritma enkripsi, baik untuk enkripsi simetris maupun untuk *public key cryptography*. Dalam aritmatika modular, konsep gcd digunakan antara lain untuk operasi *inverse*. Gcd dapat dikalkulasi secara efisien menggunakan algoritma Euclid, algoritma sangat penting yang telah berusia lebih dari 2000 tahun. Bab ini menjelaskan gcd, algoritma Euclid dan aritmatika modular. Akan tetapi, sebelum itu, kita definisikan terlebih dahulu beberapa struktur aljabar yang banyak digunakan dalam aritmatika, yaitu *group*, *monoid*, *ring* dan *field*.

3.1 Group, Monoid, Ring dan Field

Definisi 1 (Group) *Suatu group G dengan operasi biner $*$ adalah suatu himpunan dengan struktur aljabar sebagai berikut:*

- *Jika $a, b \in G$ maka $(a * b) \in G$ (closure).*
- *$a * (b * c) = (a * b) * c$ (associativity).*
- *Terdapat elemen $e \in G$ dimana $a * e = a = e * a$ untuk setiap $a \in G$ (identity).*
- *Untuk setiap $a \in G$ terdapat $b \in G$ dengan $a * b = e = b * a$ (inverse).*

Untuk commutative group (dinamakan juga Abelian group), terdapat satu syarat lagi:

- $a * b = b * a$ *(commutativity).*

Sebagai contoh, pertambahan dengan bilangan bulat adalah *Abelian group* dengan himpunan semua bilangan bulat, operasi biner $+$, *identity* 0, dan $-a$ sebagai *inverse* untuk a.

Definisi 2 (Monoid) *Suatu monoid G dengan operasi biner $*$ adalah suatu himpunan dengan struktur aljabar sebagai berikut:*

- *Jika $a, b \in G$ maka $(a * b) \in G$ (closure).*
- $a * (b * c) = (a * b) * c$ *(associativity).*
- *Terdapat elemen $e \in G$ dimana $a * e = a = e * a$ untuk setiap $a \in G$ (identity).*

Untuk commutative monoid (dinamakan juga Abelian monoid), terdapat satu syarat lagi:

- $a * b = b * a$ *(commutativity).*

Sebagai contoh, perkalian dengan bilangan bulat adalah *Abelian monoid* dengan himpunan semua bilangan bulat, operasi biner perkalian $\cdot$, dan *identity* 1.

Definisi 3 (Ring) *Suatu ring R dengan operasi $+$ dan $\cdot$ dan elemen 0 dan 1 adalah suatu himpunan dengan struktur aljabar sebagai berikut:*

- *R dengan operasi $+$ mempunyai struktur Abelian group dengan identity 0.*
- *R dengan operasi $\cdot$ mempunyai struktur Abelian monoid dengan identity 1.*
- $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ *(distributivity).*

Jadi yang dimaksud dengan *ring* dalam buku ini adalah *commutative ring with identity*. Operasi $a \cdot b$ kerap disingkat menjadi ab. Sebagai contoh dari *ring* adalah aritmatika bilangan bulat dengan pertambahan dan perkalian. Simbol $\mathbf{Z}$ digunakan sebagai notasi untuk himpunan semua bilangan bulat. Simbol $\mathbf{N}$ digunakan sebagai notasi untuk himpunan semua bilangan bulat non-negatif (disebut juga bilangan natural). Walaupun $\mathbf{N}$ bukan *ring* (karena *inverse* pertambahan tidak berlaku), $\mathbf{N}$ dengan prinsip *well-ordering* sangat berguna dalam pembuktian yang membutuhkan prinsip induksi.

Definisi 4 (Field) *Suatu field F adalah suatu ring dimana setiap $0 \neq a \in F$ mempunyai inverse perkalian a^{-1} dengan $a \cdot a^{-1} = 1$.*

Jadi suatu *field* adalah suatu *ring* R dimana $R \setminus \{0\}$ dengan operasi $\cdot$ membentuk *Abelian group*. Contoh dari *field* adalah aritmatika bilangan nyata dimana semua bilangan kecuali 0 memiliki *inverse* perkalian. Aritmatika bilangan rasional juga merupakan contoh dari *field*. Simbol yang digunakan sebagai notasi untuk himpunan semua bilangan nyata adalah $\mathbf{R}$, sedangkan simbol yang digunakan sebagai notasi untuk himpunan semua bilangan rasional adalah $\mathbf{Q}$. Tentunya $\mathbf{R}$ dan $\mathbf{Q}$ juga merupakan *ring*, akan tetapi $\mathbf{Z}$ bukan *field* karena *inverse* perkalian tidak berlaku untuk bilangan bulat.

3.2 Prinsip Induksi

Banyak teorema dalam aljabar yang pembuktiannya membutuhkan prinsip induksi. Prinsip induksi berlaku pada proposisi mengenai bilangan natural (bilangan bulat non-negatif), tetapi dapat juga digeneralisasi sehingga berlaku untuk proposisi mengenai bilangan ordinal. Dalam buku ini, kita cukup menggunakan prinsip induksi pada bilangan natural.

Teorema 1 (Prinsip Induksi) *Jika P merepresentasikan proposisi mengenai bilangan natural, kita gunakan notasi "$P(n)$" untuk merepresentasikan "proposisi P berlaku untuk bilangan natural n." Jika kita dapat buktikan bahwa:*

$$\begin{aligned} &(i) \quad P(0), \text{ dan} \\ &(ii) \quad P(n) \Longrightarrow P(n+1) \text{ untuk setiap } n \in \mathbf{N}. \end{aligned}$$

Maka $P(n)$ berlaku untuk setiap bilangan $n \in \mathbf{N}$.

Secara informal sangat masuk akal mengapa prinsip induksi berlaku. Kita umpamakan bahwa kita dapat buktikan (i) dan (ii). Dari (i) kita dapat buktikan $P(0)$. Dari $P(0)$ dan (ii) kita dapat buktikan $P(1)$, dan langkah ini dapat diulang m kali untuk membuktikan $P(m)$. Karena kita dapat membuktikan $P(m)$ untuk sembarang $m \in \mathbf{N}$, maka seharusnya $P(n)$ berlaku untuk setiap $n \in \mathbf{N}$. Akan tetapi secara matematis pembuktiannya agak sedikit lebih rumit karena melibatkan himpunan *infinite* yaitu $\mathbf{N}$. Kita tidak akan membahas pembuktian matematis yang formal karena membutuhkan pembahasan fondasi matematika.

$P(0)$ diatas kerap disebut sebagai *base case*. Selain *base case* $P(0)$, prinsip induksi dapat juga digunakan dengan *base case* $P(k)$ dimana k adalah bilangan natural bukan 0. Hasilnya adalah pembuktian $P(n)$ untuk setiap $n \in \mathbf{N}$ dimana $n \geq k$. Langkah induksi juga kerap dirumuskan sebagai berikut:

$$P(n-1) \Longrightarrow P(n) \text{ untuk setiap } n > 0 \text{ atau } n > k,$$

dimana $n \in \mathbf{N}$.

Selain teorema 1, ada dua bentuk lain dari prinsip induksi yang sering digunakan. Bentuk alternatif pertama adalah sebagai berikut (notasi $\forall$ berarti "untuk setiap"):

Teorema 2 *Jika kita dapat buktikan bahwa:*

$$\begin{array}{ll} (i) & P(0), \text{ dan} \\ (ii) & \forall n \in \mathbf{N} : (\forall m \leq n : P(m)) \Longrightarrow P(n+1). \end{array}$$

Maka $\forall n \in \mathbf{N} : P(n)$ *berlaku.*

Prinsip induksi sebagaimana dalam teorema 2 disebut juga *strong induction principle*. Perbedaan antara kedua prinsip hanya terletak pada *step case*: dengan *strong induction* kita dapat menggunakan lebih banyak asumsi dalam membuktikan $P(n+1)$. Meskipun prinsip *strong induction* sepertinya menghasilkan mekanisme yang lebih ampuh, kita dapat buktikan bahwa sebenarnya kedua prinsip ekuivalen. Untuk menunjukkan bahwa $P(n)$ dapat dibuktikan menggunakan *strong induction* jika $P(n)$ dapat dibuktikan menggunakan induksi (teorema 1) sangat mudah: pada *step case*, kita cukup melakukan instansiasi $m = n$. Untuk membuktikan sebaliknya, kita perlu membuktikan bahwa *step case* induksi dapat ditransformasi menjadi menjadi *step case* untuk *strong induction*. Kita gunakan notasi:

$$Q(n) = \forall m \leq n : P(m).$$

Jadi kita perlu membuktikan $Q(n)$ dengan asumsi $P(n)$. Kita buktikan ini menggunakan induksi. Untuk *base case* cukup mudah karena

$$\begin{array}{rcl} Q(0) & = & \forall m \leq 0 : P(m) \\ & = & P(0). \end{array}$$

Untuk *step case* kita perlu buktikan:

$$Q(n) \Longrightarrow Q(n+1).$$

Karena $Q(n) = \forall m \leq n : P(m)$ dan $P(n) \Longrightarrow P(n+1)$, maka

$$Q(n) \Longrightarrow P(n+1).$$

Dari $Q(n)$ dan $P(n+1)$ kita dapatkan $Q(n+1)$ dan selesailah pembuktian kita.

Bentuk alternatif kedua dari prinsip induksi adalah prinsip *well-ordering* sebagai berikut:

Teorema 3 *Jika* M *adalah subset non-kosong dari* $\mathbf{N}$ *maka* M *mempunyai elemen terkecil (least element).*

Kita buktikan kontrapositif dari prinsip ini, yaitu jika M tidak mempunyai elemen terkecil maka M adalah himpunan kosong. Untuk membuktikan bahwa M adalah himpunan kosong, kita buktikan bahwa $n \notin M$ untuk setiap $n \in \mathbf{N}$. Kita buktikan ini menggunakan prinsip *strong induction*. Untuk *base case* sangat mudah karena jika $0 \in M$ maka 0 adalah elemen terkecil dalam M, jadi $0 \notin M$. Untuk *step case*, kita umpamakan $\forall m \leq n : m \notin M$ dan kita harus tunjukkan bahwa $(n+1) \notin M$. Jika $(n+1) \in M$ maka $n+1$ adalah elemen terkecil dalam M karena setiap bilangan natural yang lebih kecil dari $n+1$ tidak berada dalam M. Jadi $(n+1) \notin M$ dan selesailah pembuktian kita.

3.3 GCD

Kita mulai penjelasan gcd dengan teorema mengenai pembagian:

Teorema 4 (Pembagian) *Untuk setiap pasangan bilangan bulat a dan b dengan $b > 0$, terdapat pasangan unik bilangan bulat q dan r yang mematuhi persamaan:*

$$a = qb + r \text{ dengan } 0 \leq r < b. \tag{3.1}$$

Teorema ini dapat dibuktikan menggunakan prinsip *well ordering* (teorema 3). Untuk itu kita gunakan dua himpunan "standard" yaitu:

- $\mathbf{Z}$, himpunan dari semua bilangan bulat (*integers*).
- $\mathbf{N}$, himpunan dari semua bilangan bulat non-negatif (*natural numbers*).

Pertama, kita buat:

$$\begin{aligned} S &= \{a - nb | n \in \mathbf{Z}\} = \{a, a+b, a-b, a+2b, a-2b, \ldots\}, \\ S^+ &= S \cap \mathbf{N} = \{a \in S | a \geq 0\}. \end{aligned}$$

Jadi S merupakan himpunan semua bilangan bulat yang berbeda kelipatan b dari a. Karena sebagian dari elemen himpunan S adalah bilangan bulat non-negatif, S^+ merupakan subset non-kosong dari $\mathbf{N}$. Jadi kita dapat gunakan prinsip *well-ordering*, yang mengatakan bahwa S^+ mempunyai elemen terkecil, sebut saja r, yang berdasarkan definisi S, mempunyai bentuk $r = a - qb$ dengan q berupa bilangan bulat. Ini membuktikan bahwa untuk sepasang a dan b dengan $b > 0$, terdapat pasangan q dan r yang mematuhi persamaan

$$a = qb + r.$$

Karena $r \in S^+$, maka $0 \leq r$. Karena r adalah elemen terkecil S^+, maka $r < b$, sebab konsekuensi $r \geq b$ adalah S^+ mempunyai elemen $r - b$ yang lebih kecil dari r, sesuatu yang tidak mungkin apabila r adalah elemen terkecil. Jadi

$$0 \leq r < b.$$

Untuk menunjukkan bahwa pasangan q dan r unik, kita umpamakan bahwa

$$\begin{aligned} a &= qb + r \\ &= q'b + r' \end{aligned}$$

dengan

$$0 \leq r < b \text{ dan } 0 \leq r' < b,$$

jadi $r - r' = (q' - q)b$. Jika $q \neq q'$ maka $|q' - q| \geq 1$, yang berarti $|r - r'| \geq |b| = b$, sesuatu yang tidak mungkin karena $0 \leq r < b$ dan $0 \leq r' < b$ berarti perbedaan antara r dan r' lebih kecil dari b. Oleh karena itu $q' = q$ dan akibatnya $r' = r$.

Teorema mengenai pembagian diatas menjadi dasar dari konsep *residue* untuk bilangan bulat sebagai berikut. Membagi persamaan 3.1 dengan b, kita dapatkan:

$$\frac{a}{b} = q + \frac{r}{b} \text{ dengan } 0 \leq \frac{r}{b} < 1.$$

Kita bisa lihat bahwa q adalah bagian bulat dari a/b, jadi q merupakan bilangan bulat terbesar yang $\leq a/b$. Jadi q, yang juga disebut *quotient*, dapat dikalkulasi dengan mudah karena merupakan hasil pembagian dibulatkan kebawah. Setelah q didapat, r, yang juga disebut *remainder* atau *residue* dapat dikalkulasi menggunakan rumus $r = a - qb$.

Teorema diatas berlaku untuk $b > 0$. Bagaimana dengan $b < 0$? Untuk $b < 0$, teorema diatas berlaku untuk $-b$, jadi untuk setiap pasangan bilangan bulat a dan b dengan $b < 0$ terdapat pasangan unik bilangan bulat q' dan r yang mematuhi persamaan:

$$a = q'(-b) + r \text{ dengan } 0 \leq r < -b.$$

Dengan $q = -q'$ kita dapatkan $a = qb + r$. Jadi teorema bisa direvisi menjadi:

Teorema 5 (Pembagian) *Untuk setiap pasangan bilangan bulat a dan b dengan $b \neq 0$ terdapat pasangan unik q dan r yang mematuhi persamaan:*

$$a = qb + r \text{ dengan } 0 \leq r < |b|. \tag{3.2}$$

Untuk $b < 0$,

$$\frac{a}{b} = q + \frac{r}{b} \text{ dan } 0 \geq \frac{r}{b} > -1.$$

Jadi untuk $b < 0$, q merupakan bilangan bulat terkecil $\geq a/b$ (q dibulatkan keatas).

Untuk setiap pasangan bilangan bulat a dan b, jika terdapat bilangan bulat q sehingga $a = qb$, maka b membagi a, dan b disebut pembagi (*divisor* atau faktor) dari a dengan notasi $b|a$. Notasi $b \not| a$ digunakan jika b bukan pembagi a.

Definisi 5 (GCD) *Jika $d|a$ dan $d|b$ maka d adalah pembagi persekutuan (common divisor) dari a dan b. Untuk setiap pasangan bilangan bulat a dan b kecuali jika $a = b = 0$, pembagi persekutuan terbesar (greatest common divisor atau* gcd*) dari a dan b adalah bilangan bulat unik d dimana:*

1. *d merupakan pembagi persekutuan dari a dan b,*
2. *jika c merupakan pembagi persekutuan dari a dan b, maka $c \leq d$.*

Dalam beberapa cabang matematika yang lebih abstrak, syarat 2 diubah dengan $c|d$ menggantikan $c \leq d$ sehingga d dan $-d$ keduanya merupakan $\gcd(a, b)$. Menurut teori abstrak mengenai struktur *ring*, jika d merupakan $\gcd(a, b)$ dan u adalah sebuah *unit*[1] dalam struktur *ring*, maka ud juga merupakan $\gcd(a, b)$, jadi gcd belum tentu unik jadi bukan merupakan fungsi. Untuk bilangan bulat, kita gunakan versi diatas agar gcd unik dan positif. Kita akan gunakan gcd versi abstrak dalam pembahasan beberapa konsep dalam teori *ring*.

3.4 Algoritma Euclid

Satu cara untuk mendapatkan $\gcd(a, b)$ adalah dengan membuat daftar semua faktor dari a, membuat daftar semua faktor dari b, dan kemudian mencari faktor terbesar yang ada dalam kedua daftar. Akan tetapi, untuk bilangan yang sangat besar, membuat daftar faktor bukanlah sesuatu yang mudah. Ada cara yang jauh lebih efisien untuk mendapatkan $\gcd(a, b)$ yaitu dengan menggunakan algoritma Euclid (*Euclidean algorithm*), yang seperti halnya dengan *Chinese Remainder Theorem* merupakan algoritma penting yang berusia lebih dari 2000 tahun.

Teorema yang digunakan sebagai dasar dari algoritma Euclid adalah sebagai berikut:

Teorema 6 (Algoritma Euclid)

$$\textit{Jika } a = qb + r \textit{ maka } \gcd(a, b) = \gcd(b, r).$$

Untuk meyakinkan kita sendiri bahwa teorema diatas benar, kita tahu bahwa jika $a = qb+r$ maka setiap pembagi persekutuan b dan r juga membagi $qb+r = a$. Juga, karena $r = a - qb$, setiap pembagi persekutuan a dan b juga membagi r. Akibatnya setiap pembagi persekutuan a dan b juga merupakan pembagi persekutuan b dan r, dan setiap pembagi persekutuan b dan r juga merupakan pembagi persekutuan a dan b, jadi $\gcd(a, b) = \gcd(b, r)$.

Algoritma Euclid menggunakan rumus diatas secara berulang untuk mendapatkan gcd, yaitu dengan memperkecil kedua bilangan yang dijadikan patokan untuk gcd setiap kali mengulang, tanpa merubah nilai gcd itu sendiri.

[1] Untuk struktur *ring* bilangan bulat, ada dua *unit*: 1 dan -1.

Hasil dari komputasi gcd didapat saat kedua patokan untuk gcd tidak dapat diperkecil lagi.

Untuk melakukan komputasi $d = \gcd(a, b)$, pertama dilakukan *preprocessing* sebagai berikut:

1. Jika $a = 0$ maka $d = |b|$ dan jika $b = 0$ maka $d = |a|$ (gcd tidak dapat dikomputasi jika $a = b = 0$).

2. Karena $\gcd(a, b) = \gcd(-a, b) = \gcd(a, -b) = \gcd(-a, -b)$, kita dapat mentransformasi komputasi menjadi $d = \gcd(|a|, |b|)$, jadi kedua bilangan menjadi positif.

3. Karena $\gcd(a, b) = \gcd(b, a)$, kita dapat saling tukar a dan b jika $a < b$, dengan hasil $a \geq b$.

4. Jika $a = b$ maka $d = a$.

Setelah *preprocessing*, jika $a \neq 0$, $b \neq 0$, dan $a \neq b$, maka komputasi sudah dirubah menjadi $d = \gcd(a, b)$ dengan $a > b > 0$.

Langkah berikutnya adalah membagi a dengan b menggunakan algoritma pembagian, mendapatkan *quotient* q_1 dan *residue* r_1 ($\gcd(a, b) = \gcd(b, r_1)$ berdasarkan teorema 6):

$$a = q_1 b + r_1 \text{ dengan } 0 \leq r_1 < b.$$

Jika $r_1 = 0$ maka b membagi a, jadi $d = b$ dan kita selesai. Jika $r_1 \neq 0$ kita bagi b dengan r_1 mendapatkan *quotient* q_2 dan *residue* r_2 ($\gcd(a, b) = \gcd(r_1, r_2)$):

$$b = q_2 r_1 + r_2 \text{ dengan } 0 \leq r_2 < r_1.$$

Jika $r_2 = 0$ maka r_1 membagi b, jadi $d = r_1$ dan kita selesai. Jika $r_2 \neq 0$ kita teruskan ($\gcd(a, b) = \gcd(r_2, r_3)$):

$$r_1 = q_3 r_2 + r_3 \text{ dengan } 0 \leq r_3 < r_2,$$

dan seterusnya jika $r_3 \neq 0$, sampai kita dapatkan $r_n = 0$ (sampai dengan langkah ini kita mengetahui bahwa $\gcd(a, b) = \gcd(r_{n-2}, r_{n-1}) = \gcd(r_{n-1}, r_n)$):

$$r_{n-2} = q_n r_{n-1} + r_n \text{ dengan } r_n = 0.$$

Dengan $r_n = 0$, kita tahu bahwa r_{n-1} membagi r_{n-2}, jadi $d = r_{n-1}$ dan kita selesai. Tidak terlalu sulit untuk melihat bahwa algoritma Euclid akan berhenti pada suatu r_n dengan $n \geq 0$, karena tidak mungkin terdapat deretan

$$r_1 > r_2 > r_3 > \ldots$$

yang tidak berhenti (dimana setiap r_i merupakan bilangan bulat positif).

Sebagai contoh, mari kita kalkulasi $\gcd(1485, 1745) = \gcd(1745, 1485)$:

$$\begin{aligned} 1745 &= 1 \cdot 1485 + 260 \\ 1485 &= 5 \cdot 260 + 185 \\ 260 &= 1 \cdot 185 + 75 \\ 185 &= 2 \cdot 75 + 35 \\ 75 &= 2 \cdot 35 + 5 \\ 35 &= 7 \cdot 5 + 0. \end{aligned}$$

Jadi $\gcd(1485, 1745) = 5$.

Sebagai konsekuensi dari algoritma Euclid, kita dapatkan $gcd(a, b)$ sebagai kombinasi linear dari a dan b:

Teorema 7 *Untuk setiap pasangan bilangan bulat a dan b, kecuali $a = b = 0$, terdapat pasangan bilangan bulat u dan v yang mematuhi:*

$$\gcd(a, b) = au + bv.$$

Pasangan u dan v dapat kita cari menggunakan persamaan-persamaan yang digunakan dalam algoritma Euclid. Menggunakan contoh diatas:

$$\begin{aligned} 5 &= 75 - 2 \cdot 35 \\ &= 75 - 2 \cdot (185 - 2 \cdot 75) \\ &= -2 \cdot 185 + 5 \cdot 75 \\ &= -2 \cdot 185 + 5 \cdot (260 - 185) \\ &= 5 \cdot 260 - 7 \cdot 185 \\ &= 5 \cdot 260 - 7 \cdot (1485 - 5 \cdot 260) \\ &= -7 \cdot 1485 + 40 \cdot 260 \\ &= -7 \cdot 1485 + 40 \cdot (1745 - 1485) \\ &= -47 \cdot 1485 + 40 \cdot 1745. \end{aligned}$$

Kita dapatkan pasangan $u = -47$ dan $v = 40$ sebagai solusi. Pasangan u dan v tidak unik karena kita bisa tambahkan atau kurangkan kombinasi linear a dan b yang mempunyai nilai 0 tanpa mempengaruhi nilai d. Jadi jika $u'a + v'b = 0$ dengan $u' \neq 0$ atau $v' \neq 0$ (contohnya $u' = b$ dan $v' = -a$), kita dapatkan $d = (u + u')a + (v + v')b$ dengan $u + u' \neq u$ atau $v + v' \neq v$ (pasangan $u + u'$ dan $v + v'$ tidak sama dengan pasangan u dan v). Sebagai contoh:

$$5 = 1698 \cdot 1485 + (-1445) \cdot 1745$$

Algoritma Euclid dapat direvisi agar sekaligus mendapatkan pasangan u dan v disamping mendapatkan gcd d. Algoritma yang sudah direvisi disebut juga *extended Euclidean algorithm*. Untuk $a, b > 0$, *extended Euclidean algorithm* mencari $d = \gcd(a, b), u$ dan v dengan $d = au + bv$. Agar yakin bahwa

algoritma benar, kita gunakan konsep *loop invariant*, yaitu suatu proposisi yang berlaku pada setiap putaran. *Loop invariant* yang digunakan adalah

$$\gcd(a,b) = \gcd(A,B), A = au + bv \text{ dan } B = as + bt.$$

Langkah-langkah untuk algoritma adalah sebagai berikut:

1. $A \leftarrow a, B \leftarrow b, u \leftarrow 1, v \leftarrow 0, s \leftarrow 0, t \leftarrow 1$.
2. $q \leftarrow A$ div B.
3. $r \leftarrow A - qB$.
4. $A \leftarrow B, B \leftarrow r, U \leftarrow u, V \leftarrow v$.
5. $u \leftarrow s, v \leftarrow t, s \leftarrow U - qs, t \leftarrow V - qt$.
6. Jika $B \neq 0$ kita ulangi dari langkah 2.
7. $d \leftarrow A$ dan kita selesai.

Operasi div adalah operasi pembagian dibulatkan kebawah. Setelah langkah 1, *loop invariant* berlaku karena $A = a, B = b, u = 1, v = 0, s = 0, t = 1$ berarti

$$\begin{aligned} \gcd(a,b) &= \gcd(A,B), \\ au + bv &= a \cdot 1 + b \cdot 0 = a = A, \text{ dan} \\ as + bt &= a \cdot 0 + b \cdot 1 = b = B. \end{aligned}$$

Kita harus tunjukkan bahwa jika *loop invariant* berlaku pada langkah 2, *loop invariant* juga berlaku pada langkah 6. Untuk keperluan ini, kita gunakan notasi A', B', u', v', s', t' sebagai nilai A, B, u, v, s, t pada saat langkah 2 dimulai. Langkah 2 dan 3 mendapatkan *quotient* q dan *residue* r, jadi $A' = qB' + r$. Menggunakan teorema 6, kita dapatkan

$$\gcd(a,b) = \gcd(A',B') = \gcd(B',r).$$

Langkah 4 membuat $A = B'$ dan $B = r$, jadi

$$\gcd(a,b) = \gcd(B',r) = \gcd(A,B)$$

setelah langkah 5 (langkah 5 tidak mengubah A dan B). Jadi $\gcd(a,b) = \gcd(A,B)$ berlaku pada langkah 6.

$$\begin{aligned} A &= B' \text{ (dari langkah 4)} \\ &= as' + bt' \text{ (dari } \textit{loop invariant} \text{ pada langkah 2)} \\ &= au + bv \text{ (karena langkah 5 membuat } u = s' \text{ dan } v = t'). \end{aligned}$$

Jadi $au + bv = A$ berlaku pada langkah 6.

$$\begin{aligned} B &= r \text{ (dari langkah 4)} \\ &= A' - qB' \\ &= au' + bv' - qB' \text{ (dari } \textit{loop invariant} \text{ pada langkah 2)} \\ &= au' + bv' - q(as' + bt') \text{ (dari } \textit{loop invariant} \text{ pada langkah 2)} \\ &= a(u' - qs') + b(v' - qt') \\ &= as + bt \text{ (langkah 5 membuat } s = u' - qs' \text{ dan } t = v' - qt'). \end{aligned}$$

Jadi $as + bt = B$ berlaku pada langkah 6. Akibatnya seluruh *loop invariant* berlaku pada langkah 6. Pada langkah 7, kita dapatkan $B = 0$ jadi

$$\gcd(a, b) = \gcd(A, B) = \gcd(A, 0) = A = d$$

dan

$$d = A = au + bv.$$

Jadi algoritma benar mengkalkulasi $d = \gcd(a, b)$ dan mendapatkan u, v dengan $d = au + bv$.

Teorema 8 *Untuk setiap pasangan bilangan bulat a dan b kecuali $a = b = 0$ dengan $d = \gcd(a, b)$ dan bilangan bulat c, persamaan:*

$$c = ax + by \text{ dengan } (x, y \in \mathbf{Z})$$

mempunyai solusi jika dan hanya jika ($\Longleftrightarrow$) c merupakan kelipatan d.

Untuk membuktikan bahwa c harus merupakan kelipatan d, kita gunakan fakta bahwa d membagi a dan b, alhasil d membagi $c = ax + by$. Untuk membuktikan bahwa setiap kelipatan d (sebut saja $c = de$ dengan e bilangan bulat apa saja) merupakan solusi, teorema 7 memberikan $d = au + bv$ untuk sepasang bilangan bulat u dan v, akibatnya $c = de = aue + bve$, jadi dengan $x = ue$ dan $y = ve$ kita dapatkan $c = ax + by$.

Definisi 6 (Koprima) *Sepasang bilangan bulat a dan b disebut koprima (coprime atau relatively prime) jika* $\gcd(a, b) = 1$.

Teorema 9 *Sepasang bilangan bulat a dan b koprima jika dan hanya jika ($\Longleftrightarrow$) ada pasangan bilangan bulat x dan y yang mematuhi persamaan:*

$$ax + by = 1.$$

Untuk membuktikan bahwa ada pasangan bilangan bulat x dan y dimana persamaan $ax+by = 1$ berlaku jika a dan b koprima, cukup menggunakan teorema 7 dengan $u = x$ dan $v = y$. Untuk membuktikan bahwa jika $ax+by = 1$ berarti a dan b koprima, kita gunakan teorema 8 dengan $c = 1$, jadi karena d harus membagi c, maka $d = 1$, yang berarti a dan b koprima.

3.5 Aritmatika Modular

Saat membahas analisa statistik (lihat 2.3.2), contoh *Caesar cipher* digunakan dengan enkripsi dan dekripsi yang rumusnya bersifat aritmatika. Aritmatika adalah matematika pertambahan dan perkalian dengan kemungkinan operasi *inverse* (pembalikan). Untuk aritmatika bilangan bulat, hanya 1 dan -1 yang mempunyai *inverse* perkalian[2], jadi struktur aritmatika bilangan bulat bukan *field* tetapi *ring*. *Domain* aritmatika bilangan bulat bersifat *infinite* (besarnya *domain* bukan merupakan bilangan bulat).

Lain dengan aritmatika bilangan bulat, aritmatika bilangan rasional (*rational numbers*) dan aritmatika bilangan nyata (*real numbers*) mempunyai struktur *field* dimana setiap bilangan kecuali 0 mempunyai *inverse* (setiap elemen yang mempunyai *inverse* disebut *unit*). Dalam struktur *field*, konsep gcd tidak ada artinya karena setiap bilangan kecuali 0 adalah pembagi untuk semua bilangan.

Aritmatika yang banyak digunakan dalam kriptografi adalah apa yang disebut aritmatika modular (*modular arithmetic*). Dalam aritmatika modular, *domain* yang digunakan adalah *subset* dari bilangan bulat dan bersifat *finite* (terbatas, besarnya *domain* merupakan bilangan bulat). Setiap bilangan mempunyai *inverse* pertambahan, dan jika setiap bilangan kecuali 0 mempunyai *inverse* perkalian maka struktur aritmatika disebut *finite field*. Digunakannya aritmatika modular dalam kriptografi adalah karena adanya *inverse* perkalian (terutama jika struktur berupa *field*) dan *domain* yang bersifat *finite*.

Karena *finite field* juga berupa *field*, konsep gcd tidak ada artinya dalam struktur *finite field*. Tetapi gcd dengan bilangan bulat (yang mempunyai struktur *ring*) banyak digunakan dalam membahas struktur *finite field*.

Domain dari aritmatika modular adalah $\{0, 1, 2, \ldots, n-1\}$, dimana n adalah besarnya *domain*. Aritmatika disebut aritmatika modulo n, dengan pertambahan dan perkalian seperti aritmatika biasa jika menghasilkan bilangan yang termasuk dalam *domain*. Jika hasil merupakan bilangan diluar *domain*, maka bilangan harus dikurangi dengan kelipatan n sampai menghasilkan bilangan dalam *domain*.

Tabel 3.1 menunjukkan contoh aritmatika modulo 7. Untuk $5 + 5$, hasilnya adalah 3 karena 10 dikurangi 7 menghasilkan 3.

Proses pengurangan kelipatan n dapat direpresentasikan dengan operasi mod. Menggunakan persamaan 3.1 dari teorema pembagian dengan $b = n$:

$$a = nq + r,$$

operasi mod dapat didefinisikan sebagai berikut:

[2]Selanjutnya jika tidak disebutkan pertambahan atau perkalian maka *inverse* berarti *inverse* perkalian.

Ekspressi	Hasil
$2+3$	5
$5+5$	3
$5 \cdot 6$	2
-3	4
4^{-1}	2

Tabel 3.1: Contoh aritmatika modulo 7

Definisi 7 (mod)

$$a \bmod n = r = a - nq,$$

dengan kata lain a mod n adalah *remainder* atau *residue* dari pembagian a oleh n.

Jadi operasi pertambahan dan perkalian modulo n dapat dipandang sebagai operasi aritmatika bilangan bulat yang dilanjutkan dengan operasi mod pada hasil operasi bilangan bulat. Rumus untuk pertambahan $x + y$ menjadi:

$$x + y = x + y \bmod n. \tag{3.3}$$

dimana operasi disebelah kanan persamaan menggunakan aritmatika bilangan bulat. Rumus untuk perkalian $x \cdot y$ menjadi:

$$x \cdot y = xy \bmod n. \tag{3.4}$$

Dengan menggunakan rumus perkalian untuk aritmatika modulo 7, $5 \cdot 6$ menghasilkan $30 \bmod 7 = 2$ ($a = xy = 30$, $n = 7$, $q = 4$).

Definisi untuk *inverse* pertambahan $-b$ adalah:

$$b + -b = 0.$$

Jadi $-b$ adalah bilangan bulat yang mematuhi persamaan:

$$(b + -b) \bmod n = 0 \text{ dan } 0 \leq -b < n.$$

Karena $0 \leq b < n$, rumus untuk $-b$ menjadi:

$$-b = n - b. \tag{3.5}$$

Jadi -3 menghasilkan $7 - 3 = 4$.

Definisi untuk *inverse* perkalian b^{-1} adalah:

$$b \cdot b^{-1} = 1. \tag{3.6}$$

Jadi b^{-1} adalah bilangan bulat yang mematuhi persamaan:

$$(b \cdot b^{-1}) \bmod n = 1 \text{ dengan } 0 \leq b^{-1} < n$$

Jadi 4^{-1} menghasilkan 2 karena $4 \cdot 2$ menghasilkan $8 \bmod 7 = 1$ ($a = 4 \cdot 2 = 8$, $n = 7$, $q = 1$). Kita akan melihat bagaimana kita dapat mengkalkulasi *inverse* perkalian. Kita definisikan dahulu konsep *congruent modulo n*.

Definisi 8 (Congruence) *Untuk setiap pasangan bilangan bulat a dan b, a congruent dengan b modulo n, dengan notasi*

$$a \equiv b \pmod{n},$$

jika a dan b mempunyai residue yang sama jika dibagi oleh n.

Menggunakan teorema 4 untuk pembagian, ada q' dan q'' dengan:

$$a = q'n + r, \text{ dan}$$

$$b = q''n + r$$

(kedua persamaan menggunakan r karena a dan b mempunyai *residue* yang sama). Jadi jika $a \equiv b \pmod{n}$ maka ada bilangan bulat $q = q' - q''$ yang mematuhi persamaan $a - b = qn$ (perbedaan antara a dan b adalah kelipatan n).

Konsep yang sering digunakan untuk menjelaskan aritmatika modular adalah konsep *congruence classes*. Relasi *congruent modulo n* adalah relasi ekuivalen yang mempartisi himpunan dari semua bilangan bulat ($\mathbf{Z}$) menjadi n kelas ekuivalen yang disebut juga *congruence classes*:

$$\begin{aligned}
[0] &= \{\ldots, -2n, -n, 0, n, 2n, \ldots\}, \\
[1] &= \{\ldots, -2n+1, -n+1, 1, n+1, 2n+1, \ldots\}, \\
[2] &= \{\ldots, -2n+2, -n+2, 2, n+2, 2n+2, \ldots\}, \\
&\ldots \\
[n-1] &= \{\ldots, -n-1, -1, n-1, 2n-1, 3n-1, \ldots\}.
\end{aligned}$$

Rumus untuk *congruence class* $[i]$ dengan modulus n adalah:

$$[i] = \{j \in \mathbf{Z} | i \equiv j \pmod{n}\} = \{jn + i | j \in \mathbf{Z}\}. \tag{3.7}$$

Dengan modulus n, hanya ada n kelas, tidak ada kelas lain. Sebagai contoh

$$[n] = \{\ldots, -n, 0, n, 2n, 3n, \ldots\} = [0].$$

Secara umum

$$[a] = [b] \Longleftrightarrow a \equiv b \pmod{n}. \tag{3.8}$$

Himpunan *congruence classes* mempunyai struktur *quotient ring* (akan dibahas di bab 5), dengan notasi $\mathbf{Z}/n\mathbf{Z}$ untuk bilangan bulat n. Sebagai contoh, untuk $n = 7$ ada 7 kelas:

$$\begin{aligned}
[0] &= \{\ldots, -14, -7, 0, 7, 14, \ldots\},\\
[1] &= \{\ldots, -13, -6, 1, 8, 15, \ldots\},\\
[2] &= \{\ldots, -12, -5, 2, 9, 16, \ldots\},\\
[3] &= \{\ldots, -11, -4, 3, 10, 17, \ldots\},\\
[4] &= \{\ldots, -10, -3, 4, 11, 18, \ldots\},\\
[5] &= \{\ldots, -9, -2, 5, 12, 19, \ldots\},\\
[6] &= \{\ldots, -8, -1, 6, 13, 20, \ldots\}.
\end{aligned}$$

Setiap elemen dalam kelas adalah representatif kelas, jadi setiap bilangan yang berbeda kelipatan 7 dari 2 (contohnya -5, 2 dan 9) merupakan representatif dari [2].

Aritmatika dapat didefinisikan terhadap kelas. Aritmatika dilakukan dahulu terhadap representatif kelas (elemen mana saja dalam kelas dapat digunakan). Hasil aritmatika bilangan kemudian digunakan untuk menentukan kelas yang merupakan hasil aritmatika kelas. Sebagai contoh, untuk $[2] + [3]$, kita dapat tambahkan -5 (yang merupakan representatif [2]) dengan 3 (yang merupakan representatif [3]) untuk mendapatkan -2, yang merupakan representatif dari [5]. Jadi $[2] + [3] = [5]$. Secara formal, rumus untuk pertambahan adalah:

$$[a] + [b] = [a + b], \tag{3.9}$$

rumus untuk *inverse* pertambahan adalah:

$$-[a] = [-a], \tag{3.10}$$

dan rumus untuk perkalian adalah:

$$[a] \cdot [b] = [a \cdot b]. \tag{3.11}$$

Tidak semua kelas mempunyai *inverse* perkalian, kita tunda pembahasan rumus untuk mencari *inverse* perkalian. Untuk meyakinkan bahwa operasi telah didefinisikan dengan baik (*well-defined*), kita gunakan teorema:

Teorema 10 *Untuk modulus $n > 1$, jika $a' \equiv a$ dan $b' \equiv b$, maka $a' + b' \equiv a + b$, $-a' \equiv -a$ dan $a' \cdot b' \equiv a \cdot b$.*

Pembuktian teorema adalah sebagai berikut: jika $a' \equiv a \pmod{n}$ maka terdapat bilangan bulat k sehingga $a' = a + kn$. Demikian juga untuk $b' \equiv b \pmod{n}$ terdapat l sehingga $b' = b + ln$. Jadi

$$\begin{aligned}
a' + b' = a + b + (k + l)n &\equiv a + b \pmod{n},\\
-a' = -a - kn &\equiv -a \pmod{n},\\
a' \cdot b' = a \cdot b + (al + bk + kln)n &\equiv a \cdot b \pmod{n}.
\end{aligned}$$

Teorema 10 menyatakan bahwa operasi terhadap kelas tidak tergantung representatif kelas yang dipilih, jadi operasi telah didefinisikan dengan baik.

Selanjutnya tidak terlalu sukar untuk menunjukkan bahwa aritmatika *congruence classes* mempunyai struktur aljabar *ring* dengan elemen [0] dan [1]. *Closure* untuk + dan · didapat karena bilangan apapun yang dihasilkan aritmatika bilangan bulat akan menghasilkan satu diantara *congruence classes* yang ada.
Associativity untuk +:

$$\begin{aligned}[a] + ([b] + [c]) &= [a] + [b + c] \\ &= [a + (b + c)] \\ &= [(a + b) + c] \\ &= [a + b] + [c] \\ &= ([a] + [b]) + [c].\end{aligned}$$

Identity untuk +:

$$[a] + [0] = [a + 0] = [a].$$

Commutativity untuk +:

$$[a] + [b] = [a + b] = [b + a] = [b] + [a].$$

Inverse untuk +:

$$[a] + (-[a]) = [a] + [-a] = [a + (-a)] = [0].$$

Associativity untuk ·:

$$\begin{aligned}[a] \cdot ([b] \cdot [c]) &= [a] \cdot [b \cdot c] \\ &= [a \cdot (b \cdot c)] \\ &= [(a \cdot b) \cdot c] \\ &= [a \cdot b] \cdot [c] \\ &= ([a] \cdot [b]) \cdot [c].\end{aligned}$$

Identity untuk ·:

$$[a] \cdot [1] = [a \cdot 1] = [a].$$

Commutativity untuk ·:

$$[a] \cdot [b] = [a \cdot b] = [b \cdot a] = [b] \cdot [a].$$

Distributivity:

$$\begin{aligned}
[a] \cdot ([b] + [c]) &= [a] \cdot [b + c] \\
&= [a \cdot (b + c)] \\
&= [(a \cdot b) + (a \cdot c)] \\
&= [a \cdot b] + [a \cdot c] \\
&= ([a] \cdot [b]) + ([a] \cdot [c]).
\end{aligned}$$

Kembali ke aritmatika modular, kita dapat menggunakan konsep aritmatika *congruence classes* untuk menjelaskannya. Karena aritmatika modular tidak tergantung pada representatif kelas yang dipilih, untuk setiap kelas kita dapat memilih representatif $0 \leq r < n$, yaitu *residue* modulo n. Jadi aritmatika modular dapat dianggap sebagai aritmatika *congruence classes* dengan *residue* modulo n digunakan sebagai representatif setiap kelas.

Sebagai contoh, untuk mencari hasil aritmatika modulo 7, kita bisa cari dahulu hasil aritmatika bilangan bulat, kemudian cari kelas dari hasil aritmatika bilangan bulat, dan ahirnya cari bilangan r dalam kelas yang sama yang mematuhi $0 \leq r < 7$ (r adalah *residue* modulo 7 representatif kelas). Contoh kongkrit, untuk *inverse* pertambahan 3, hasil aritmatika bilangan bulat adalah -3, dan hasil pencarian r dalam kelas yang berisi -3 dengan $0 \leq r < 7$ menghasilkan $r = 4$. (Kita dapat juga mengkalkulasi $r = -3 \bmod 7 = 4$.)

Teorema berikut kita gunakan sebagai dasar cara mengkalkulasi *inverse* perkalian dalam aritmatika modular.

Teorema 11 (Inverse) *Suatu bilangan a mempunyai inverse modulo n jika dan hanya jika ($\Longleftrightarrow$)* $\gcd(a, n) = 1$.

Untuk membuktikan teorema ini, definisi *inverse* mengatakan bahwa jika a mempunyai *inverse* (sebut saja x) berarti $ax \equiv 1 \pmod{n}$. Jadi ada bilangan bulat q yang mematuhi persamaan $ax - 1 = qn$ atau $ax - nq = 1$. Menggunakan teorema 9 dengan $b = n$ dan $y = -q$ berarti a koprima dengan n jadi $\gcd(a, n) = 1$. Sebaliknya jika $\gcd(a, n) = 1$, maka berdasarkan teorema 7, ada pasangan u dan v yang mematuhi $1 = au + nv$, jadi $au \equiv 1 \pmod{n}$, yang berarti a mempunyai *inverse* yaitu u. *Extended Euclidean algorithm* (lihat 3.4) dapat digunakan untuk mendapatkan *inverse* u.

Tentunya jika n merupakan bilangan prima, untuk setiap $a \not\equiv 0 \pmod{n}$ (termasuk $0 < a < n$) kita dapatkan $\gcd(a, n) = 1$, sehingga setiap $a \not\equiv 0 \pmod{n}$ mempunyai *inverse*. Jadi aritmatika modulo bilangan prima n mempunyai struktur *finite field*.

3.6 Ringkasan

Bab ini dimulai dengan pembahasan struktur-struktur aljabar, antara lain *group*, *monoid*, *ring* dan *field*. Struktur-struktur tersebut banyak digunakan dalam matematika untuk kriptografi. Prinsip induksi dibahas karena diperlukan untuk pembuktian berbagai teorema. Konsep gcd dan kalkulasinya menggunakan algoritma *Euclid* juga dibahas dengan rinci. Konsep aritmatika modular dijelaskan menggunakan struktur aljabar dan konsep gcd. Antara lain, *inverse* dalam aritmatika modular dapat dikalkulasi menggunakan *extended Euclidean algorithm*.

Bab 4

Kriptografi Simetris Sederhana

4.1 Enkripsi Affine

Enkripsi yang digunakan Julius Caesar (*Caesar cipher*) menggunakan transformasi yang sederhana yaitu *shift transformation*. Pembahasan analisa statistik (lihat 2.3.2) menunjukkan bahwa *shift transformation* sangat rentan terhadap analisa frekuensi. Untuk mencoba mempersulit analisa frekuensi, enkripsi *affine* menggunakan *affine transformation*, dengan rumus:

$$C \equiv aP + b \pmod{n} \qquad \text{untuk enkripsi,} \tag{4.1}$$

dan

$$P \equiv a^{-1}C - a^{-1}b \pmod{n} \qquad \text{untuk dekripsi.} \tag{4.2}$$

Jadi kunci untuk enkripsi *affine* terdiri dari dua parameter: a dan b. Agar a mempunyai *inverse* a^{-1}, a harus mematuhi $\gcd(a, n) = 1$ (lihat teorema 11).

Sebagai contoh dari enkripsi *affine*, mari kita ganti *shift transformation* yang digunakan Julius Caesar dengan *affine transformation* menggunakan parameter $a = 7$, $b = 12$. Rumus untuk enkripsi menjadi:

$$C \equiv 7P + 12 \pmod{26}. \tag{4.3}$$

Nilai parameter $a = 7$ dapat digunakan karena $\gcd(7, 26) = 1$, jadi a mempunyai *inverse* a^{-1} dengan nilai $7^{-1} \equiv 15 \pmod{26}$. Oleh karena itu, rumus dekripsi menjadi:

$$P \equiv 15C - 15 \cdot 12 \equiv 15C + 2 \pmod{26}. \tag{4.4}$$

Dengan rumus 4.3 sebagai rumus enkripsi, huruf "a" yang mempunyai kode 0 mempunyai kode acak $7 \cdot 0 + 12 \equiv 12 \pmod{26}$ yang merupakan kode untuk huruf "m"; huruf "b" yang mempunyai kode 1 mempunyai kode acak $7 \cdot 1 + 12 \equiv 19 \pmod{26}$ yang merupakan kode untuk huruf "t"; dan seterusnya. Tabel 4.1 menunjukkan tabel lengkap untuk enkripsi *affine* dengan parameter $a = 7$, $b = 12$ dan $n = 26$.

Huruf Asli	Kode Asli	Kode Acak	Huruf Acak
a	0	12	m
b	1	19	t
c	2	0	a
d	3	7	h
e	4	14	o
f	5	21	v
g	6	2	c
h	7	9	j
i	8	16	q
j	9	23	x
k	10	4	e
l	11	11	l
m	12	18	s
n	13	25	z
o	14	6	g
p	15	13	n
q	16	20	u
r	17	1	b
s	18	8	i
t	19	15	p
u	20	22	w
v	21	3	d
w	22	10	k
x	23	17	r
y	24	24	y
z	25	5	f

Tabel 4.1: Tabel untuk contoh enkripsi affine

Tabel untuk enkripsi *affine transformation* menunjukkan bahwa sepasang huruf asli yang berurutan, sebagai contoh "c" dan "d", huruf acaknya ("a" dan "h") berjarak 7 $\pmod{26}$. Ini menunjukkan mengapa $\gcd(a, n) = 1$ diperlukan untuk enkripsi *affine*. Jika $\gcd(a, n) = 1$, seperti halnya dengan $a = 7$ dan $n = 26$, menambahkan $a \pmod{n}$ secara berulang akan mengembalikan

kita ke bilangan semula (sebut saja x) setelah n kali dan sedikitnya n kali penambahan:

$$x + an \equiv x \pmod{n} \text{ dan } x + an' \not\equiv x \pmod{n} \text{ untuk } 0 < n' < n.$$

Akibatnya semua bilangan bulat $0 \leq y < n$ akan "dikunjungi" setelah menambahkan a secara berulang sebanyak n kali. Jadi banyaknya kode acak yang digunakan sama dengan banyaknya kode asli, oleh sebab itu setiap kode asli mempunyai kode acak sendiri. Jadi setiap kode acak dapat didekripsi dengan unik.

Akan tetapi, jika $d = \gcd(a, n) \neq 1$, maka:

$$x + a(n/d) \equiv x \pmod{n} \text{ dan } x + an' \not\equiv x \pmod{n} \text{ untuk } 0 < n' < n/d,$$

jadi hanya ada n/d bilangan yang dikunjungi, tidak semua bilangan y dengan $0 \leq y < n$ dikunjungi, jadi tidak semua kode acak digunakan. Akibatnya setiap kode acak yang digunakan merupakan kode acak untuk lebih dari satu kode asli, tepatnya sebanyak d kode asli mempunyai kode acak yang sama. Jadi kode acak tidak dapat didekripsi dengan unik (penjelasannya secara matematis, ini disebabkan a tidak mempunyai *inverse*). Sebagai contoh, jika $a = 8$, $b = 12$ dan $n = 26$, maka setiap kode acak mempunyai $d = \gcd(8, 26) = 2$ kode asli, kode acak 12 ("m") mempunyai dua kode asli: 0 ("a") dan 13 ("n"). Jadi huruf "m" sebagai huruf acak tidak dapat didekripsi dengan unik. Untuk $a = 8$, $b = 12$ dan $n = 26$, hanya bilangan genap yang lebih kecil dari 26 digunakan sebagai kode acak. Bilangan ganjil tidak dikunjungi, jadi tidak digunakan sebagai kode acak. Akibatnya hanya ada 13 kode acak yang digunakan.

Kembali ke enkripsi *affine transformation* dengan parameter $a = 7$, $b = 12$ dan $n = 26$. Mari kita coba lakukan analisa frekuensi terhadap naskah acak yang kita enkripsi dengan *affine transformation* dan kita bandingkan dengan analisa sebelumnya terhadap *shift transformation* (lihat 2.3.2).

Naskah Asli	`Jangan rahasiakan pesan ini!`
Naskah Acak	`Xmzcmz bmjmiqmemz noimz qzq!`

Tabel 4.2: Enkripsi dengan *affine transformation*

Dengan *shift transformation*, untuk setiap percobaan kita mencari satu parameter kunci menggunakan satu persamaan, jadi untuk setiap percobaan kita pasangkan satu kode asli dengan satu kode acak yang sesuai berdasarkan statistik dari pengamatan empiris. Strategi pencarian adalah menggunakan pasangan dimana karakter aslinya mempunyai statistik penggunaan terbesar.

Karena *affine transformation* menggunakan dua parameter, setiap percobaan kita harus mencari kedua parameter sedikitnya menggunakan dua persamaan dengan dua pasangan. Strategi pencarian adalah dengan mencoba dua

pasangan dimana dua karakter aslinya merupakan dua karakter dengan statistik penggunaan terbesar.

Untuk contoh diatas, dengan perumpamaan urutan tiga besar untuk penggunaan huruf dalam bahasa Indonesia adalah "A", "N" dan kemudian "I", kita coba dahulu pasangkan "A" dengan "M" dan "N" dengan "Z" untuk mendapatkan dua persamaan:

$$12 \equiv a \cdot 0 + b \pmod{26}$$

dan

$$25 \equiv a \cdot 13 + b \pmod{26}.$$

Dengan mengurangkan persamaan pertama dari persamaan kedua, kita hilangkan b mendapatkan:

$$13 \equiv a \cdot 13 \pmod{26}.$$

Akan tetapi karena 13 tidak mempunyai *inverse* (karena 13 membagi 26), kita harus membagi persamaan dengan 13 menjadi:

$$1 \equiv a \cdot 1 \pmod{2},$$

yang berarti a adalah bilangan ganjil. Kita dapat mencoba setiap bilangan ganjil $0 < a < 26$ untuk mencari hasil yang cocok, akan tetapi mungkin kita tidak sabar untuk melakukan hal itu.

Sebagai alternatif, kita dapat lanjutkan analisa frekuensi dengan mencoba huruf dengan statistik penggunaan terbesar nomor tiga. Kita pasangkan "I" dengan "Q", menggantikan persamaan pertama. Jadi dua persamaan yang kita gunakan adalah:

$$16 \equiv a \cdot 8 + b \pmod{26}$$

dan

$$25 \equiv a \cdot 13 + b \pmod{26}.$$

Kita kurangkan persamaan pertama dari persamaan kedua mendapatkan:

$$9 \equiv a \cdot 5 \pmod{26}.$$

Jadi $a \equiv 9 \cdot 5^{-1} \equiv 9 \cdot 21 \equiv 7 \pmod{26}$, dan $b \equiv 16 - (7 \cdot 8) \equiv 12 \pmod{26}$. Dengan percobaan ini kita dapatkan parameter $a = 7$ dan $b = 12$.

Contoh diatas menunjukkan bahwa, dengan *affine transformation*, dua pasangan tidak selalu menghasilkan nilai parameter secara langsung. Kadang kita harus mencoba pasangan lain. Analisa frekuensi terhadap enkripsi *affine* memang lebih sulit dibandingkan analisa frekuensi terhadap enkripsi yang menggunakan *shift transformation*, namun analisa frekuensi terhadap enkripsi *affine* masih tergolong mudah untuk dilakukan.

4.2 Transformasi Digraph

Analisa frekuensi menjadi lebih dipersulit lagi jika enkripsi terhadap karakter tidak dilakukan satu persatu melainkan sekaligus terhadap beberapa karakter. Jika transformasi dilakukan terhadap dua karakter sekaligus, maka transformasi disebut transformasi *digraph* (*digraph transformation*). Sebelum enkripsi, jika jumlah karakter ganjil, naskah dapat ditambah dengan *padding character* agar jumlah karakter genap. Setelah dekripsi, *padding character* jika ada dapat dibuang.

Setiap *digraph* terdiri dari dua karakter dan diberi kode bilangan. Cara termudah untuk memberi kode bilangan adalah dengan rumus:

$$xn + y$$

dimana x adalah kode bilangan untuk karakter pertama, y adalah kode bilangan untuk karakter kedua, dan n adalah besarnya perbendaharaan karakter untuk enkripsi Jadi kode untuk digraph mempunyai nilai antara 0 dan $n^2 - 1$ inklusif.

Affine transformation dapat dilakukan terhadap digraph menggunakan rumus enkripsi:

$$C \equiv aP + b \pmod{n^2}. \tag{4.5}$$

Tentunya kita memerlukan $\gcd(a, n^2) = 1$ agar a mempunyai *inverse*. Akan tetapi kita cukup melakukan test $\gcd(a, n) = 1$ karena jika $\gcd(a, n) = 1$ maka kita tahu $\gcd(a, n^2) = 1$, sedangkan jika $\gcd(a, n) \neq 1$ maka kita tahu $\gcd(a, n^2) \neq 1$, meskipun $\gcd(a, n^2) \neq \gcd(a, n)$.

Rumus untuk dekripsi adalah:

$$P \equiv a^{-1}C - a^{-1}b \pmod{n^2}. \tag{4.6}$$

Sebagai contoh, kita gunakan perbendaharaan karakter terdiri dari semua huruf besar plus spasi, dengan kode bilangan 26 untuk spasi dan 0 sampai dengan 25 untuk huruf "A" sampai dengan "Z". Jadi $n = 27$ dan $n^2 = 729$. Menggunakan parameter $a = 614$ dan $b = 47$, rumus enkripsi menjadi:

$$C \equiv 614P + 47 \pmod{729}. \tag{4.7}$$

Parameter $a = 614$ dapat digunakan karena $\gcd(614, 729) = 1$, jadi a mempunyai *inverse* $a^{-1} \equiv 374 \pmod{729}$.

Rumus untuk dekripsi menjadi:

$$P \equiv 374C - 374 \cdot 47 \pmod{729} \equiv 374C + 647 \pmod{729}. \tag{4.8}$$

Mari kita coba enkripsi *digraph* dengan *affine transformation* diatas terhadap naskah: `"JANGAN RAHASIAKAN PESAN INI "`. Naskah harus ditambah *padding* ahir berupa spasi supaya jumlah karakter genap.

Digraph Asli	Kode Asli	Kode Acak	Digraph Acak
JA	243	533	TU
NG	357	545	UF
AN	13	10	AK
_R	719	468	RJ
AH	7	700	ZZ
AS	18	164	GC
IA	216	722	_U
KA	270	344	MU
N_	377	432	QA
PE	409	397	OT
SA	486	290	KU
N_	377	432	QA
IN	229	685	ZK
I_	242	648	YA

Tabel 4.3: Tabel untuk contoh enkripsi digraph

Naskah acak menjadi: `"TUUFAKRJZZGC UMUQAOTKUQAZKYA"`. Kita dapat melihat bahwa huruf acak tidak selalu sama untuk huruf asli yang sama. Contohnya, huruf asli "A" mempunyai huruf acak "A", "G", "U" dan "Z", meskipun diposisi kedua dalam *digraph*, "A" selalu ditukar dengan "U".

Mari kita coba lakukan analisa frekuensi terhadap naskah dengan menghitung frekuensi *digraph*. Namun naskah acak kurang panjang untuk mendapatkan statistik penggunaan *digraph* yang cukup baik. Hanya *digraph* "QA" yang digunakan lebih dari satu kali dalam naskah acak. Jadi kita bisa menyimpulkan bahwa analisa frekuensi terhadap *digraph* membutuhkan *sample* naskah acak yang lebih panjang daripada yang dibutuhkan jika transformasi dilakukan terhadap setiap huruf.

Mari kita umpamakan bahwa disamping naskah acak diatas, kita mendapatkan naskah acak lain yang cukup panjang untuk mendapatkan statistik penggunaan yang cukup baik dengan urutan empat terbesar penggunaan *digraph* sebagai berikut: "AK", "QA", "YA" dan "_A", dimana karakter "_" dalam *digraph* merepresentasikan spasi. Mari umpamakan juga bahwa dalam bahasa Indonesia, empat terbesar penggunaan *digraph* secara berurut adalah: "AN", "N_", "I_" dan "A_". Kita pasangkan *digraph* acak dengan *digraph* asli menurut urutan penggunaan.

Keempat pasangan menghasilkan empat persamaan yang dapat digunakan untuk mencari parameter a dan b:

$$10 \equiv 13a + b \pmod{729}$$

Digraph Acak (C)	Digraph Asli (P)
AK (10)	AN (13)
QA (432)	N_(377)
YA (648)	I_ (242)
_A (702)	A_ (26)

Tabel 4.4: Pasangan digraph menurut urutan frekuensi penggunaan

$$\begin{aligned} 432 &\equiv 377a + b \pmod{729} \\ 648 &\equiv 242a + b \pmod{729} \\ 702 &\equiv 26a + b \pmod{729} \end{aligned}$$

Dengan mengurangkan persamaan pertama dari persamaan kedua, kita hilangkan parameter b mendapatkan:

$$422 \equiv 364a \pmod{729}$$

Karena $\gcd(364, 729) = 1$, kita bisa mengkalkulasi $364^{-1} \pmod{729}$ menggunakan *extended Euclidean algorithm* menghasilkan $727 \pmod{729}$. Jadi:

$$a \equiv 422 \cdot 364^{-1} \equiv 422 \cdot 727 \equiv 614 \pmod{729}.$$

Dengan memasukkan nilai parameter a kedalam persamaan pertama, kita dapatkan:

$$b \equiv 10 - 13 \cdot 614 \equiv 47 \pmod{729}.$$

Jadi analisa frekuensi menghasilkan parameter $a = 614$ dan $b = 47$.

Kita beruntung mendapatkan nilai untuk parameter a dan b hanya dengan satu percobaan menggunakan persamaan pertama dan kedua. Seperti halnya analisa frekuensi *affine transformation* per karakter, analisa frekuensi *affine transformation* per *digraph* tidak selalu mendapatkan nilai a dan b secara langsung dari dua persamaan. Bahkan, pada umumnya, analisa frekuensi per *digraph* jauh lebih sulit dibandingkan analisa frekuensi per karakter. Namun enkripsi dengan *affine transformation* terhadap *digraph* masih dianggap rentan terhadap analisa frekuensi.

Pengamatan terhadap *affine transformation* terhadap *digraph* juga menunjukkan bahwa huruf kedua dalam *digraph* acak ditentukan hanya oleh huruf kedua dalam *digraph* asli ("U" sebagai huruf kedua dalam *digraph* acak selalu berasal dari "A" sebagai huruf kedua dalam *digraph* asli, "K" berasal dari "N", dan seterusnya). Ini disebabkan huruf kedua *digraph* acak hanya tergantung nilai *digraph* asli modulo n, jadi hanya tergantung huruf kedua *digraph* asli. Akibatnya analisa frekuensi terhadap huruf kedua semua *digraph* acak dapat menghasilkan parameter a dan b modulo n. Meskipun belum menentukan a

dan b modulo n^2, informasi ini cukup berharga dalam memecahkan enkripsi, karena ruang pencarian diperkecil secara signifikan. Kita tidak perlu mengindahkan berbagai nilai untuk parameter a dan b yang tidak sesuai dengan nilai a dan b modulo n. Sebagai contoh, jika $a \equiv 5 \pmod{27}$, kita tidak perlu mengindahkan $a \equiv 33 \pmod{729}$ karena $33 \equiv 6 \not\equiv 5 \pmod{27}$.

4.3 Matrik Enkripsi

Kita sudah melihat bagaimana *digraph* dapat direpresentasikan menggunakan bilangan bulat modulo n^2. Sudut pandang lain adalah melihat *digraph* sebagai vektor dua dimensi. Sebagai contoh, *digraph* "DI" dapat dipandang sebagai vektor

$$\begin{bmatrix} 3 \\ 8 \end{bmatrix}.$$

Enkripsi terhadap *digraph* dapat dipandang sebagai transformasi vektor yang bersifat *one-to-one*. Sifat *one-to-one* diperlukan agar kode acak dapat didekripsi secara unik. Karena *domain* dan *codomain* (target) sama dan bersifat *finite* (yaitu ruang vektor $(\mathbf{Z}/n\mathbf{Z})^2$ dimana n adalah besarnya perbendaharaan karakter), transformasi *one-to-one* vektor adalah *bijective map*. Jadi transformasi dekripsi, yang merupakan *inverse* dari transformasi enkripsi, mempunyai *domain* yang meliputi seluruh ruang vektor $(\mathbf{Z}/n\mathbf{Z})^2$.

Sudut pandang enkripsi sebagai transformasi vektor berlaku juga untuk blok yang berisi lebih dari dua karakter, dengan dimensi vektor sama dengan banyaknya karakter dalam blok. Dimasa silam, teknik enkripsi yang sangat populer selama ratusan tahun adalah *Vigenère cipher*. Dengan *Vigenère cipher*, setiap blok yang terdiri dari k karakter direpresentasikan menggunakan vektor dengan ruang $(\mathbf{Z}/n\mathbf{Z})^k$, dan enkripsi berupa *shift transformation* sebagai berikut:

$$C = P + b, \tag{4.9}$$

dimana b yang berupa vektor dengan dimensi k adalah kunci enkripsi. Transformasi dilakukan dengan menambahkan vektor b ke vektor asli P sebagai berikut:

$$P = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_k \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}, C = P + b = \begin{bmatrix} p_1 + b_1 \\ p_2 + b_2 \\ \vdots \\ p_k + b_k \end{bmatrix}.$$

Pertambahan skalar $p_i + b_i$ untuk $1 \leq i \leq k$ menggunakan aritmatika modulo n (aritmatika $\mathbf{Z}/n\mathbf{Z}$). Dekripsi dilakukan sebagai berikut:

$$P = C + b', \tag{4.10}$$

dengan

$$b' = -b = \begin{bmatrix} -b_1 \\ -b_2 \\ \vdots \\ -b_k \end{bmatrix}.$$

Setiap $-b_i$ untuk $1 \leq i \leq k$ menggunakan *inverse* pertambahan dalam aritmatika $\mathbf{Z}/n\mathbf{Z}$.

Vigenère cipher sangat rentan terhadap analisa frekuensi. Analisa frekuensi *shift transformation* terhadap karakter seperti yang dijelaskan di 2.3.2 dapat dilakukan terhadap setiap posisi dalam vektor menghasilkan komponen vektor b. Jadi analisa posisi 1 menghasilkan b_1, analisa posisi 2 menghasilkan b_2, dan seterusnya sampai semua komponen vektor kunci b ditemukan.

Selain *shift transformation* dimana *shift vector* ditambahkan ke vektor asal, transformasi terhadap vektor juga dapat dilakukan dengan mengalikan matrik ke vektor asal. Untuk keperluan enkripsi, matrik pengali yang disebut matrik enkripsi (*enciphering matrix*) harus mempunyai *inverse* matrik. Perkalian matrik dengan vektor sesuai dengan aljabar linear. Sebagai contoh, mengalikan matrik berdimensi 2×2 dengan vektor dimensi 2 kita dapatkan:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix},$$

dimana $ax + by$ dan $cx + dy$ dikalkulasi menggunakan aritmatika $\mathbf{Z}/n\mathbf{Z}$.

Dalam aljabar linear dengan bilangan nyata (*real*), sebuah matrik mempunyai *inverse* jika dan hanya jika ($\Longleftrightarrow$) determinan (*determinant*) matrik D mempunyai *inverse* D^{-1}, yang dalam aritmatika bilangan nyata berarti $D \neq 0$. Demikian juga untuk matrik enkripsi A, A mempunyai *inverse* jika dan hanya jika ($\Longleftrightarrow$) deteminan A, yang dikalkulasi menggunakan aritmatika $\mathbf{Z}/n\mathbf{Z}$, D mempunyai inverse D^{-1}. Tetapi berbeda dengan aritmatika bilangan nyata, ini berarti $\gcd(D, n) = 1$ (lihat teorema 11).

Mari kita tinjau kembali konsep-konsep aljabar linear yang dapat digunakan untuk mengkalkulkasi *inverse* matrik. Konsep utama adalah konsep determinan, yang dapat didefinisikan secara rekursif (*recursive definition* atau *inductive definition*). Untuk matrik dengan dimensi 1×1, definisi determinan sangat sederhana:

$$A = [a_{1,1}],\ \det(A) = a_{1,1}.$$

Untuk matrik dengan dimensi $n \times n$ dimana $n > 1$, kita dapat memilih baris i dengan $1 \leq i \leq n$, dan deteminan dapat dikalkulasi sebagai berikut:

$$\det(A) = a_{i,1} \cdot \text{cofactor}_{i,1}(A) + a_{i,2} \cdot \text{cofactor}_{i,2}(A) + \ldots + a_{i,n} \cdot \text{cofactor}_{i,n}(A),$$

dimana $\text{cofactor}_{i,j}(A)$ adalah determinan matrik yang didapat dengan menghapus baris i dan kolom j dari matrik A, dikalikan dengan $(-1)^{i+j}$ (jika $i + j$

ganjil, determinan dikalikan dengan -1, jika $i + j$ genap, determinan tidak dikalikan). Sebagai contoh, untuk matrik dengan dimensi 2×2:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix},$$

terdapat empat *cofactor*, masing-masing:

$$\begin{aligned} \text{cofactor}_{1,1}(A) &= a_{2,2}, \\ \text{cofactor}_{1,2}(A) &= -a_{2,1}, \\ \text{cofactor}_{2,1}(A) &= -a_{1,2}, \\ \text{cofactor}_{2,2}(A) &= a_{1,1}, \end{aligned}$$

dan determinan dapat dikalkulasi dengan:

$$\begin{aligned} \det(A) &= a_{1,1} \cdot \text{cofactor}_{1,1}(A) + a_{1,2} \cdot \text{cofactor}_{1,2}(A) \\ &= a_{1,1} \cdot a_{2,2} + a_{1,2} \cdot (-a_{2,1}) \end{aligned}$$

atau

$$\begin{aligned} \det(A) &= a_{2,1} \cdot \text{cofactor}_{2,1}(A) + a_{2,2} \cdot \text{cofactor}_{2,2}(A) \\ &= a_{2,1} \cdot (-a_{1,2}) + a_{2,2} \cdot a_{1,1}. \end{aligned}$$

Selain determinan dan *cofactor*, kita memerlukan konsep *transpose* matrik:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & & & \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix}, A^T = \begin{bmatrix} a_{1,1} & a_{2,1} & \dots & a_{n,1} \\ a_{1,2} & a_{2,2} & \dots & a_{n,2} \\ \vdots & & & \\ a_{1,n} & a_{2,n} & \dots & a_{n,n} \end{bmatrix},$$

dan *adjoint* matrik:

$$\text{adj}(A) = \begin{bmatrix} \text{cofactor}_{1,1}(A) & \text{cofactor}_{1,2}(A) & \dots & \text{cofactor}_{1,n}(A) \\ \text{cofactor}_{2,1}(A) & \text{cofactor}_{2,2}(A) & \dots & \text{cofactor}_{2,n}(A) \\ \vdots & & & \\ \text{cofactor}_{n,1}(A) & \text{cofactor}_{n,2}(A) & \dots & \text{cofactor}_{n,n}(A) \end{bmatrix}^T.$$

Jadi *transpose* sebuah matrik didapatkan dengan mempertukarkan baris dengan kolom: baris 1 menjadi kolom 1, kolom 1 menjadi baris 1, baris 2 menjadi kolom 2, kolom 2 menjadi baris 2, dan seterusnya. Sedangkan *adjoint* dari sebuah matrik adalah *transpose* dari matrik *cofactor*.

Kita dapat mengkalkulasi *inverse* matrik sebagai berikut:

$$A^{-1} = \det(A)^{-1} \cdot \text{adj}(A). \tag{4.11}$$

Untuk matrik dengan dimensi 2×2, rumus untuk *inverse* menjadi:

$$A^{-1} = (a_{1,1} \cdot a_{2,2} + a_{1,2} \cdot (-a_{2,1}))^{-1} \cdot \begin{bmatrix} a_{2,2} & -a_{1,2} \\ -a_{2,1} & a_{1,1} \end{bmatrix}.$$

Kita coba kalkulasi *inverse* matrik enkripsi:

$$A = \begin{bmatrix} 2 & 3 \\ 7 & 8 \end{bmatrix},$$

dimana $n = 26$, jadi aritmatika yang digunakan adalah aritmatika $\mathbf{Z}/26\mathbf{Z}$. Determinan $D = \det(A) = (2 \cdot 8 + (-3) \cdot 7) \bmod 26 = -5 \bmod 26 = 21$. Karena $\gcd(21, 26) = 1$, D mempunyai *inverse*, yang dikalkulasi menggunakan *extended Euclidean algorithm* menghasilkan 5. Jadi,

$$A^{-1} = 5 \cdot \begin{bmatrix} 8 & -3 \\ -7 & 2 \end{bmatrix} = \begin{bmatrix} 40 \bmod 26 & -15 \bmod 26 \\ -35 \bmod 26 & 10 \bmod 26 \end{bmatrix} = \begin{bmatrix} 14 & 11 \\ 17 & 10 \end{bmatrix}.$$

Kita dapat periksa:

$$\begin{bmatrix} 14 & 11 \\ 17 & 10 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 105 \bmod 26 & 130 \bmod 26 \\ 104 \bmod 26 & 131 \bmod 26 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Mari kita coba enkripsi "DIMANASAJA" menggunakan matrik enkripsi diatas. Naskah terdiri dari 5 *digraph* yang direpresentasikan menggunakan 5 vektor:

$$\begin{bmatrix} 3 \\ 8 \end{bmatrix} \begin{bmatrix} 12 \\ 0 \end{bmatrix} \begin{bmatrix} 13 \\ 0 \end{bmatrix} \begin{bmatrix} 18 \\ 0 \end{bmatrix} \begin{bmatrix} 9 \\ 0 \end{bmatrix}.$$

Kelima vektor dapat digabung menjadi satu matrik:

$$\begin{bmatrix} 3 & 12 & 13 & 18 & 9 \\ 8 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Kita kalikan matrik enkripsi dengan matrik naskah asli untuk mendapatkan matrik naskah acak:

$$\begin{aligned} C &= AP \\ &= \begin{bmatrix} 2 & 3 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 3 & 12 & 13 & 18 & 9 \\ 8 & 0 & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 30 \bmod 26 & 24 \bmod 26 & 26 \bmod 26 & 36 \bmod 26 & 18 \bmod 26 \\ 85 \bmod 26 & 84 \bmod 26 & 91 \bmod 26 & 126 \bmod 26 & 63 \bmod 26 \end{bmatrix} \\ &= \begin{bmatrix} 4 & 24 & 0 & 10 & 18 \\ 7 & 6 & 13 & 22 & 11 \end{bmatrix}. \end{aligned}$$

Jadi kita dapatkan naskah acak: "EHYGANKWSL".

Kita periksa apakah *inverse* matrik enkripsi dapat digunakan untuk dekripsi matrik naskah acak:

$$\begin{aligned} P &= A^{-1}C \\ &= \begin{bmatrix} 14 & 11 \\ 17 & 10 \end{bmatrix} \begin{bmatrix} 4 & 24 & 0 & 10 & 18 \\ 7 & 6 & 13 & 22 & 11 \end{bmatrix} \\ &= \begin{bmatrix} 133 & 402 & 143 & 382 & 373 \\ 138 & 468 & 130 & 390 & 416 \end{bmatrix} \bmod 26 \\ &= \begin{bmatrix} 3 & 12 & 13 & 18 & 9 \\ 8 & 0 & 0 & 0 & 0 \end{bmatrix}. \end{aligned}$$

Jadi mengalikan *inverse* matrik enkripsi dengan matrik naskah acak menghasilkan matrik naskah asli.

Mari kita selidiki bagaimana transformasi dengan matrik enkripsi dapat dianalisa. Kita andaikan naskah acak "YPKWPRHODEZKCJ" dienkripsi dengan matrik enkripsi terhadap *digraph* dan kita mengetahui bahwa naskah asli diahiri dengan nama pengirim yaitu "NETTY". Jadi kita mengetahui bahwa dua *digraph* terahir dalam naskah asli adalah "ET" dan "TY". Dua *digraph* terahir dalam naskah acak adalah "ZK" dan "CJ", jadi "ZK" dan "CJ" adalah enkripsi dari "ET" dan "TY". Kita representasikan dua *digraph* asli dengan matrik:

$$P = \begin{bmatrix} 4 & 19 \\ 19 & 24 \end{bmatrix}$$

dan dua *digraph* acak dengan matrik:

$$C = \begin{bmatrix} 25 & 2 \\ 10 & 9 \end{bmatrix}.$$

Menggunakan rumus untuk dekripsi $P = A^{-1}C$, kita dapatkan:

$$\begin{bmatrix} 4 & 19 \\ 19 & 24 \end{bmatrix} = A^{-1} \begin{bmatrix} 25 & 2 \\ 10 & 9 \end{bmatrix}.$$

Jadi kita bisa mendapatkan A^{-1} sebagai berikut:

$$A^{-1} = \begin{bmatrix} 4 & 19 \\ 19 & 24 \end{bmatrix} \begin{bmatrix} 25 & 2 \\ 10 & 9 \end{bmatrix}^{-1} = \begin{bmatrix} 4 & 19 \\ 19 & 24 \end{bmatrix} \begin{bmatrix} 23 & 18 \\ 12 & 9 \end{bmatrix} = \begin{bmatrix} 8 & 9 \\ 23 & 12 \end{bmatrix}.$$

Kita periksa apakah A^{-1} bisa mendapatkan naskah asli yang "masuk akal:"

$$\begin{aligned} P &= \begin{bmatrix} 8 & 9 \\ 23 & 12 \end{bmatrix} \begin{bmatrix} 24 & 10 & 15 & 7 & 3 & 25 & 2 \\ 15 & 22 & 17 & 14 & 4 & 10 & 9 \end{bmatrix} \\ &= \begin{bmatrix} 15 & 18 & 13 & 0 & 8 & 4 & 19 \\ 4 & 0 & 3 & 17 & 13 & 19 & 24 \end{bmatrix}. \end{aligned}$$

Naskah asli yang kita dapatkan yaitu "PESANDARINETTY" cukup masuk akal. Jadi kita bisa cukup yakin bahwa A^{-1} yang kita dapatkan adalah benar.

Kita baru saja berhasil melakukan *known-plaintext attack* terhadap transformasi dengan matrik enkripsi. Analisa frekuensi juga dapat dilakukan terhadap transformasi matrik enkripsi, dengan matrik acak dan matrik asli dipasangkan menurut analisa frekuensi, bukan berdasarkan pengetahuan pasti mengenai naskah asli. Rumus untuk mendapatkan A^{-1} hanya dapat digunakan apabila matrik acak mempunyai *inverse*, jadi ada kemungkinan lebih dari satu pasangan harus dicoba (ini berlaku juga untuk *known-plaintext attack*). Matrik acak dan asli harus berdimensi $n \times n$ dimana n adalah jumlah karakter dalam blok.

Transformasi dengan matrik enkripsi tergolong *linear transformation*, yang berarti jika C_1 adalah naskah acak untuk P_1 dan C_2 adalah naskah acak untuk P_2, maka $C_1 + C_2$ adalah naskah acak untuk $P_1 + P_2$. Transformasi matrik enkripsi dapat digabung dengan *shift transformation* menghasilkan *affine transformation*. Matrik enkripsi A dikalikan dengan vektor asli, kemudian hasilnya ditambahkan dengan *shift vector* B:

$$C = AP + B. \tag{4.12}$$

Sebagai contoh, untuk *digraph* rumus menjadi:

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_{1,1}p_1 + a_{1,2}p_2 + b_1 \\ a_{2,1}p_1 + a_{2,2}p_2 + b_2 \end{bmatrix}.$$

Rumus untuk dekripsi adalah:

$$P = A^{-1}C - A^{-1}B. \tag{4.13}$$

Analisa *affine transformation* matrik enkripsi terhadap *digraph* membutuhkan minimum 3 pasang *digraph* dengan 3 persamaan:

$$\begin{aligned} P_1 &= A^{-1}C_1 - A^{-1}B \\ P_2 &= A^{-1}C_2 - A^{-1}B \\ P_3 &= A^{-1}C_3 - A^{-1}B \end{aligned}$$

Persamaan ketiga dikurangkan dari dua persamaan pertama untuk menghilangkan *shift vector* B. Teknik untuk *linear transformation* yang telah dijelaskan dapat digunakan untuk mencari A^{-1} menggunakan dua persamaan yang sudah tidak mengandung B (kedua persamaan dapat digabung dengan menggabungkan vektor menjadi matrik). Setelah A^{-1} didapat, maka B dapat dikalkulasi.

Secara umum, untuk *affine transformation* menggunakan matrik enkripsi dengan besar blok n karakter, dibutuhkan $n+1$ persamaan untuk mengkalkulasi A^{-1} dan B. (Lagi n vektor dapat digabung menjadi matrik.)

4.4 Ringkasan

Enkripsi *Caesar cipher* sangat mudah untuk dipecahkan, baik menggunakan analisa statistik, maupun dengan cara *known plaintext attack*. Di bab ini telah diperkenalkan teknik-teknik enkripsi yang mempersulit analisa statistik dan cara pemecahan lainnya. Teknik *affine transformation* membuat enkripsi semakin tidak linear jadi lebih sukar analisanya. Enkripsi beberapa karakter sekaligus juga merupakan kemajuan dalam kriptografi karena analisanya lebih sulit dibandingkan jika setiap karakter dienkripsi sendiri. Konsep *digraph* digunakan jika dua karakter sekaligus dienkripsi. Transformasi beberapa karakter sekaligus dapat direpresentasikan menggunakan matrik enkripsi. Meskipun teknik-teknik yang diperkenalkan dalam bab ini membuat enkripsi lebih kuat dibandingkan dengan *Caesar cipher*, hasilnya masih rentan terhadap berbagai analisa statistik. Dalam bab-bab selanjutnya, kita akan bahas teknik-teknik yang dapat membuat enkripsi lebih kuat lagi.

Bab 5

Matematika II - Polynomial Field

Di ahir bab 3 kita melihat bagaimana aritmatika modulo sebuah bilangan prima mempunyai struktur *finite field*. *Finite field* seperti itu dinamakan *prime field*, dan dari *prime field*, kita dapat membuat *field* yang lebih besar yang dinamakan *polynomial field*. Dalam bab ini kita akan bahas aritmatika *polynomial field*, yang digunakan antara lain dalam enkripsi AES, dimana transformasi *affine* dengan aritmatika *polynomial field* digunakan untuk substitusi. Pembaca yang cukup paham dengan *cyclic redundancy check* (CRC) tentunya mengetahui bahwa CRC juga menggunakan aritmatika *polynomial field*.

Sebelum membahas aritmatika *polynomial field*, kita perlu kembangkan dahulu teori mengenai *ring* dengan membahas beberapa konsep, antara lain konsep *integral domain*, *homomorphism*, *ideal* dalam suatu *ring*, *principal ideal domain*, *polynomial ring* dan *irreducible polynomial*.

Notasi logika matematika akan banyak digunakan di bab ini. Tabel 5.1 menjelaskan notasi logika yang digunakan.

Beberapa pembuktian matematika di bab ini akan menggunakan rantaian $\Longrightarrow$ dan rantaian $\Longleftrightarrow$. Pembuktian dengan bentuk

$$A_1 \Longrightarrow A_2 \Longrightarrow A_3 \Longrightarrow A_4$$

agar dibaca sebagai A_4 merupakan konsekuensi dari A_3, yang merupakan konsekuensi dari A_2, yang merupakan konsekuensi dari A_1. Demikian juga, pembuktian dengan bentuk

$$A_1 \Longleftrightarrow A_2 \Longleftrightarrow A_3 \Longleftrightarrow A_4$$

agar dibaca sebagai A_4 ekuivalen dengan A_3, yang ekuivalen dengan A_2, yang ekuivalen dengan A_1.

Notasi	**Penjelasan**
$A \Longrightarrow B$	A hanya jika B (atau B jika A).
$A \Longleftrightarrow B$	A jika dan hanya jika B.
$A \wedge B$	A dan B.
$A \vee B$	A atau B.
$\neg A$	Tidak A.
$\forall x : P(x)$	Untuk setiap x, $P(x)$ berlaku.
$\forall x \in S : P(x)$	Untuk setiap $x \in S$, $P(x)$ berlaku.
$\exists x : P(x)$	Terdapat x dimana $P(x)$ berlaku.
$\exists x \in S : P(x)$	Terdapat $x \in S$ dimana $P(x)$ berlaku.

Tabel 5.1: Tabel Notasi Logika Matematika

5.1 Integral Domain

Kita akan mulai dengan membahas konsep *integral domain*, tetapi sebelumnya kita definisikan konsep *zero divisor* dan *unit*.

Definisi 9 *Untuk ring R dan elemen $a \in R$,*

- *$a \neq 0$ adalah zero divisor jika ada $0 \neq b \in R$ dengan $ab = 0$,*
- *a adalah unit jika a mempunyai inverse.*

Suatu *zero divisor* tidak mungkin juga merupakan *unit* karena jika $a \in R$ adalah *zero divisor*, maka terdapat $0 \neq b \in R$ dengan $ab = 0$, sedangkan jika a juga merupakan *unit*, maka terdapat $0 \neq c \in R$ dengan $ac = 1$, jadi $0 = c \cdot 0 = c(ab) = (ac)b = 1 \cdot b = b$, suatu kontradiksi.

Suatu *ring* yang tidak mempunyai *zero divisor* dinamakan *integral domain*. $\mathbf{Z}$ merupakan *integral domain* karena dalam aritmatika bilangan bulat tidak ada *zero divisor*.

Teorema 12 *Jika R merupakan suatu integral domain, $a, b, c \in R \setminus \{0\}$, dan $ab = ac$, maka $b = c$. Menggunakan notasi logika:*

$$\forall a, b, c \in R \setminus \{0\} : (ab = ac) \Longrightarrow (b = c).$$

Kita buktikan teorema 12 secara kontra-positif. Jika $b \neq c$ maka konsekuensi $ab = ac$ adalah

$$a(b - c) = 0,$$

sesuatu yang tidak mungkin karena a dan $b - c$ keduanya bukan *zero divisor*. Jadi jika $ab = ac$ maka $b = c$.

Teorema 13 *Suatu finite integral domain R merupakan suatu field (tentu saja finite field).*

Untuk membuktikan teorema 13, kita tunjukkan bahwa setiap elemen dari R yang bukan 0 merupakan *unit*. Jika $a \in R \setminus \{0\}$ maka fungsi

$$f_a(x) = ax$$

untuk $x \in R \setminus \{0\}$ merupakan suatu *bijection* dari $R \setminus \{0\}$ ke $R \setminus \{0\}$. Jadi terdapat $b \in R \setminus \{0\}$ dimana $ab = 1$, dengan kata lain a merupakan *unit* karena mempunyai *inverse* yaitu $a^{-1} = b$. Jadi setiap elemen dari R yang bukan 0 merupakan *unit*, jadi R merupakan *field*.

5.2 Homomorphism dan Ideal

Suatu *homomorphism* antara dua himpunan adalah suatu fungsi yang mempertahankan struktur aljabar.

Definisi 10 (Homomorphism) *Untuk ring, suatu homomorphism $\varphi : R \longrightarrow S$ dari ring R ke ring S mempertahankan struktur ring sebagai berikut:*

- $\forall a, b \in R : \varphi(a + b) = \varphi(a) + \varphi(b)$,
- $\forall a, b \in R : \varphi(ab) = \varphi(a) \cdot \varphi(b)$, *dan*
- $\varphi(1_R) = 1_S$,

dimana 1_R adalah 1 untuk R dan 1_S adalah 1 untuk S.

Kerap 1 dan 0 tanpa subskrip digunakan jika jelas apa yang dimaksud. Jika *homomorphism* bersifat *injective* ($\varphi(a) = \varphi(b)$ hanya jika $a = b$), maka *homomorphism* disebut *embedding*. Jika *homomorphism* bersifat *bijective* (*injective* dan *surjective*), maka *homomorphism* disebut *isomorphism*. *Homomorphism* φ dari R ke S bersifat *surjective* jika untuk setiap $b \in S$, terdapat $a \in R$ dimana $b = \varphi(a)$, jadi φ "mengisi penuh" S. Jika φ adalah *isomorphism* dari R ke S, maka φ^{-1} adalah *isomorphism* dari S ke R, dan S disebut *isomorphic* dengan R dan diberi notasi $R \simeq S$.

Contoh dari *homomorphism* adalah *canonical homomorphism* dari $\mathbf{Z}$ ke $\mathbf{Z}/7\mathbf{Z}$ sebagai berikut:

$$\begin{array}{ccc} \mathbf{Z} & \longrightarrow & \mathbf{Z}/7\mathbf{Z} \\ a & \mapsto & [a]. \end{array}$$

Kita periksa apakah ini benar merupakan homomorphism:

$$\varphi(a + b) = [a + b] = [a] + [b] = \varphi(a) + \varphi(b)$$

$$\varphi(a \cdot b) = [a \cdot b] = [a] \cdot [b] = \varphi(a) \cdot \varphi(b)$$

$$\varphi(1) = [1] = 1_{\mathbf{Z}/7\mathbf{Z}}$$

Jadi ada *homomorphism* dari aritmatika bilangan bulat ke aritmatika modulo 7. (Pembaca dapat meninjau kembali bagian 3.5 mengenai aritmatika modular.)

Konsep berikutnya yang perlu dibahas adalah konsep *ideal* dalam suatu *ring*. Mari kita tinjau kembali aritmatika modulo sebuah bilangan, sebut saja n. Dalam aritmatika modulo n, setiap bilangan jika dikalikan dengan bilangan yang berada dalam *congruence class* [0] (yang berisi semua kelipatan n, jadi $[0] = n\mathbf{Z}$) akan menghasilkan bilangan dalam *congruence class* [0]. Konsep ini sangat penting dalam teori *ring*, kita katakan $n\mathbf{Z}$ adalah *ideal* dalam *ring* $\mathbf{Z}$. Jadi sebetulnya aritmatika modulo 7 adalah aritmatika bilangan bulat modulo *ideal* $7\mathbf{Z}$.

Dalam struktur *ring*, sesuatu yang berada dalam *ring* jika dikalikan dengan sesuatu yang berada dalam suatu *ideal* dalam *ring* akan menghasilkan sesuatu dalam *ideal* (*ideal* mempunyai sifat *inside-outside multiplication*), dan sesuatu yang berada dalam *ideal* jika ditambahkan dengan sesuatu yang juga berada dalam *ideal* menghasilkan sesuatu dalam *ideal*. Jadi $n\mathbf{Z}$ adalah suatu *ideal* dalam $\mathbf{Z}$ (*ring* bilangan bulat) karena kelipatan n dikalikan apa saja menghasilkan kelipatan n, dan kelipatan n ditambahkan dengan kelipatan n menghasilkan kelipatan n.

Secara formal definisi untuk *ideal* adalah sebagai berikut (dengan menggunakan $\Longrightarrow$ untuk "berarti"):

Definisi 11 (ideal) *$I \subseteq R$ adalah ideal dari ring R jika:*

- $\forall a, b \in I : (a + b) \in I$ *dan*
- $\forall a \in R, n \in I : (a \cdot n) \in I$.

Untuk setiap *ring* R, jelas bahwa $\{0\}$ merupakan *ideal* dari R (dinamakan *trivial ideal*), karena $0 + 0 = 0$ dan $a \cdot 0 = 0$. Juga jelas bahwa R merupakan *ideal* dalam R karena sifat *closure* untuk *ring*. *Ideal* I dalam *ring* R adalah *proper ideal* dalam R jika $I \neq R$. Jika I adalah suatu *ideal*, maka

$$0 \in I$$

karena apapun dikalikan dengan 0 akan menghasilkan 0.

Satu dari sekian cara untuk mendapatkan *ideal* dalam suatu *ring* adalah dengan menggunakan *generator* tunggal berupa elemen dalam *ring*. Menggunakan *generator* tunggal $n \in R$, suatu *ideal* dalam *ring* R didapat dengan mengumpulkan semua kelipatan n. *Ideal* yang didapat dengan cara ini dinamakan *principal ideal* dengan *generator* tunggal n, dan diberi notasi nR. Jadi *ideal* $7\mathbf{Z}$ adalah *principal ideal* dengan *generator* tunggal 7. Notasi $7\mathbf{Z}$ digunakan untuk *ideal*, bukan [0], agar jelas apa yang dimaksud (notasi [0] hanya menjelaskan himpunan sebagai *congruence class* yang mempunyai elemen 0,

tidak menjelaskan *ideal* yang dimaksud yaitu berisi semua bilangan kelipatan 7).

Secara umum, *generator* untuk *ideal* berupa himpunan A yang merupakan *subset* dari R ($A \subseteq R$). *Ideal* yang didapat adalah himpunan semua kombinasi linear dari elemen-elemen A:

$$\left\{ \sum_{i=1}^{n} a_i r_i \middle| 0 < n \in \mathbf{N}, r_i \in R \text{ dan } a_i \in A \text{ untuk } 1 \leq i \leq n \right\}.$$

Jika *generator* A merupakan *finite subset* dari R, maka *ideal* disebut *finitely generated*. Jelas bahwa suatu *principal ideal* adalah *finitely generated*.

Dalam memperkenalkan konsep *homomorphism*, *canonical homomorphism* dari $\mathbf{Z}$ ke $\mathbf{Z}/7\mathbf{Z}$ dijadikan contoh, dengan pemetaan elemen a: $a \mapsto [a]$. Pemetaan ini khusus untuk $\mathbf{Z}/n\mathbf{Z}$. Secara umum, untuk *ideal* I dalam *ring* R, elemen a dari R dapat dipetakan sebagai berikut:

$$\begin{array}{ccc} R & \longrightarrow & R/I \\ a & \mapsto & a+I \end{array}$$

menghasilkan *canonical homomorphism* dari R ke R/I (*ring* R modulo *ideal* I) dimana $a + I$ didefinisikan sebagai berikut:

$$a + I = \{a + e | e \in I\}.$$

Jadi $a + I$ adalah himpunan yang didapat dengan menambahkan a terhadap setiap elemen dari I. Sebagai contoh, kembali ke $\mathbf{Z}/7\mathbf{Z}$ dimana $I = [0]$, $3 + I = 3 + [0] = [3]$.

Kita definisikan pertambahan, perkalian dan *inverse* pertambahan untuk $a + I$ sebagai berikut:

$$\begin{aligned} (a+I)+(b+I) &= ((a+b)+I) \\ (a+I)\cdot(b+I) &= ((a\cdot b)+I) \\ -(a+I) &= (-a+I) \end{aligned}$$

Mari kita tunjukkan bahwa R/I adalah *ring* (semua aksioma *ring* berlaku) untuk sembarang *ring* R dan *proper ideal* I, dengan

$$\begin{aligned} 0_{R/I} &= 0_R + I = I \\ 1_{R/I} &= 1_R + I \end{aligned}$$

Associativity untuk $+$:

$$\begin{aligned} (a+I)+((b+I)+(c+I)) &= (a+I)+((b+c)+I) \\ &= ((a+(b+c))+I) \end{aligned}$$

$$\begin{aligned} &= (((a+b)+c)+I) \\ &= ((a+b)+I)+(c+I) \\ &= ((a+I)+(b+I))+(c+I). \end{aligned}$$

Identity untuk $+$:

$$(a+I)+(0_R+I) = ((a+0_R)+I) = (a+I).$$

Commutativity untuk $+$:

$$(a+I)+(b+I) = ((a+b)+I) = ((b+a)+I) = (b+I)+(a+I).$$

Inverse untuk $+$:

$$(a+I)+(-(a+I)) = (a+I)+(-a+I) = ((a+(-a))+I) = (0_R+I) = I.$$

Associativity untuk $\cdot$:

$$\begin{aligned} (a+I)\cdot((b+I)\cdot(c+I)) &= (a+I)\cdot((b\cdot c)+I) \\ &= ((a\cdot(b\cdot c))+I) \\ &= (((a\cdot b)\cdot c)+I) \\ &= ((a\cdot b)+I)\cdot(c+I) \\ &= ((a+I)\cdot(b+I))\cdot(c+I). \end{aligned}$$

Identity untuk $\cdot$:

$$(a+I)\cdot(1_R+I) = ((a\cdot 1_R)+I) = (a+I).$$

Commutativity untuk $\cdot$:

$$(a+I)\cdot(b+I) = ((a\cdot b)+I) = ((b\cdot a)+I) = (b+I)\cdot(a+I).$$

Distributivity:

$$\begin{aligned} (a+I)\cdot((b+I)+(c+I)) &= (a+I)\cdot((b+c)+I) \\ &= ((a\cdot(b+c))+I) \\ &= (((a\cdot b)+(a\cdot c))+I) \\ &= ((a\cdot b)+I)+((a\cdot c)+I) \\ &= ((a+I)\cdot(b+I))+((a+I)\cdot(c+I)). \end{aligned}$$

Ring R/I dinamakan *quotient ring*.

Sekarang kita bahas bagaimana *ideal* dipetakan oleh suatu *homomorphism*. Jika $\varphi : R \longrightarrow S$ merupakan *homomorphism* dari *ring* R ke *ring* S, maka *kernel* dari *homomorphism* dengan notasi $\ker(\varphi)$ adalah subset dari R sebagai berikut:

$$\ker(\varphi) = \{a \in R | \varphi(a) = 0_S\}$$

jadi *kernel* dari *homomorphism* adalah subset dari *ring* asal R, dan terdiri dari semua elemen R yang dipetakan ke 0 dalam *ring* tujuan S. Tidak terlalu sulit untuk membuktikan bahwa $\ker(\varphi)$ adalah suatu *proper ideal* dalam R. Kita tahu bahwa $0_R \in \ker(\varphi)$ karena $\varphi(0_R) = 0_S$, jadi $\ker(\varphi) \neq \emptyset$. Kita buktikan bahwa $\ker(\varphi)$ adalah suatu *ideal* dalam R:

- Jika $a, b \in \ker(\varphi)$, maka $\varphi(a) = \varphi(b) = 0_S$, dan $\varphi(a+b) = \varphi(a) + \varphi(b) = 0_S + 0_S = 0_S$, jadi $a + b \in \ker(\varphi)$.
- Jika $a \in \ker(\varphi)$ dan $r \in R$, maka $\varphi(a) = 0_S$, dan $\varphi(ar) = \varphi(a) \cdot \varphi(r) = 0_S \cdot \varphi(r) = 0_S$, jadi $ar \in \ker(\varphi)$.

Jadi $\ker(\varphi)$ adalah suatu *ideal* dalam R. Karena $\varphi(1_R) = 1_S \neq 0_S$, $1_R \notin \ker(\varphi)$, $\ker(\varphi)$ adalah *proper ideal*.

Masih menyangkut *homomorphism* φ dari R ke S, setiap *ideal* J dalam S "berasal" dari *ideal* yang mencakup $\ker(\varphi)$ dalam R:

$$I = \{c | c \in R, \varphi(c) \in J\} \text{ adalah } \textit{ideal} \text{ dalam } R \text{ dan } \ker(\varphi) \subseteq I. \tag{5.1}$$

Dengan notasi himpunan, konsep "J berasal dari I" diformalkan, dan I disebut *inverse image* dari J menurut φ. Mari kita buktikan 5.1. Jika $a, b \in I$, maka $\varphi(a), \varphi(b) \in J$, jadi karena J merupakan *ideal*,

$$\varphi(a+b) = \varphi(a) + \varphi(b) \in J$$

jadi $a + b \in I$. Jika $a \in I$ dan $r \in R$ maka $\varphi(a) \in J$ dan $\varphi(r) \in S$, jadi

$$\varphi(ra) = \varphi(r) \cdot \varphi(a) \in J$$

jadi $ar \in I$, dan I merupakan *ideal* dalam R. Karena $0_S \in J$ maka $\ker(\varphi) = \{c | c \in R, \varphi(c) = 0_S\} \subseteq I$.

Sebaliknya apakah setiap *ideal* I dalam R dipetakan oleh φ menjadi suatu *ideal* dalam S? Ternyata ini hanya bisa dipastikan bila φ *surjective* sehingga "mengisi penuh" S, karena jika tidak, ada pertanyaan dengan elemen S yang diluar target φ apabila dikalikan dengan elemen dari $J = \{\varphi(c) | c \in I\}$.

$$\text{Jika } \varphi \textit{ surjective maka } J = \{\varphi(c) | c \in I\} \text{ adalah } \textit{ideal} \text{ dalam } S. \tag{5.2}$$

Mari kita buktikan 5.2. Kita tahu bahwa J tidak kosong (karena I tidak kosong), jadi jika $b_1, b_2 \in J$, maka terdapat $a_1, a_2 \in I$ dimana $b_1 = \varphi(a_1)$ dan $b_2 = \varphi(a_2)$, jadi

$$b_1 + b_2 = \varphi(a_1) + \varphi(a_2) = \varphi(a_1 + a_2) \in J$$

karena I adalah suatu *ideal* ($a_1 + a_2 \in I$). Jika $b \in J$ dan $s \in S$ maka terdapat $a \in I$ dan $r \in R$ dimana $b = \varphi(a)$ dan $s = \varphi(r)$, jadi

$$sb = \varphi(r) \cdot \varphi(a) = \varphi(ra) \in J.$$

Jadi J merupakan *ideal* dalam S.

Kita sudah buktikan bahwa jika *homomorphism* φ dari *ring* R ke *ring* S *surjective*, maka setiap *ideal* I dalam R dipetakan oleh φ menjadi suatu *ideal* dalam S. Apakah hubungan antara *ideal* dalam R dengan *ideal* dalam S jika φ *surjective*? Ternyata jika φ *surjective*, ada korespondensi satu dengan satu antara himpunan semua *ideal* yang mencakup $\ker(\varphi)$ dalam R dengan himpunan semua *ideal* dalam S. Mari kita buat

$$\begin{aligned} I_\varphi &= \{I | I \textit{ ideal} \text{ dalam } R, \ker(\varphi) \subseteq I\}, \\ I_S &= \{J | J \textit{ ideal} \text{ dalam } S\}. \end{aligned}$$

Dengan pemetaan $\chi : I_\varphi \longrightarrow I_S$ sebagai berikut

$$\begin{aligned} \chi : I_\varphi &\longrightarrow I_S \\ I &\mapsto \{\varphi(a) | a \in I\} \end{aligned}$$

jika φ *surjective* maka

$$\chi \textit{ bijective} \text{ dengan } \chi^{-1}(J) = \{a | a \in R, \varphi(a) \in J\}. \tag{5.3}$$

Definisi χ diatas mengatakan bahwa untuk $I \in I_\varphi$, $\chi(I)$ adalah *image* dari I menurut φ dan notasi $\varphi(I)$ kerap digunakan walaupun notasi ini agak membingungkan. Untuk χ^{-1} definisi diatas mengatakan bahwa untuk $J \in I_S$, $\chi^{-1}(J)$ adalah *inverse image* dari J menurut φ dan notasi $\varphi^{-1}(J)$ kerap digunakan.

Pembuktian 5.3 mempunyai dua bagian:

- membuktikan bahwa $\chi^{-1}(\chi(I)) = I$ untuk setiap $I \in I_\varphi$, dan
- membuktikan bahwa $\chi(\chi^{-1}(J)) = J$ untuk setiap $J \in S$.

Walaupun teori mengenai pemetaan dengan konsep *image* dan *inverse image* dapat mempersingkat pembuktian, teori tersebut tidak akan digunakan, jadi kita akan membuktikan secara langsung.

- Kita tunjukkan bahwa $I \subseteq \chi^{-1}(\chi(I))$ untuk setiap $I \in I_\varphi$:

$$\begin{aligned} b \in I &\implies \varphi(b) \in \{\varphi(a) | a \in I\} \\ &\implies b \in \{a_1 | a_1 \in R, \varphi(a_1) \in \{\varphi(a) | a \in I\}\} \\ &\implies b \in \{a_1 | a_1 \in R, \varphi(a_1) \in \chi(I)\} \\ &\implies b \in \chi^{-1}(\chi(I)). \end{aligned}$$

  Jadi $I \subseteq \chi^{-1}(\chi(I))$ untuk setiap $I \in I_\varphi$. Berikutnya, kita tunjukkan bahwa $\chi^{-1}(\chi(I)) \subseteq I$ untuk setiap $I \in I_\varphi$:

$$\begin{aligned} b \in \chi^{-1}(\chi(I)) &\implies b \in \{a_1 | a_1 \in R, \varphi(a_1) \in \chi(I)\} \\ &\implies b \in \{a_1 | a_1 \in R, \varphi(a_1) \in \{\varphi(a) | a \in I\}\} \\ &\implies \varphi(b) \in \{\varphi(a) | a \in I\}\} \\ &\implies \text{terdapat } a \in I \text{ dimana } \varphi(b) = \varphi(a) \\ &\implies (b - a) \in \ker(\varphi) \subseteq I \\ &\implies b = (a + (b - a)) \in I. \end{aligned}$$

  Jadi $\chi^{-1}(\chi(I)) \subseteq I$ untuk setiap $I \in I_\varphi$. Menggabungkan kedua cakupan, kita dapatkan $\chi^{-1}(\chi(I)) = I$ untuk setiap $I \in I_\varphi$.

- Kita tunjukkan bahwa $\chi(\chi^{-1}(J)) \subseteq J$ untuk setiap $J \in S$:

$$\begin{aligned} b \in \chi(\chi^{-1}(J)) &\implies b \in \{\varphi(a) | a \in \chi^{-1}(J) \\ &\implies b \in \{\varphi(a) | a \in \{a_1 | a_1 \in R, \varphi(a_1) \in J\} \\ &\implies \text{terdapat } a \in R, \varphi(a) \in J \text{ dimana } b = \varphi(a) \\ &\implies b \in J. \end{aligned}$$

  Jadi $\chi(\chi^{-1}(J)) \subseteq J$ untuk setiap $J \in S$. Berikutnya, kita tunjukkan bahwa $J \subseteq \chi(\chi^{-1}(J))$ untuk setiap $J \in S$ dan φ *surjective*:

$$\begin{aligned} b \in J \text{ dan } \varphi \text{ surjective} &\implies \text{terdapat } I \in I_\varphi, a \in I \text{ dimana } b = \varphi(a) \\ &\implies a \in I, b = \varphi(a) \text{ dan} \\ &\qquad a \in \{a_1 | a_1 \in R, \varphi(a_1) \in J\} \\ &\implies a \in I, b = \varphi(a) \text{ dan } a \in \chi^{-1}(J) \\ &\implies b \in \chi(\chi^{-1}(J)) \end{aligned}$$

  Jadi $J \subseteq \chi(\chi^{-1}(J))$ untuk setiap $J \in S$. Menggabungkan kedua cakupan, kita dapatkan $\chi(\chi^{-1}(J)) = J$ untuk setiap $J \in I_S$.

Jadi kita selesai dengan pembuktian 5.3, dan selesai sudah pembahasan mengenai bagaimana *ideal* dipetakan oleh *homomorphism* secara umum.

Sekarang kita sudah dapat menjelaskan lebih lanjut, menggunakan hasil 5.3, *canonical homomorphism* φ dari *ring* R ke ring R/I, dimana I adalah suatu *ideal* dalam R. R/I disebut *quotient ring* modulo suatu *ideal*, dan setiap elemen dalam R/I adalah suatu *congruence class* $a + I$ untuk suatu a (kerap juga disebut *coset*). Untuk *ring* R, *proper ideal* I dalam R, dan *canonical homomorphism* φ dari R ke R/I, kita dapatkan:

$$I = \ker(\varphi). \tag{5.4}$$

Pembuktian 5.4 adalah sebagai berikut:

$$\begin{aligned} a \in \ker(\varphi) &\iff \varphi(a) = 0 \\ &\iff a + I = 0 + I \\ &\iff a \in I. \end{aligned}$$

Akibatnya, berdasarkan prinsip *extensionality* dari teori himpunan, $I = \ker(\varphi)$.

Teorema 14 *Untuk ring R, proper ideal I dalam R, dan φ canonical homomorphism dari R ke R/I dengan*

$$\begin{aligned} I_\varphi &= \{I_1 | I_1 \text{ ideal dalam } R, I \subseteq I_1\}, \\ I_S &= \{J | J \text{ ideal dalam } S\}. \end{aligned}$$

dan pemetaan $\chi : I_\varphi \longrightarrow I_S$ sebagai berikut

$$\begin{aligned} \chi : I_\varphi &\longrightarrow I_S \\ I_1 &\mapsto \{\varphi(a) | a \in I_1\} \end{aligned}$$

kita dapatkan

$$\chi \text{ bijective, dengan } \chi^{-1}(J) = \{a | a \in R, \varphi(a) \in J\}.$$

Teorema 14 adalah aplikasi dari 5.3 dengan $S = R/I$ dan menggunakan 5.4. Teorema mengatakan ada korespondensi satu dengan satu antara himpunan *ideal* dalam R yang mencakup $\ker(\varphi)$ dengan himpunan *ideal* dalam R/I. Lebih terperinci lagi, setiap *ideal* dalam R yang mencakup I, dipasangkan secara unik oleh fungsi χ dengan satu *ideal* dalam R/I, dan tidak ada *ideal* dalam R/I yang tidak dipasangkan. Teorema ini akan digunakan dalam pembahasan *polynomial field* yaitu *polynomial ring* modulo *ideal* dengan *generator irreducible polynomial*.

Yang terahir dibagian ini adalah teorema mengenai *ideal* dalam *field*.

Teorema 15 *Suatu ring R adalah field jika dan hanya jika ($\Longleftrightarrow$) tidak ada ideal untuk R selain $\{0\}$ dan R.*

Pembuktian teorema 15 mempunyai dua bagian. Di bagian pertama kita tunjukkan bahwa untuk *field* R hanya ada dua *ideal* yaitu $\{0\}$ dan R. Di bagian kedua kita tunjukkan bahwa jika *ring* R hanya mempunyai dua *ideal* $\{0\}$ dan R, maka R adalah suatu *field*.

- Mari kita umpamakan bahwa R adalah suatu *field* dan kita cari *ideal* untuk R selain $\{0\}$ dan R, sebut saja I. Ini berarti I harus berupa *proper ideal* dan *non-trivial*. Untuk *proper ideal*, kita tahu bahwa $1 \notin I$ karena jika $1 \in I$, seluruh ring R akan masuk dalam *ideal* berdasarkan *inside-outside multiplication*. Akibatnya, untuk setiap *unit* u dari R, $u \notin I$ sebab $u \in I$ berarti $u \cdot u^{-1} = 1 \in I$. Akan tetapi untuk *field* setiap elemen kecuali 0 merupakan *unit*, jadi hanya ada satu *proper ideal* untuk R yaitu $\{0\}$ yang merupakan *trivial ideal*. Berarti untuk *field* R hanya ada dua *ideal*: $\{0\}$ dan R.

- Untuk kebalikannya, kita umpamakan hanya $\{0\}$ dan R merupakan *ideal* dalam R. Jika ada *non-unit* $a \neq 0$, maka a dapat digunakan sebagai *generator* untuk membuat *ideal* aR yang tidak mengandung *unit* (kelipatan *non-unit* a tidak bisa berupa *unit*). Jadi aR harus berupa *non-trivial proper ideal* dalam R, suatu kontradiksi jika hanya $\{0\}$ dan R merupakan *ideal* dalam R. Jadi setiap elemen kecuali 0 berupa *unit*, yang berarti R harus berupa *field*.

Selesai sudah pembuktian teorema 15.

5.3 Principal Ideal Domain

Jika setiap *ideal* dalam suatu *ring* R merupakan *principal ideal* maka R dinamakan *principal ideal ring*. Jika R juga merupakan *integral domain* maka R merupakan *principal ideal domain* (PID).

$\mathbf{Z}$ merupakan *integral domain* karena dalam aritmatika bilangan bulat tidak ada *zero divisor*. Mari kita coba buktikan bahwa $\mathbf{Z}$ merupakan PID, jadi kita harus buktikan bahwa setiap *ideal* I dalam $\mathbf{Z}$ adalah *principal ideal*. Jika $I = \{0\}$ maka I adalah *principal ideal* dengan *generator* 0. Jika $I \neq \{0\}$, mari kita fokus pada himpunan

$$I^+ = \{m \in I | m > 0\}.$$

I^+ merupakan himpunan non-kosong karena $I \neq \{0\}$ dan $k \in I$ berarti $-k \in I$ untuk setiap $k \in \mathbf{Z}$. Berdasarkan prinsip *well-ordering*, I^+, yang merupakan subset dari $\mathbf{N}$, mempunyai elemen terkecil, sebut saja n. Karena $n \in I$ maka $n\mathbf{Z} \subseteq I$. Untuk kebalikannya, mari kita analisa apa konsekuensi dari $m \in I$. Kita gunakan algoritma pembagian untuk membagi m dengan n menghasilkan

$$m = nq + r$$

dengan $0 \leq r < n$. Tetapi $r = m - nq \in I$, jadi $r = 0$ karena n adalah minimal dalam I^+. Jadi $m = nq$ yang berarti $m \in n\mathbf{Z}$ yang juga berarti $I \subseteq n\mathbf{Z}$. Jadi $I = n\mathbf{Z}$ yang berarti I adalah *principal ideal* dengan *generator* n. Jadi $\mathbf{Z}$ adalah suatu PID.

Sekarang kita bahas konsep gcd abstrak dalam *integral domain* dengan definisi gcd sebagai berikut:

Definisi 12 (GCD untuk ring) *Untuk $a, b \in R$ dimana R adalah ring, d adalah* gcd *dari a dan b jika*

1. *$d|a$ dan $d|b$ (a dan b merupakan kelipatan d), dan*
2. *$d'|a$ dan $d'|b$ berarti $d'|d$.*

Menggunakan notasi logika:

$$\forall a, b, d \in R : d|a \wedge d|b \wedge (\forall d' \in R : (d'|a \wedge d'|b) \Longrightarrow d'|d) \Longrightarrow d \equiv \gcd(a, b).$$

Kita gunakan $d \equiv \gcd(a, b)$ untuk mengatakan "d adalah gcd dari a dan b" karena bisa terdapat banyak gcd tetapi semua ekuivalen karena berasosiasi. Untuk $a, b \in R$ dimana R adalah suatu *ring*, u suatu *unit* dari R, dan $a = ub$, a disebut *associated* dengan b (a dan b berasosiasi). Jika d adalah gcd dari a dan b dan d' juga merupakan gcd dari a dan b, maka d dan d' berasosiasi. Untuk membuktikan ini, definisi gcd mengatakan $d|a$, $d|b$, $d'|a$ dan $d'|b$, jadi karena syarat 2 dari definisi, $d|d'$ dan $d'|d$. Akibatnya, terdapat $a, b \in R$ dimana $d' = ad$ dan $d = bd'$. Jadi

$$\begin{aligned} d|d' \text{ dan } d'|d &\Longrightarrow d' = ad = abd' \\ &\Longrightarrow ab = 1 \end{aligned}$$

yang berarti $b = a^{-1}$, jadi a dan a^{-1} adalah *unit* dalam R. Jadi d dan d' berasosiasi.

Jika R adalah *integral domain* dan $a, b, d \in R$, maka kedua proposisi sebagai berikut ekuivalen:

1. d adalah gcd dari a dan b dan terdapat $s, t \in R$ dengan $d = sa + tb$.
2. $aR + bR = dR$.

Pembuktian bahwa proposisi 1 ekuivalen proposisi 2 mempunyai dua bagian:

- Kita tunjukkan bahwa proposisi 1 $\Longrightarrow$ proposisi 2. Karena d adalah gcd dari a dan b, maka $d|a$ dan $d|b$.

$$\begin{aligned} d|a \text{ dan } d|b &\Longrightarrow aR \subseteq dR \text{ dan } bR \subseteq dR \\ &\Longrightarrow aR + bR \subseteq dR. \end{aligned}$$

Karena terdapat $s, t \in R$ dengan $d = sa + tb$

$$\begin{aligned} d = sa + tb &\implies d \in aR + bR \\ &\implies dR \subseteq aR + bR. \end{aligned}$$

Jadi $aR + bR = dR$.

- Kita tunjukkan bahwa proposisi 2 $\implies$ proposisi 1.

$$\begin{aligned} aR + bR = dR &\implies aR + bR \subseteq dR \\ &\implies aR \subseteq dR \text{ dan } bR \subseteq dR \\ &\implies d|a \text{ dan } d|b. \end{aligned}$$

Juga

$$\begin{aligned} aR + bR = dR &\implies dR \subseteq aR + bR \\ &\implies d \in dR \subseteq aR + bR \\ &\implies \text{terdapat } s, t \in R \text{ dengan } d = sa + tb. \end{aligned}$$

Jika ada $d' \in R$ dengan $d'|a$ dan $d'|b$, karena $d = sa + tb$ maka $d'|d$. Jadi d adalah gcd dari a dan b.

Selesai sudah pembuktian bahwa proposisi 1 ekuivalen dengan proposisi 2. Karena PID merupakan *integral domain*, ini menghasilkan teorema berikut.

Teorema 16 *Jika R adalah PID dan $a, b \in R$ maka a dan b mempunyai* gcd *$d \in R$ dengan $aR+bR = dR$ jadi ada $s, t \in R$ dengan $d = sa+tb$. Menggunakan notasi logika:*

$$\forall a, b \in R : \exists d, s, t \in R : d \equiv \gcd(a, b) \wedge aR + bR = dR \wedge d = sa + tb.$$

5.4 Prime Ideal dan Maximal Ideal

Definisi 13 (Prime Ideal) *Suatu proper ideal I dalam ring R disebut prima (prime ideal) jika $ab \in I$ berarti $a \in I$ atau $b \in I$ untuk setiap $a, b \in R$.*

Untuk $2 \leq p \in \mathbf{Z}$, p adalah bilangan prima jika dan hanya jika ($\iff$) *ideal* $p\mathbf{Z}$ dalam $\mathbf{Z}$ adalah prima. Untuk $2 \leq p \in \mathbf{Z}$:

$$\begin{aligned} p \text{ prima} &\iff p|mn \text{ berarti } p|m \text{ atau } p|n \text{ untuk setiap } m, n \in \mathbf{Z} \\ &\iff mn \in p\mathbf{Z} \text{ berarti } m \in p\mathbf{Z} \text{ atau } n \in p\mathbf{Z} \text{ untuk setiap } m, n \in \mathbf{Z} \\ &\iff p\mathbf{Z} \text{ adalah } \textit{ideal} \text{ prima dalam } \mathbf{Z}. \end{aligned}$$

Jadi ada korespondensi satu dengan satu antara bilangan prima dalam $\mathbf{Z}$ dengan *non-trivial ideal* prima dalam $\mathbf{Z}$.

Definisi 14 (Maximal Ideal) *Suatu ideal I dalam ring R disebut maksimal (maximal ideal) jika $I \neq R$ dan untuk setiap ideal J dalam R yang mencakup I ($I \subseteq J$), $I = J$ atau $J = R$ (I merupakan proper ideal dalam R yang tidak tercakup oleh proper ideal dalam R lainnya).*

Teorema 17 *Jika I adalah proper ideal dalam R, maka:*

- *I prima $\iff$ R/I adalah suatu integral domain.*
- *I maksimal $\iff$ R/I adalah suatu field.*

Mari kita buktikan teorema 17. Jika I adalah *proper ideal* dalam R, maka

$$\begin{aligned} ab \in I &\iff ab + I = 0 \\ &\iff (a+I)(b+I) = 0. \end{aligned}$$

Untuk bagian pertama, pembuktian cukup mudah:

$$\begin{aligned} I \text{ prima} &\iff \forall a, b \in R : ab \in I \implies a \in I \text{ atau } b \in I \\ &\iff \forall a, b \in R : (a+I)(b+I) = 0 \implies a + I = 0 \text{ atau } b + I = 0 \\ &\iff R/I \text{ adalah } \textit{integral domain}. \end{aligned}$$

Untuk bagian kedua, menggunakan teorema 14:

$$\begin{aligned} I \text{ maksimal} &\iff \text{tidak ada } \textit{ideal} \supseteq I \text{ dalam } R \text{ kecuali } R \text{ dan } I \\ &\iff \text{tidak ada } \textit{ideal} \text{ dalam } R/I \text{ kecuali } R/I \text{ dan } \{I\} \\ &\iff \text{tidak ada } \textit{ideal} \text{ dalam } R/I \text{ kecuali } R/I \text{ dan } \{0_{R/I}\} \\ &\iff R/I \text{ adalah } \textit{field}. \end{aligned}$$

Apa kaitan *ideal* prima dengan *ideal* maksimal? Untuk setiap *ring* R dan *ideal* I dalam R

$$I \text{ maksimal} \implies I \text{ prima}. \tag{5.5}$$

Dengan kata lain, *ideal* yang maksimal juga merupakan suatu *ideal* prima. Untuk membuktikan ini, menggunakan teorema 17, I maksimal berarti R/I adalah suatu *field* yang juga berarti R/I adalah suatu *integral domain* yang berarti I prima.

Kebalikannya tidak selalu benar. Tidak semua *ideal* yang prima juga merupakan *ideal* maksimal. Tetapi untuk *principal ideal domain*, kita dapatkan hasil yang cukup memuaskan. Untuk setiap PID R dan *non-trivial ideal* I dalam R

$$I \text{ prima} \implies I \text{ maksimal}. \tag{5.6}$$

Setiap *non-trivial ideal* prima dalam PID juga merupakan *ideal* maksimal. Untuk membuktikan ini, karena R merupakan PID, kita cukup membuktikan

bahwa setiap *ideal* prima aR dengan $a \neq 0$ adalah *ideal* maksimal. Jadi jika bR adalah *ideal* dalam R dan $aR \subseteq bR$, maka kita harus tunjukkan bahwa $bR = aR$ atau $bR = R$.

$$\begin{aligned} aR \subseteq bR &\implies a \in bR \\ &\implies a = bc \text{ untuk suatu } c \in R \\ &\implies b \in aR \text{ atau } c \in aR \text{ (karena } aR \text{ prima).} \end{aligned}$$

Untuk $b \in aR$

$$\begin{aligned} b \in aR &\implies bR \subseteq aR \\ &\implies bR = aR. \end{aligned}$$

Untuk $c \in aR$

$$\begin{aligned} c \in aR &\implies c = ad \text{ untuk suatu } d \in R \\ &\implies a = bc = bad = abd \\ &\implies bd = 1 \\ &\implies 1 \in bR \\ &\implies bR = R. \end{aligned}$$

Jadi $bR = aR$ atau $bR = R$.

Jadi, dengan menggabungkan 5.5 dengan 5.6 kita dapatkan

Teorema 18 *Untuk setiap R yang berupa PID dan I suatu non-trivial ideal dalam R*

$$I \text{ prima} \iff I \text{ maksimal.}$$

Jadi untuk PID, konsep *ideal* prima dan konsep *ideal* maksimal menjadi satu.

Selanjutnya, kita definisikan konsep elemen prima dan elemen *irreducible* dalam *integral domain* yang akan kita kaitkan dengan konsep *ideal* prima dan *ideal* maksimal.

Definisi 15 *Untuk suatu integral domain R dan $0 \neq a$ suatu non-unit dalam R:*

- *a irreducible jika $a = bc$ berarti b adalah unit atau c adalah unit untuk setiap $b, c \in R$.*
- *a prima jika $a|bc$ berarti $a|b$ atau $a|c$ untuk setiap $b, c \in R$.*

Jika $a \in R$ dan u adalah suatu *unit* dalam R, maka a dapat diuraikan secara *trivial* menjadi $a = u(u^{-1}a)$. Jadi elemen *irreducible* adalah elemen yang tidak dapat diuraikan kecuali secara *trivial*. Jika a *irreducible* dan u unit, maka ua

juga *irreducible* karena ua tetap tidak dapat diuraikan kecuali secara *trivial*. Kebalikannya juga berlaku, jika ua *irreducible* maka a juga *irreducible*, sebab jika a *reducible* berarti terdapat *non-unit* $b, c \in R$ dengan $a = bc$ jadi terdapat *non-unit* ub dan c dengan $ua = (ub)c$ yang berarti ua juga *reducible*. Jadi untuk R suatu *integral domain*, $a \in R$ dan *unit* $u \in R$:

$$ua \text{ irreducible} \iff a \text{ irreducible} \tag{5.7}$$

Untuk R suatu *integral domain*,

$$a \text{ prima} \implies a \text{ irreducible}. \tag{5.8}$$

Untuk membuktikan 5.8, dengan a prima, mari kita lihat apakah mungkin a merupakan elemen yang *reducible*.

$$\begin{aligned} a \text{ reducible} &\implies \text{terdapat } \textit{non-unit } b, c : a = bc \\ &\implies a|b \text{ atau } a|c \end{aligned}$$

karena a prima dan $a|a$. Jika $a|b$, maka

$$\begin{aligned} a|b &\implies \exists d \in R : b = ad \\ &\implies \exists d \in R : a = bc = adc \\ &\implies \exists d \in R : dc = 1, \end{aligned}$$

sesuatu yang tidak mungkin karena c *non-unit*. Untuk menunjukkan bahwa $a|c$ juga sesuatu yang tidak mungkin, karena simetris, cara pembuktian sama tetapi dengan b dan c dipertukarkan, jadi tidak perlu diulang (cara pembuktian dimana kita cukup membuktikan satu dari beberapa pilihan karena cara pembuktian untuk pilihan lainnya serupa sering dijuluki *without loss of generality*). Jadi selesai sudah pembuktian 5.8.

Jika R suatu *integral domain* dan $a \in R$ *irreducible*

$$\forall b \in R : a \not| b \implies 1 \text{ adalah gcd dari } a \text{ dan } b \tag{5.9}$$

Untuk membuktikan bahwa 1 adalah gcd dari a dan b, jelas $1|a$ dan $1|b$ berlaku. Kita tinggal membuktikan bahwa jika $d|a$ dan $d|b$ maka $d|1$ untuk setiap $d \in R$. Jika $d|a$ maka terdapat $c \in R$ dengan $a = cd$. Karena a *irreducible*, maka ada dua kemungkinan untuk d:

- d adalah *unit* atau
- d *irreducible* sebab jika d *reducible* maka a juga *reducible*.

Jika d *irreducible*, maka c harus berupa *unit*, sebab jika tidak, maka a menjadi *reducible*. Jadi $d = c^{-1}a$. Karena $d|b$ maka terdapat $r \in R$ dengan $b = rd = rc^{-1}a$, jadi $a|b$, suatu kontradiksi. Jadi d harus berupa *unit* yang berarti $d|1$.

Teorema 19 *Untuk R suatu PID dan $a \in R$:*

$$a \text{ irreducible} \Longrightarrow a \text{ prima.} \tag{5.10}$$

Kita harus buktikan

$$a \text{ irreducible} \Longrightarrow \forall b, c \in R : a|bc \Longrightarrow a|b \text{ atau } a|c.$$

Jadi kita harus tunjukkan

$$a \text{ irreducible}, a|bc, a \not| b \Longrightarrow a|c.$$

Menggunakan 5.9 kita dapatkan

$$\begin{aligned} a \text{ irreducible}, a|bc, a \not| b &\Longrightarrow 1 \text{ adalah gcd dari } a \text{ dan } b \\ &\Longrightarrow \text{terdapat } s, t \in R : 1 = sa + tb \\ &\Longrightarrow c = sac + tbc \end{aligned}$$

Karena $a|bc$, maka $a|tbc$, jadi $a|c = sac + tbc$. Kita selesai dengan pembuktian teorema 19. Menggabungkan teorema 19 dengan 5.8 kita dapatkan:

Teorema 20 *Untuk R suatu PID dan $a \in R$:*

$$a \text{ irreducible} \Longleftrightarrow a \text{ prima.} \tag{5.11}$$

Untuk R suatu *integral domain*, $a \in R$

$$a \text{ prima} \Longleftrightarrow aR \text{ } ideal \text{ prima.} \tag{5.12}$$

Pembuktian 5.12 adalah sebagai berikut:

$$\begin{aligned} a \text{ prima} &\Longleftrightarrow a|bc \Longrightarrow a|b \text{ atau } a|c \text{ untuk setiap } b, c \in R \\ &\Longleftrightarrow bc \in aR \Longrightarrow b \in aR \text{ atau } c \in aR \text{ untuk setiap } b, c \in R \\ &\Longleftrightarrow aR \text{ } ideal \text{ prima.} \end{aligned}$$

Teorema terahir sebelum kita bahas *polynomial ring* adalah sebagai berikut.

Teorema 21 *Untuk R suatu PID dan $a \in R$, a irreducible $\Longleftrightarrow$ aR maksimal.*

Pembuktian teorema ini adalah sebagai berikut:

$$\begin{aligned} a \text{ irreducible} &\Longleftrightarrow a \text{ prima (5.11)} \\ &\Longleftrightarrow aR \text{ } ideal \text{ prima (5.12)} \\ &\Longleftrightarrow aR \text{ } ideal \text{ maksimal (teorema 18)} \end{aligned}$$

5.5 Polynomial Ring

Jika R adalah suatu *ring*, maka $R[x]$ adalah *polynomial ring* dengan variabel x. Setiap elemen dari $R[x]$ adalah *polynomial* dengan variabel x dan koefisien dari *ring* R. Sebaliknya, setiap *polynomial* dengan variabel x dan koefisien dari *ring* R merupakan elemen dari $R[x]$.

Sebagai contoh, dengan *ring* untuk koefisien berupa *field* $K = \mathbf{Z}/3\mathbf{Z}$,

$$x^5 + 2x^3 + x^2 + 2$$

merupakan *polynomial* elemen $K[x]$ dengan *degree* (pangkat terbesar) 5. Suatu *polynomial* p dapat ditulis sebagai:

$$p = \sum_{i=0}^{n} a_i x^i$$

dimana n adalah *degree* dari p dan a_i adalah koefisien untuk suku dengan pangkat i, jadi setiap a_i adalah elemen dari *ring* R, dan $a_n \neq 0$. Sebetulnya a_i berlaku untuk setiap $i \in \mathbf{Z}$ tetapi $a_i = 0$ untuk $i > n$ dan $i < 0$. Untuk contoh diatas, $a_0 = 2$, $a_1 = 0$, $a_2 = 1$, $a_3 = 2$, $a_4 = 0$ dan $a_5 = 1$. Kita akan namakan fungsi *degree* deg, jadi $\deg(p) = n$.

Aritmatika dalam $R[x]$ adalah sebagai berikut:

- Pertambahan dilakukan dengan menjumlahkan semua suku dari kedua *polynomial* (suku dengan pangkat yang sama dijadikan satu dengan menjumlahkan koefisien). Sebagai contoh, dengan $R = \mathbf{Z}/3\mathbf{Z}$, jika $p_1 = x^5+2x^3+x^2+2$ dan $p_2 = x^4+2x^3+x^2$ maka $p_1+p_2 = x^5+x^4+x^3+2x^2+2$. Penjumlahan koefisien dilakukan dengan aritmatika R, dalam contoh menggunakan aritmatika modulo 3.

- Perkalian dilakukan dengan mengalikan setiap suku dari *polynomial* pertama dengan setiap suku dari *polynomial* kedua dan menjumlahkan semua hasil perkalian. Sebagai contoh, dengan $R = \mathbf{Z}/3\mathbf{Z}$, jika $p_1 = x^2 + 2x$ dan $p_2 = 2x + 1$ maka $p_1 \cdot p_2 = 2x^3 + (2 \cdot 2)x^2 + x^2 + 2x = 2x^3 + 2x^2 + 2x$. Lagi, aritmatika koefisien menggunakan aritmatika R.

Menggunakan notasi penjumlahan dengan

$$p_1 = \sum_{i=0}^{m} a_i x^i \text{ dan } p_2 = \sum_{j=0}^{n} b_j x^j$$

rumus pertambahan menjadi:

$$p_1 + p_2 \quad = \quad (\sum_{i=0}^{m} a_i x^i) + (\sum_{j=0}^{n} b_j x^j)$$

$$= \sum_{i=0}^{max(m,n)} (a_i + b_i)x^i.$$

Rumus untuk perkalian menjadi:

$$\begin{aligned} p_1 \cdot p_2 &= (\sum_{i=0}^{m} a_i x^i) \cdot (\sum_{j=0}^{n} b_j x^j) \\ &= \sum_{i=0}^{m} (a_i x^i \cdot (\sum_{j=0}^{n} b_j x^j)) \\ &= \sum_{i=0}^{m} \sum_{j=0}^{n} a_i b_j x^{i+j} \\ &= \sum_{i=0}^{m+n} \left(\sum_{j=0}^{min(i,m)} a_j b_{i-j} \right) x^i. \end{aligned}$$

Tidak terlalu sukar untuk menunjukkan bahwa aritmatika dalam $R[x]$ mempunyai struktur aljabar *ring*. Rumus untuk pertambahan dan perkalian menunjukkan bahwa hasil pertambahan dan perkalian adalah *polynomial* dalam $R[x]$ juga, jadi kita dapatkan *closure*. Dengan

$$p_3 = \sum_{k=0}^{q} c_k x^k$$

Associativity untuk +:

$$\begin{aligned} p_1 + (p_2 + p_3) &= (\sum_{i=0}^{m} a_i x^i) + ((\sum_{j=0}^{n} b_j x^j) + (\sum_{k=0}^{q} c_k x^k)) \\ &= ((\sum_{i=0}^{m} a_i x^i) + (\sum_{j=0}^{n} b_j x^j)) + (\sum_{k=0}^{q} c_k x^k) \\ &= (p_1 + p_2) + p_3. \end{aligned}$$

Identity untuk +:

$$p_1 + 0 = (\sum_{i=0}^{m} a_i x^i) + 0 = \sum_{i=0}^{m} a_i x^i = p_1.$$

Commutativity untuk $+$:

$$p_1 + p_2 = (\sum_{i=0}^{m} a_i x^i) + (\sum_{j=0}^{n} b_j x^j) = (\sum_{j=0}^{n} b_j x^j) + (\sum_{i=0}^{m} a_i x^i) = p_2 + p_1.$$

Inverse untuk $+$:

$$\begin{aligned} p_1 + (-p_1) &= (\sum_{i=0}^{m} a_i x^i) + (-(\sum_{i=0}^{m} a_i x^i)) \\ &= (\sum_{i=0}^{m} a_i x^i) + (\sum_{i=0}^{m} -a_i x^i) \\ &= \sum_{i=0}^{m} (a_i x^i - a_i x^i) \\ &= 0. \end{aligned}$$

Associativity untuk $\cdot$:

$$\begin{aligned} p_1 \cdot (p_2 \cdot p_3) &= (\sum_{i=0}^{m} a_i x^i) \cdot (\sum_{j=0}^{n} \sum_{k=0}^{q} b_j c_k x^{j+k}) \\ &= \sum_{i=0}^{m} \sum_{j=0}^{n} \sum_{k=0}^{q} a_i b_j c_k x^{i+j+k} \\ &= (\sum_{i=0}^{m} \sum_{j=0}^{n} a_i b_j x^{i+j}) \cdot (\sum_{k=0}^{q} c_k x^k) \\ &= (p_1 \cdot p_2) \cdot p_3. \end{aligned}$$

Identity untuk $\cdot$:

$$p_1 \cdot 1 = (\sum_{i=0}^{m} a_i x^i) \cdot 1 = \sum_{i=0}^{m} a_i x^i = p_1.$$

Commutativity untuk $\cdot$:

$$p_1 \cdot p_2 = \sum_{i=0}^{m} \sum_{j=0}^{n} a_i b_j x^{i+j} = \sum_{i=0}^{n} \sum_{j=0}^{m} b_i a_j x^{i+j} = p_2 \cdot p_1.$$

Distributivity:

$$p_1 \cdot (p_2 + p_3) = (\sum_{i=0}^{m} a_i x^i) \cdot ((\sum_{j=0}^{n} b_j x^j) + (\sum_{k=0}^{q} c_k x^k))$$

$$\begin{aligned} &= (\sum_{i=0}^{m}\sum_{j=0}^{n} a_i b_j x^{i+j}) + (\sum_{i=0}^{m}\sum_{k=0}^{q} a_i c_k x^{i+k}) \\ &= (p_1 \cdot p_2) + (p_1 \cdot p_3). \end{aligned}$$

Jadi $R[x]$ mempunyai struktur aljabar *ring*. Jika K adalah suatu *field*, maka $K[x]$ mempunyai struktur *integral domain* karena jika

$$p_1 = \sum_{i=0}^{m} a_i x^i$$

dan

$$p_2 = \sum_{j=0}^{n} b_j x^j$$

dengan $a_m, b_n \neq 0$, maka

$$p_1 p_2 = a_m b_n x^{m+n} + \sum_{i=0}^{m+n-1} (\sum_{j=0}^{min(i,m)} a_j b_{i-j}) x^i \neq 0,$$

jadi tidak ada *zero divisor*. Untuk bab ini, kita akan fokus pada *polynomial ring* $K[x]$ dimana K merupakan suatu *field*.

Untuk bilangan bulat, efek dari algoritma pembagian dirumuskan oleh teorema 4. Teorema serupa diperlukan untuk pembagian *polynomial* dalam $K[x]$.

Teorema 22 (Pembagian Polynomial) *Untuk setiap pasangan polynomial $f, g \in K[x]$ dengan $g \neq 0$, ada sepasang polynomial $q, r \in K[x]$ dimana $f = qg + r$, dan $deg(r) < deg(g)$ atau $r = 0$.*

Jika $\deg(f) < \deg(g)$, kita dapatkan $q = 0$ dan $r = f$ sesuai dengan teorema. Jika $\deg(f) \geq \deg(g)$, kita perlu lakukan algoritma *long division* sebagai berikut, menggunakan notasi penjumlahan untuk f dan g:

$$f = \sum_{i=0}^{m} a_i x^i \text{ dan } g = \sum_{i=0}^{n} b_i x^i$$

dengan $a_m, b_n \neq 0$ dan $m \geq n$. Berikut algoritma untuk *long division*:

1. $r \leftarrow f$; $q \leftarrow 0$.
2. $c \leftarrow \frac{a_m}{b_n}$ dimana $r = \sum_{i=0}^{m} a_i x^i$ sebelum langkah ini dilakukan.
3. $r \leftarrow r - cx^{n-m} g$.

4. $q \leftarrow q + cx^{n-m}$.

5. Jika $r = 0$ atau $\deg(r) < \deg(g)$, kita selesai dengan q, r yang diinginkan. Jika tidak, ulangi dari langkah 2.

Algoritma *long division* mempunyai proposisi invarian (*invariant*):

$$f \quad = \quad qg + r$$

yang berlaku setelah langkah 1 dan dipertahankan oleh langkah-langkah selanjutnya. Setelah algoritma selesai,

$$r = 0 \text{ atau } \deg(r) < \deg(g)$$

juga berlaku, jadi algoritma *long division* menghasilkan q, r yang sesuai dengan teorema 22. Kita tunjukkan bahwa pasangan ini unik, jadi andaikan pasangan q, r dan pasangan q', r' keduanya sesuai dengan teorema 22, kita harus tunjukkan bahwa $q = q'$ dan $r = r'$.

$$f = qg + r = q'g + r' \Longrightarrow (q - q')g = r' - r.$$

Karena $g \neq 0$ dan $K[x]$ merupakan *integral domain*, maka $q - q' \neq 0 \iff r' - r \neq 0$ (jadi $q - q' = 0 \iff r' - r = 0$). Jika $q - q' \neq 0$ dan $r' - r \neq 0$

$$\deg(r') < \deg(g) \text{ dan } \deg(r) < \deg(g) \Longrightarrow \deg(r' - r) < \deg(g),$$

sedangkan

$$(q - q')g = r' - r \Longrightarrow \deg(r' - r) = \deg(q - q') + \deg(g) \geq \deg(g),$$

suatu kontradiksi. Jadi $q - q' = 0$ dan $r' - r = 0$, yang berarti $q = q'$ dan $r = r'$.

5.6 Euclidean Domain

Kita ingin tunjukkan bahwa $K[x]$ merupakan suatu *principal ideal domain*. Untuk itu kita gunakan konsep *Euclidean domain*.

Definisi 16 (Euclidean Domain) *Suatu ring R disebut Euclidean domain jika R adalah suatu integral domain dan terdapat fungsi $\delta : R \setminus \{0\} \longrightarrow \mathbf{N}$ dengan ketentuan sebagai berikut:*

1. *$\delta(fg) \geq \delta(f)$ untuk setiap $f, g \in R$ dengan $f, g \neq 0$.*

2. *Untuk setiap $f, g \in R$ dengan $f, g \neq 0$ dan $\delta(f) \geq \delta(g)$, terdapat $s, t \in R$ dimana $f - sg = t$, dan $\delta(t) < \delta(g)$ atau $t = 0$.*

Fungsi δ dinamakan *abstract degree function*, dan untuk *polynomial ring* merupakan fungsi *degree* deg. Dari definisi dan namanya, kita bisa menyimpulkan bahwa algoritma Euclid dapat digunakan untuk mencari gcd dalam *Euclidean domain*.

Teorema 23 *Jika K adalah suatu field, maka polynomial ring $K[x]$ adalah suatu Euclidean domain.*

Syarat 1 dari *Euclidean domain* dengan mudah dipenuhi oleh $K[x]$ karena mengalikan suatu *polynomial* yang bukan 0, dengan *polynomial* yang juga bukan 0, tidak akan mengurangi *degree* dari *polynomial* pertama. Syarat 2 dipenuhi oleh teorema 22.

Teorema 24 *Setiap Euclidean domain merupakan principal ideal domain.*

Untuk membuktikan teorema ini, pertama kita beri nama R untuk *Euclidean domain*. Mari kita analisa pembuatan *ideal* I untuk R. Jika $I = \{0\}$, maka $I = 0 \cdot R$ merupakan *principal ideal* dengan *generator* 0. Jika $I \neq \{0\}$, maka himpunan

$$\{\delta(r) | 0 \neq r \in I\} \subseteq \mathbf{N}$$

tidak kosong dan mempunyai elemen terkecil, sebut saja m. Jadi ada elemen $a \in I$ dengan $\delta(a) = m$. Kita akan buktikan bahwa $I = aR$. Untuk $aR \subseteq I$, pembuktiannya jelas dari definisi *ideal* yaitu jika elemen *ideal* (dalam hal ini a) dikalikan dengan elemen *ring*, hasilnya tetap dalam *ideal*. Untuk kebalikannya, kita umpamakan $b \in I$. Definisi *Euclidean domain* mengatakan terdapat $s, t \in R$ dengan $b - sa = t$ dan $\delta(t) < \delta(a)$ atau $t = 0$. Karena $t = (b - sa) \in I$ dan m minimal, tidak mungkin $\delta(t) < \delta(a)$, jadi $t = 0$ dan $b = sa$ ($b \in aR$). Jadi $I \subseteq aR$ dan kita selesai membuktikan $I = aR$ yang berarti I adalah *principal ideal* dengan *generator* a, membuktikan teorema 24.

Konsep terahir yang kita bahas sebelum membahas *polynomial field* adalah konsep *unique factorization*. Sebelum menjelaskan teorema mengenai *unique factorization*, ada beberapa fakta mengenai *Euclidean domain* yang akan digunakan untuk membuktikan teorema. Untuk R suatu *Euclidean domain* dengan *abstract degree function* δ dan $0 \neq a \in R$:

1. Jika $a = bc$ adalah uraian non-trivial (b dan c adalah non-unit dalam R), maka $\delta(b) < \delta(a)$ dan $\delta(c) < \delta(a)$.

2. Jika $\delta(a) = 0$ maka a adalah unit dalam R.

3. Jika $\delta(a) = 1$ dan a adalah non-unit, maka a *irreducible*.

Untuk membuktikan fakta 1, karena simetris, kita cukup membuktikan $\delta(b) < \delta(a)$. Karena R merupakan *Euclidean domain*, kita mengetahui bahwa $\delta(b) \leq$

$\delta(a)$. Jika $\delta(b) = \delta(a)$, karena R merupakan *Euclidean domain*, terdapat $q, r \in R$ dengan

$$b = qa + r, \delta(r) < \delta(a) = \delta(b) \text{ atau } r = 0.$$

Jadi

$$\begin{aligned} r = b - qa = b - qbc = (1 - qc)b &\implies \delta(r) \geq \delta(b) \text{ atau } r = 0 \\ &\implies r = 0 \\ &\implies qc = 1 \\ &\implies c \text{ adalah unit,} \end{aligned}$$

suatu kontradiksi karena c adalah non-unit. Jadi $\delta(b) < \delta(a)$.

Untuk membuktikan fakta 2, karena R adalah *Euclidean domain*, terdapat $q, r \in R$ dengan

$$1 = qa + r, \delta(r) < \delta(a) \text{ atau } r = 0.$$

Jadi $r = 0$ karena tidak mungkin $\delta(r) < 0$, jadi $qa = 1$ yang berarti a adalah *unit*.

Untuk membuktikan fakta 3, jika a *non-unit* dan mempunyai uraian *non-trivial* $a = bc$, menurut fakta 1, $\delta(b) < \delta(a) = 1$ dan $\delta(c) < \delta(a) = 1$, jadi $\delta(b) = \delta(c) = 0$. Berdasarkan fakta 2, ini berarti b dan c adalah *unit* yang juga berarti a adalah *unit*, suatu kontradiksi. Jadi a tidak mempunyai uraian *non-trivial*, dan karena a *non-unit*, berarti a *irreducible*.

Teorema 25 (Unique Factorization) *Jika R suatu Euclidean domain,*

1. *Suatu non-unit $a \in R$ dapat diuraikan menjadi produk dari satu atau lebih faktor irreducible.*

2. *Jika*

   $$a = p_1 p_2 \dots p_m = q_1 q_2 \dots q_n$$

   dengan p_i, q_j irreducible untuk $1 \leq i \leq m$ dan $1 \leq j \leq n$, maka $m = n$ dan urutan faktor dapat diubah sehingga $p_i = u_i q_i$ dengan u_i berupa unit untuk setiap $1 \leq i \leq m = n$. Jadi untuk setiap i, p_i berasosiasi dengan q_i.

Untuk bagian pertama dari teorema 25, pembuktian menggunakan induksi dengan variabel induksi $k = \delta(a)$. Sebagai dasar induksi, $k = 1$ karena $\delta(a) = 0$ berarti a suatu *unit* (fakta 2 untuk *Euclidean domain*). Fakta 3 untuk *Euclidean domain* mengatakan bahwa a *irreducible* jika $\delta(a) = k = 1$, jadi uraian *non-trivial* menghasilkan satu faktor yaitu a sendiri. Sekarang kita tunjukkan langkah induksi untuk $k > 1$:

- Jika a *irreducible* kita selesai dengan uraian *non-trivial* untuk a terdiri dari satu faktor yaitu a sendiri.

- Jika tidak, berarti ada uraian *non-trivial* $a = bc$ dimana b dan c *non-unit* dan fakta 1 untuk *Euclidean domain* mengatakan $\delta(b) < \delta(a)$ dan $\delta(c) < \delta(a)$. Hipotesis induksi mengatakan b dan c masing-masing mempunyai uraian faktor *irreducible*, jadi kedua uraian dapat digabung menjadi uraian faktor *irreducible* untuk a.

Pembuktian bagian kedua dari teorema 25 menggunakan induksi dengan variabel induksi m dan dasar induksi $m = 1$. Jika $m = 1$ maka $n = 1$ karena jika $n > 1$ maka

$$p_1 = (q_1 \dots q_{n-1})q_n$$

merupakan uraian *non-trivial* untuk p_1, sesuatu yang tidak mungkin. Jadi $m = n = 1$ dan $p_1 = q_1$. Untuk langkah induksi, $m > 1$,

$$p_1 | p_1 \dots p_m \Longrightarrow p_1 | q_1 \dots q_n$$

dan karena p_1 *irreducible* maka p_1 prima menurut teorema 20 (*Euclidean domain* merupakan PID). Jadi terdapat j dengan $1 \leq j \leq n$ dimana $p_1 | q_j$. Dengan mengubah urutan jika perlu, kita dapatkan $p_1 | q_1$ dan karena p_1 dan q_1 keduanya *irreducible* berarti p_1 berasosiasi dengan q_1 (karena jika tidak maka terdapat uraian *non-trivial* $q_1 = p_1 r$, sesuatu yang tidak mungkin jika q_1 *irreducible*). Jadi terdapat *unit* u dimana $q_1 = up_1$. Setelah melakukan substitusi $q_1 = up_1$ dan menghilangkan p_1 dari persamaan, kita dapatkan

$$p_2 \dots p_m = uq_2 \dots q_n$$

dengan uq_2 *irreducible*. Hipotesis induksi menghasilkan $m = n$ dan dengan mengubah urutan jika perlu, p_i berasosiasi dengan q_i untuk $2 \leq i \leq m$. Selesai sudah pembuktian teorema 25.

Selain $K[x]$, $\mathbf{Z}$ juga merupakan *Euclidean domain*, dengan $\delta(a) = |a|$. Untuk setiap bilangan *irreducible* $a \in \mathbf{Z}$ terdapat bilangan *irreducible* $a' > 0$ dan *unit* u dimana

$$a = ua'.$$

Jika $a > 0$, kita gunakan $u = 1$ jadi $a' = a$. Jika $a < 0$, kita gunakan $u = -1$ jadi $a' = -a > 0$.

Jadi untuk $\mathbf{Z}$, setiap faktor *irreducible* dalam uraian berasosiasi dengan faktor *irreducible* positif. Setiap bilangan $a \neq 0$ *non-unit* dapat diuraikan sebagai berikut (dengan bagian 2 dari teorema 25 juga berlaku):

$$a = up_1 p_2 \dots p_n$$

dimana u adalah 1 atau -1 dan p_i adalah bilangan positif *irreducible* (jadi merupakan bilangan prima) untuk setiap $1 \leq i \leq n$. Fakta ini kerap disebut

sebagai *fundamental theorem of arithmetic*. Tentunya uraian dapat mengandung suatu bilangan prima lebih dari satu kali, sebagai contoh

$$20 = 2 \cdot 2 \cdot 5.$$

Untuk $K[x]$, konsep bilangan positif *irreducible* (bilangan prima) diganti oleh konsep *monic irreducible polynomial* dimana suku dengan pangkat tertinggi dalam *irreducible polynomial* mempunyai koefisien 1. Setiap *irreducible polynomial* berasosiasi dengan suatu *monic irreducible polynomial*, jadi setiap *polynomial* f dengan $\deg(f) > 0$ (jadi $f \neq 0$ bukan *unit*) dapat diuraikan sebagai berikut:

$$f = uf_1f_2 \dots f_n$$

dimana u adalah *unit* (jadi $u \in K$) dan f_i adalah *monic irreducible polynomial* untuk setiap $1 \leq i \leq n$. Untuk $K[x]$, setiap $a \in K$ adalah konstan dan merupakan *unit*.

Extended Euclidean algorithm dapat digunakan untuk *polynomial*, dengan input $f, g \in K[x]$ kita dapatkan $d, s, t \in K[x]$ dimana

$$d = fs + gt$$

dan d merupakan gcd dari f dan g. Tentunya kalkulasi *quotient* dan *residue* dilakukan menggunakan *long division* untuk *polynomial*. Jika hasil untuk d berupa konstan, maka f dan g koprima dan ini dapat digunakan untuk kalkulasi *inverse* modulo *irreducible polynomial* (lihat pembahasan kalkulasi *inverse* modulo bilangan yang koprima menggunakan *extended Euclidean algorithm* di bagian 3.5). Jika $d = 1$ maka *inverse* langsung didapat, sedangkan jika $d \neq 1$ merupakan konstan, maka

$$dd^{-1} = 1 = fsd^{-1} + gtd^{-1}$$

jadi kita tinggal kalikan s dan t dengan d^{-1}.

Bagaimana kita dapat memastikan bahwa suatu *monic polynomial* dengan koefisien dari suatu *finite field* merupakan *irreducible polynomial*? Seperti halnya dengan bilangan prima, algoritma deterministik untuk menentukan *irreducibility* tidak efisien, akan tetapi mempunyai kompleksitas *polynomial time*. Secara naif kita dapat mencoba membagi dengan setiap *monic polynomial* dengan *degree* tidak lebih dari setengah *degree polynomial* yang sedang diperiksa. Jika tidak ada yang dapat membagi maka *polynomial* yang diperiksa adalah *monic irreducible polynomial*. Untuk algoritma yang lebih efisien, silahkan membaca [bac96] (teorema 7.6.2), dimana pembuktiannya menggunakan konsep aljabar Berlekamp.

5.7 Polynomial Field

Kita mulai pembahasan *polynomial field* dengan teorema mengenai konstruksi *polynomial field* sebagai *quotient ring* dari *polynomial ring*.

Teorema 26 (Polynomial Field) *Jika K adalah suatu field dan $g(x)$ adalah irreducible polynomial dalam $K[x]$, maka $K[x]/g(x)K[x]$ adalah suatu field.*

Karena K merupakan *field*, teorema 23 mengatakan bahwa $K[x]$ adalah *Euclidean domain*, yang juga berarti bahwa $K[x]$ adalah PID (teorema 24). Karena $g(x)$ *irreducible*, teorema 21 mengatakan bahwa $g(x)K[x]$ adalah *ideal* maksimal. Jadi menurut bagian kedua teorema 17, $K[x]/g(x)K[x]$ adalah suatu *field*, dinamakan *polynomial field*.

Sebagai contoh, mari kita bahas *polynomial field* yang digunakan dalam algoritma enkripsi AES. *Field* untuk koefisien yang digunakan adalah $K = \mathbf{Z}/2\mathbf{Z}$, jadi aritmatika untuk koefisien adalah aritmatika modulo 2, dimana operasi pertambahan dan pengurangan menjadi operasi *exclusive or* dan operasi pengalian menjadi *logical and*. *Irreducible polynomial* yang digunakan adalah

$$g(x) = x^8 + x^4 + x^3 + x + 1$$

yang mempunyai *degree* 8, jadi operasi *polynomial field* adalah operasi terhadap data sebesar 8 bit yang diinterpretasikan sebagai *polynomial* dengan *degree* maksimum 7 (setiap bit merepresentasikan koefisien dari suatu suku).

Karena *polynomial field* merupakan *quotient ring* dari *polynomial ring*, aritmatika *polynomial field* mirip dengan aritmatika *polynomial ring* (lihat bagian 5.5). Pertambahan *polynomial* dalam *polynomial field* sama dengan pertambahan *polynomial* dalam *polynomial ring*. Untuk perkalian, hasil perkalian adalah *residue* (sisa setelah dibagi dengan *irreducible polynomial*) dari perkalian *polynomial* sebagai operasi *polynomial ring*. Algoritma *long division* dapat digunakan untuk mendapatkan residue. Kita gunakan

$$\begin{aligned} f_1 &= x^6 + x^4 + x^2 + x + 1 \\ f_2 &= x^7 + x + 1 \end{aligned}$$

dengan aritmatika *polynomial field* AES sebagai contoh. Untuk pertambahan,

$$\begin{aligned} f_1 + f_2 &= x^6 + x^4 + x^2 + x + 1 + x^7 + x + 1 \\ &= x^7 + x^6 + x^4 + x^2. \end{aligned}$$

Untuk perkalian, kita lakukan perkalian dalam *polynomial ring* $K[x]$ dahulu:

$$\begin{aligned} f_1 f_2 &= x^{13} + x^{11} + x^9 + x^8 + x^7 + \\ &\quad x^7 + x^5 + x^3 + x^2 + x + \\ &\quad x^6 + x^4 + x^2 + x + 1 \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \end{aligned}$$

lalu kita lakukan *long division* dengan $g(x)$ untuk mendapatkan *residue*.

$$\begin{aligned} r_0 = f_1 f_2 &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \\ x^5 g(x) &= x^{13} + x^9 + x^8 + x^6 + x^5 \\ r_1 = r_0 - x^5 g(x) &= x^{11} + x^4 + x^3 + 1 \\ x^3 g(x) &= x^{11} + x^7 + x^6 + x^4 + x^3 \\ r_2 = r_1 - x^3 g(x) &= x^7 + x^6 + 1 \end{aligned}$$

dan kita selesai karena r_2 tidak dapat dibagi oleh $g(x)$. Jadi untuk *polynomial field* AES, hasil perkalian menjadi

$$f_1 f_2 = r_2 = x^7 + x^6 + 1.$$

Dengan komputer, pertambahan untuk *polynomial field* AES dapat dilakukan secara sangat efisien menggunakan *bitwise exclusive or* dengan *operand* masing-masing 8 bit. Perkalian dapat dilakukan melalui kombinasi *shift* dan *exclusive or* dengan akumulator sebesar 16 bit. Kalkulasi *inverse* dapat dilakukan menggunakan *extended Euclidean algorithm* untuk *polynomial*.

Finite field, termasuk juga *polynomial field*, dinamakan juga *Galois field* dan diberi notasi **GF**. *Polynomial field* untuk AES diberi notasi $\mathbf{GF}(2^8)$ karena mempunyai 2^8 elemen.

5.8 Ringkasan

Tujuan utama dari bab ini adalah untuk menjelaskan *polynomial field*. Berbagai konsep digunakan untuk menjelaskan *polynomial field*, antara lain *homomorphism*, *ideal*, *principal ideal domain*, *polynomial ring* dan *Euclidean domain*. Konsep dan teorema yang berada dalam bab ini juga akan digunakan pada pembahasan *finite field* di bab 10.

Bab 6

Kriptografi Stream Cipher

Deretan kunci untuk enkripsi *one-time pad* dapat dipandang sebagai suatu *keystream* yang tidak mempunyai periode. Kriptografi *stream cipher* mencoba menggunakan konsep ini tetapi dengan *keystream* yang mempunyai periode dan menggunakan *generator* yang relatif pendek berupa kunci. Baik enkripsi *one-time pad* maupun *stream cipher* mendapatkan naskah acak dari *exclusive or* (XOR) naskah asli dengan *keystream*, jadi keduanya merupakan apa yang dinamakan *Vernam cipher* (lihat tabel 6.1) yang ditemukan oleh Gilbert Vernam tahun 1917. Bedanya hanya pada pembuatan *keystream*:

- Untuk *one-time pad*, *keystream* didapat langsung dari *key generation* (*random number generation*).
- Untuk *stream cipher*, *keystream* didapat dari *pseudo-random number generation* menggunakan kunci enkripsi.

10010111001011101001...	naskah asli
01001110001101001101...	*keystream*
11011001000110100100...	naskah acak

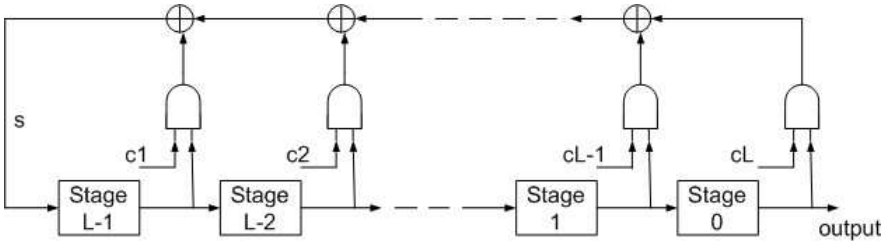
Tabel 6.1: *Vernam Cipher*

Berbeda dengan *keystream* untuk *one-time pad* yang tidak mempunyai periode, *keystream* untuk *stream cipher* mempunyai periode (kecuali jika naskah acak dijadikan *feedback*), meskipun periode sangat panjang. Ini karena *pseudo-random number generator* adalah suatu *deterministic finite state automaton*, jadi karena jumlah *state finite* dan kunci juga *finite*, maka setelah seluruh kunci diproses, banyaknya *state* yang dapat dikunjungi terbatas dan *automaton* akan

kembali ke *state* yang pernah dikunjungi, dan karena *automaton* bersifat *deterministic*, maka siklus akan diulang. Beberapa cara *pseudo-random number generation* untuk mendapatkan *keystream* antara lain:

- menggunakan *linear feedback shift register* (LFSR),
- menggunakan *block cipher* dalam *feedback mode* (lihat bagian 7.2), dan
- menggunakan *state automaton* dalam *software* (contohnya RC4).

Gambar 6.1 memperlihatkan suatu *linear feedback shift register* (LFSR). Setiap



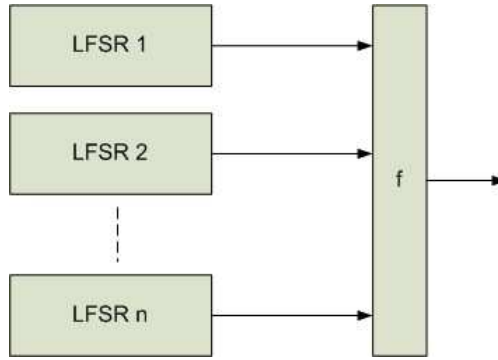
Gambar 6.1: Linear feedback shift register

stage i, $0 \leq i \leq L-1$ dipisahkan menggunakan *delay flip-flop*, atau disebut juga *latch*. Setiap $cL - i$ dimana $0 \leq i \leq L-1$ menentukan apakah *stage* i ditambahkan ke *feedback* s (modulo 2), jika $cL - i$ mempunyai nilai 1 maka *stage* i ditambahkan, sedangkan jika $cL - i$ mempunyai nilai 0 maka *stage* i tidak ditambahkan. Setiap *clock cycle*, *output* 1 bit didapat dari *stage* 0. Untuk membuat *pseudo-random number generator*, beberapa LFSR biasanya dikombinasikan menggunakan fungsi non-linear. Gambar 6.2 memperlihatkan kombinasi LFSR menggunakan fungsi non-linear f.

Beberapa kelemahan *stream cipher* antara lain:

- Jika naskah asli dan naskah acak diketahui, maka menggunakan xor kita bisa dapatkan *keystream*.
- *Stream cipher* rentan terhadap *tampering*. Seseorang yang mengetahui posisi data tertentu dalam naskah acak dan mengetahui nilai data tersebut bisa mengubahnya menggunakan xor. Contohnya, angka 50 bisa diubah menjadi angka 99 dengan melakukan $x \oplus (50 \oplus 99)$, dimana x merepresentasikan 50 dalam naskah acak.

Efek serupa dengan penggunaan kunci secara berulang pada enkripsi *one-time pad* juga mudah terjadi. Penggunaan *stream cipher* memang harus dengan sangat hati-hati, oleh sebab itu buku ini tidak merekomendasikan penggunaan *stream cipher*.



Gambar 6.2: Kombinasi non-linear LFSR

6.1 RC4

RC4 adalah *stream cipher* yang dirancang di RSA Security oleh Ron Rivest tahun 1987. Pada mulanya cara kerja RC4 dirahasiakan oleh RSA Security, akan tetapi ini dibocorkan di internet tahun 1994 di milis Cypherpunks. RSA Security tidak pernah merilis RC4 secara resmi, akibatnya banyak yang menyebutnya sebagai ARC4 (*alleged* RC4 atau tersangka RC4) untuk menghindari masalah *trademark*.

Berbeda dengan mayoritas *stream cipher* sebelumnya yang implementasinya dioptimalkan untuk *hardware* menggunakan *linear feedback shift registers*, RC4 dirancang agar dapat diimplementasikan di *software* secara sangat efisien. Ini membuat RC4 sangat populer untuk aplikasi internet, antara lain RC4 digunakan dalam standard TLS (*transport layer security*) dan WEP (*wireless equivalent privacy*).

Cara membuat *keystream* dalam RC4 adalah dengan *state automaton* dan terdiri dari dua tahap:

1. Tahap *key scheduling* dimana *state automaton* diberi nilai awal berdasarkan kunci enkripsi.

2. Tahap *pseudo-random generation* dimana *state automaton* beroperasi dan outputnya menghasilkan *keystream*.

Tahap pertama dilakukan menggunakan *key scheduling algorithm* (KSA). *State* yang diberi nilai awal berupa *array* yang merepresentasikan suatu permutasi dengan 256 elemen, jadi hasil dari algoritma KSA adalah permutasi awal. *Array* yang mempunyai 256 elemen ini (dengan indeks 0 sampai dengan 255) dinamakan `S`. Berikut adalah algoritma KSA dalam bentuk *pseudo-code* dimana

`key` adalah kunci enkripsi dan `keylength` adalah besar kunci enkripsi dalam *bytes* (untuk kunci 128 bit, `keylength` = 16):

```
for i = 0 to 255
  S[i] := i
j := 0
for i = 0 to 255
  j := (j + S[i] + key[i mod keylength]) mod 256
  swap(S[i],S[j])
```

Tahap kedua menggunakan algoritma yang dinamakan *pseudo-random generation algorithm* (PRGA). Setiap putaran, bagian *keystream* sebesar 1 byte (dengan nilai antara 0 sampai dengan 255) dioutput oleh PRGA berdasarkan *state* `S`. Berikut adalah algoritma PRGA dalam bentuk *pseudo-code*:

```
i := 0
j := 0
loop
  i := (i + 1) mod 256
  j := (j + S[i]) mod 256
  swap(S[i],S[j])
  output S[(S[i] + S[j]) mod 256]
```

Permutasi dengan 255 elemen mempunyai 255! kemungkinan. Ditambah dua indeks (i dan j) yang masing-masing dapat mempunyai nilai antara 0 dan 255, maka *state automaton* yang digunakan untuk membuat *keystream* mempunyai

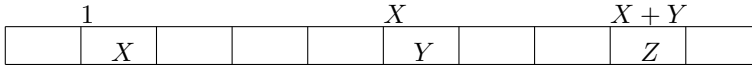
$$255! \times 255^2 \approx 2^{1700}$$

kemungkinan *internal states*. Karena banyaknya jumlah kemungkinan untuk *internal state*, sukar untuk memecahkan RC4 dengan menganalisa PRGA (teknik paling efisien saat ini harus menjajagi $> 2^{700}$ kemungkinan).

Karena algoritma KSA relatif sangat sederhana, banyak ahli kriptografi yang fokus pada KSA dalam mencari kelemahan enkripsi RC4. Fokus ini membuahkan hasil dengan ditemukannya beberapa kelemahan KSA yang dapat digunakan untuk memecahkan aplikasi RC4, termasuk memecahkan penggunaan RC4 dalam protokol WEP. Metode pertama yang digunakan untuk memecahkan WEP dikembangkan oleh Fluhrer, Mantin dan Shamir [flu01], kita sebut saja FMS *attack*. Implementasi pertama FMS *attack* dilaporkan di [stu01]. Berikut kita bahas enkripsi RC4 dalam WEP dan metode FMS *attack* yang dapat digunakan untuk memecahkannya.

WEP menggunakan gabungan *initialization vector* (IV) dan kunci rahasia (yang diketahui oleh pengirim dan penerima) untuk mendapatkan kunci RC4 (kunci enkripsi dimulai dengan IV diikuti oleh kunci rahasia). Untuk setiap paket, IV sebesar 3 byte digunakan bersama kunci rahasia untuk mengenkripsi

paket. IV dikirim tanpa terenkripsi bersama paket yang dienkripsi, jadi IV dapat diketahui oleh pemecah. Byte pertama dalam paket yang dienkripsi juga selalu diketahui, yaitu byte dengan nilai $0xAA$ (10101010) di-XOR dengan byte pertama output dari PRGA. Byte pertama output PRGA hanya tergantung pada 3 elemen *state* S saat KSA baru saja selesai sebagai berikut, dimana nilai byte output pertama dilabel Z:



Metode pemecahan berfokus pada kasus dimana 3 elemen tersebut diatas ditentukan oleh komponen kunci. Saat KSA berada pada tahap $i \geq 1$, maka $X = S_i[1]$ dan $X + Y = S_i[1] + S_i[S_i[1]]$. Jika kita modelkan pertukaran elemen sebagai acak, maka ada kemungkinan yang meskipun kecil tetapi cukup signifikan (probabilitas ≈ 0.05) bahwa ketiga elemen tersebut diatas tidak ditukar. Untuk WEP dimana IV sebesar 3 byte, kunci enkripsi terdiri dari $(IV[0], IV[1], IV[2], K[0], K[1], \ldots, K[l-1])$ dimana K adalah kunci rahasia sebesar l byte. Untuk mendapatkan byte $K[B]$ kunci rahasia, maka setelah 3 langkah pertama KSA kita butuhkan

$$S_3[1] < 3 \text{ dan } S_3[1] + S_3[S_3[1]] = 3 + B. \tag{6.1}$$

Dengan probabilitas sekitar 0.05, setelah langkah $3 + B$, ketiga elemen yang menentukan byte pertama output PRGA tidak ditukar lagi[1]. Jika syarat 6.1 dipenuhi, kemungkinan terbesar untuk nilai byte pertama output adalah:

$$Out = S_{3+B-1}[j_{3+B}] = S_{3+B-1}[j_{3+B-1} + K[B] + S_{3+B-1}[3+B]].$$

Karena nilai j_{3+B-1} dan $S_{3+B-1}[3+B]$ dapat diketahui, maka nilai $K[B]$ dapat diprediksi sebagai berikut, dimana $S_i^{-1}[X]$ adalah indeks untuk elemen S_i yang mempunyai nilai X:

$$K[B] = S_{3+B-1}^{-1}[Out] - j_{3+B-1} - S_{3+B-1}[3+B].$$

Dengan syarat 6.1 terpenuhi, prediksi ini benar sekitar 5% dari semua percobaan (jika ada 100 percobaan acak, maka 5 percobaan menghasilkan prediksi yang benar). Jika cukup banyak percobaan yang dilakukan maka hasil yang benar akan menonjol dibandingkan hasil-hasil lainnya. Untuk mendapatkan hasil yang baik, diperlukan sekitar 60 IV dengan nilai X yang berbeda dengan format

$$(B + 3, 255, X).$$

Format ini diperlukan untuk memenuhi syarat 6.1. Untuk mendapatkan seluruh kunci rahasia, pencarian dimulai dengan $B = 0$, kemudian $B = 1$ dan

[1]Kondisi ini disebut *resolved condition*.

seterusnya sampai dengan $B = 12$ untuk WEP. Eksperimen yang dilaporkan [stu01] menunjukkan bahwa sekitar 6 juta paket WEP diperlukan untuk mendapatkan kunci rahasia. Banyak jaringan Wi-Fi yang hanya menggunakan satu password untuk semua pengguna (jadi hanya ada satu kunci rahasia). Jika jaringan tersebut mempunyai kepadatan lalu-lintas yang moderat, kunci rahasia dapat ditemukan dalam waktu sehari.

Setelah FMS *attack* dipublikasikan, banyak peneliti yang mencoba menganalisa kelemahan WEP lebih lanjut. Ada beberapa peneliti yang berhasil memperkecil jumlah paket WEP yang diperlukan untuk mendapatkan rahasia sehingga WEP dapat dipecahkan dalam waktu yang cukup singkat (bisa dibawah 60 detik). Kita akan bahas dua *attack* yang kinerjanya cukup menakjubkan yaitu *chopchop attack* dan PTW *attack*.

Chopchop attack sebetulnya bukan *attack* untuk mendapatkan kunci WEP, tetapi merupakan *attack* untuk mendekripsi paket WEP tanpa mengetahui kunci WEP. *Attack* cerdik ini dikembangkan oleh seseorang yang menggunakan nama KoreK dan dipublikasikan dalam milis netstumbler (dapat diakses di website `http://www.netstumbler.org`). *Attack* ini didasarkan pada pengamatan bahwa dalam suatu jaringan WiFi, *access point* menindak-lanjuti suatu paket hanya jika paket tersebut lolos pengecekan CRC (*cyclic redundancy check*). Jika byte terahir dari paket dihilangkan, hampir dapat dipastikan bahwa CRC paket harus diubah. Bagaimana CRC harus diubah menentukan nilai asli dari byte yang dihilangkan. Dengan mencoba semua kemungkinan pengubahan CRC dan mengamati reaksi *access point* terhadap paket yang telah diubah CRC-nya, penyadap dapat mengetahui bagaimana CRC harus diubah, dan dengan demikian dapat mengetahui nilai asli byte. Setelah CRC dan nilai byte terahir didapat, penyadap dapat mengulang proses dengan juga menghilangkan byte kedua dari terahir, dan seterusnya. Matematika yang digunakan untuk CRC tentunya adalah aritmatika *polynomial field* (lihat bagian 5.7). Jika pesan diinterpretasikan sebagai *polynomial* P dalam $\mathbf{GF}(2)[x]$, maka

$$P \bmod R_{CRC} = \sum_{i=0}^{31} x^i$$

dimana R_{CRC} adalah *polynomial* untuk CRC32:

$$\begin{aligned} R_{CRC} &= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} \\ &\quad + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1. \end{aligned}$$

Karena R_{CRC} *irreducible*, maka aritmatika modulo R_{CRC} adalah aritmatika *polynomial field*. Pesan P dapat ditulis sebagai

$$P = Q \cdot x^8 + P_7$$

dimana P_7 adalah $P \bmod x^8$, jadi P_7 merepresentasikan nilai asli byte terahir. Karena P mempunyai *checksum* yang benar, maka

$$(Q \cdot x^8 + P_7) \bmod R_{CRC} = \sum_{i=0}^{31} x^i.$$

Menggunakan aritmatika *polynomial* modulo R_{CRC} kita bisa dapatkan $(x^8)^{-1}$, jadi

$$Q \equiv (x^8)^{-1}(P_7 + \sum_{i=0}^{31} x^i) \pmod{R_{CRC}}.$$

Jadi kita perlu koreksi Q menjadi Q' untuk mendapatkan *checksum* yang benar. Ini dapat dilakukan dengan menambahkan (modulo R_{CRC}) P_{CORR} ke Q, dimana

$$P_{CORR} = ((x^8)^{-1}(P_7 + \sum_{i=0}^{31} x^i) + \sum_{i=0}^{31} x^i) \bmod R_{CRC}.$$

Dengan mencoba semua kemungkinan nilai P_7 (dari 0 sampai dengan 255), dan mengamati reaksi dari *access point*, penyadap dapat menentukan nilai asli byte terahir.

PTW *attack* (Pyshkin, Tews dan Weinmann) (lihat [tew07]) dikembangkan dari analisa yang dilakukan oleh Andreas Klein (lihat [kle07]). Selain kelemahan pada KSA, analisa juga menunjukkan kecenderungan pada algoritma PRGA. Kecenderungan ini dirumuskan dalam teorema berikut yang membuat korelasi antara i dan *internal state*.

Teorema 27 *Jika probabilitas internal state untuk RC4 terdistribusi secara uniform, dengan setiap pilihan i, kita dapatkan berbagai probabilitas sebagai berikut:*

1. *Dengan $k = S[i] + S[j]$ dan $n = 256$, kita dapatkan*

$$P(S[j] + S[k] \equiv i \pmod{n}) = \frac{2}{n}.$$

2. *Untuk $c \not\equiv i \pmod{n}$, kita dapatkan*

$$P(S[j] + S[k] \equiv c \pmod{n}) = \frac{n-2}{n(n-1)}.$$

Pembuktian teorema 27 adalah sebagai berikut. Untuk bagian 1, kita buktikan bahwa untuk setiap $x \in \{0, 1, \ldots, 255\}$,

$$P(S[j] + S[k] \equiv i \pmod{n} \mid S[j] = x) = \frac{2}{n}.$$

Kita hitung banyaknya *internal state* dimana $S[j] + S[k] \equiv i \pmod{n}$ dan $S[j] = x$. Karena $k = (S[i] + S[j]) \bmod n$, maka $S[j] + S[k] \equiv i \pmod{n}$ dapat ditulis sebagai $k + S[k] \equiv i + S[i] \pmod{n}$ dengan penjelasan sebagai berikut:

$$\begin{aligned} S[j] + S[k] &\equiv i \pmod{n}, \\ k + S[j] + S[k] &\equiv S[i] + S[j] + i \pmod{n}, \\ k + S[k] &\equiv i + S[i] \pmod{n}. \end{aligned}$$

Kita bagi menjadi dua kasus yaitu kasus $i = k$ dan $i \neq k$.

1. Untuk $i = k$, karena $S[i] = S[k]$, tidak ada syarat lainnya, persamaan langsung tercapai. Untuk sisa $S[y]$ dimana $y \neq i = k$, terdapat $(n-1)!$ kemungkinan permutasi.

2. Untuk $i \neq k$, kita harus buat $S[k] = (i - x) \bmod n$ dan $S[i] = (k + S[k] - i) \bmod n$. Terdapat $n - 1$ pilihan untuk k dan sisanya (setelah i dan k terpilih) masih ada $(n-2)!$ kemungkinan permutasi. Jadi untuk $i \neq k$ ada $(n-1)(n-2)! = (n-1)!$ kemungkinan permutasi.

Total untuk $i = k$ dan $i \neq k$ terdapat $2(n-1)!$ kemungkinan permutasi, sedangkan *internal state* secara keseluruhan mempunyai $n!$ kemungkinan permutasi. Jadi probabilitas yang kita dapatkan adalah

$$\frac{2(n-1)!}{n!} = \frac{2}{n}.$$

Untuk bagian 2, kita buktikan bahwa untuk setiap $x \in \{0, 1, \ldots, 255\}$,

$$P(S[j] + S[k] \equiv c \pmod{n} \mid S[j] = x) = \frac{n-2}{n(n-1)}.$$

Kita bagi menjadi 3 kasus:

1. Kasus $i = k$. Karena $c \neq i = k$, maka tidak mungkin kondisi $k + S[k] \equiv c + S[i]$ tercapai, jadi kasus ini tidak memberi kontribusi terhadap penghitungan.

2. Kasus $c = k$. Kondisi $k + S[k] \equiv c + S[i] \pmod{n}$ juga tidak mungkin tercapai karena $S[k] \equiv S[i] \pmod{n}$ menjadi sesuatu yang tidak mungkin. Jadi kasus ini juga tidak memberi kontribusi terhadap penghitungan.

3. Kasus $i \neq k$ dan $c \neq k$. Untuk kasus ini, kita harus buat $S[k] = (i - x) \bmod n$ dan $S[i] = (k + S[k] - c) \bmod n$. Terdapat $n - 2$ pilihan untuk k dan sisanya (setelah i dan k terpilih) masih ada $(n-2)!$ kemungkinan permutasi. Jadi total ada $(n-2)(n-2)!$ kemungkinan permutasi.

Total terdapat $(n-2)(n-2)!$ kemungkinan permutasi, sedangkan *internal state* secara keseluruhan mempunyai $n!$ kemungkinan permutasi. Jadi probabilitas yang kita dapatkan adalah

$$\frac{(n-2)(n-2)!}{n!} = \frac{n-2}{n(n-1)}.$$

Selesailah pembuktian teorema 27.

Secara garis besar, PTW *attack* bertahap mencari $K[l]$ setelah mengetahui $K[0], K[1], \ldots, K[l-1]$. Karena $K[0], K[1], \ldots, K[l-1]$ diketahui, maka l putaran permutasi pertama menggunakan KSA dapat disimulasi. Kita gunakan notasi $S_m[x]$ untuk nilai $S[x]$ pada putaran m setelah permutasi dilakukan, dimana untuk putaran pertama $m = 1$. (Yang mungkin agak membingungkan adalah dalam putaran permutasi KSA, i mulai dari 0, sedangkan dalam putaran permutasi PRGA, i mulai dari 1.) Dengan simulasi, kita bisa dapatkan seluruh *internal state* S_l. Dalam putaran permutasi ke-$l+1$, nilai $S[l]$ dan nilai $S[(j + S[l] + K[l]) \bmod n]$ saling dipertukarkan. Tepatnya

$$S_{l+1}[l] = S_l[(j_l + S_l[l] + K[l]) \bmod n].$$

Untuk singkatnya, kita gunakan $t = S_{l+1}[l]$. Jika nilai t diketahui, karena simulasi bisa menghasilkan seluruh *internal state* S_l, maka nilai $K[l]$ dapat dicari dengan rumus

$$K[l] = (S_l^{-1}[t] - (j_l + S_l[l])) \bmod n$$

dimana $S_l^{-1}[t]$ adalah indeks untuk S_l yang memberikan t, dengan kata lain

$$S_l[S_l^{-1}[t]] = t.$$

Esensi dari PTW *attack* adalah mencari nilai t dari pengamatan output, yang kemudian digunakan untuk menentukan $K[l]$. Kita kaitkan t dengan output menggunakan persamaan

$$t \equiv l - X[l-1] \pmod{n}.$$

Berikutnya kita akan bahas probabilitas bahwa persamaan ini berlaku.

Setelah putaran $l+1$, maka terdapat $n-2$ putaran permutasi lagi sebelum i kembali mempunyai nilai l ($n-l-1$ putaran permutasi pada tahap KSA dan $l-1$ putaran permutasi pada tahap PRGA). Selama $n-2$ putaran itu, nilai $S[l]$ hanya akan berubah jika j menunjuknya. Setiap putaran, probabilitas $j = l$ adalah $1/n$, jadi probabilitas bahwa nilai $S[l]$ tidak berubah dalam $n-2$ putaran adalah

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

Probabilitas bahwa nilai $S[l]$ ditukar adalah $1 - (1 - 1/n)^{n-2}$. Untuk kasus dimana $S[l]$ tidak berubah ($S_{n+l-1}[l] = t$), menggunakan teorema 27 kita dapatkan probabilitas p_1 bahwa t yang didapat adalah benar

$$\begin{aligned} p_1 &= P(t \equiv l - X[l-1] \pmod n \mid t = S_{n+l-1}[l]) \\ &= P(S_{n+l-1}[l] + X[l-1] \equiv l \pmod n) \\ &= P(S_{n+l}[j_l] + S_{n+l}[k] \equiv i \pmod n) \\ &= \frac{2}{n}, \end{aligned}$$

dimana $i = l$, $S_{n+l}[j_l] = S_{n+l-1}[i]$ (karena permutasi), $X[l-1] = S_{n+l}[k]$, dan $k = S_{n+l}[i] + S_{n+l}[j_l]$. Untuk kasus $S_{n+l-1}[l] \neq t$ kita dapatkan probabilitas p_2 bahwa t yang didapat adalah benar

$$\begin{aligned} p_2 &= P(t \equiv l - X[l-1] \pmod n \mid t \neq S_{n+l-1}[l]) \\ &= P(t + X[l-1] \equiv l \pmod n \mid t \neq S_{n+l-1}[l]) \\ &= P(t + S_{n+l}[k] \equiv l \pmod n \mid t \neq S_{n+l-1}[l]) \\ &= P(S_{n+l-1}[l] + d + S_{n+l}[k] \equiv l \pmod n) \\ &= P(S_{n+l}[j_l] + S_{n+l}[k] \equiv i - d \pmod n) \\ &= P(S_{n+l}[j_l] + S_{n+l}[k] \equiv c \pmod n) \\ &= \frac{n-2}{n(n-1)}, \end{aligned}$$

dimana $i = l$, $S_{n+l}[j_l] = S_{n+l-1}[i]$ (karena permutasi), $X[l-1] = S_{n+l}[k]$, $d = t - S_{n+l-1}[l]$, $c = i - d \not\equiv i \pmod n$, dan $k = S_{n+l}[i] + S_{n+l}[j_l]$. Secara keseluruhan, probabilitas bahwa $t \equiv l - X[l-1] \pmod n$ adalah

$$\left(1 - \frac{1}{n}\right)^{n-2} \frac{2}{n} + \left(1 - \left(1 - \frac{1}{n}\right)^{n-2}\right) \frac{n-2}{n(n-1)} \approx \frac{1.36}{n}.$$

Hasil yang benar, meskipun prosentasinya kelihatan kecil, akan menonjol dibandingkan hasil-hasil yang salah yang masing-masing mempunyai probabilitas kurang dari $\frac{1}{n}$. Tentunya untuk mendapatkan hasil yang baik diperlukan *sample* yang cukup banyak. Untuk tingkat kesuksesan diatas 50 persen, diperlukan *sample* sebesar 43 ribu paket. Ini jauh lebih sedikit dibandingkan *sample* yang dibutuhkan FMS *attack* yaitu 9 juta paket. Untuk tingkat kesuksesan diatas 95 persen dibutuhkan *sample* sebesar 70 ribu paket. Jadi jelas PTW *attack* lebih efisien dibandingkan FMS *attack*.

Penggunaan RC4 dalam WEP mempunyai kelemahan yang menyebabkan WEP menjadi tidak aman. Kelemahan ini terutama berada pada bagian KSA, tetapi terdapat juga pada bagian PRGA. Kunci rahasia dapat terbongkar karena "jejak" dari kunci rahasia masih terdapat pada *keystream* dibagian awal.

PRGA harus berjalan lebih lama lagi sebelum "jejak" kunci rahasia menjadi sulit untuk terdeteksi. Itulah sebabnya dianjurkan agar bagian awal dari *keystream* (256 byte pertama) dibuang dan tidak digunakan untuk mengenkripsi, satu anjuran yang diikuti oleh WPA (Wi-Fi *protected access*). Tetapi meskipun bagian awal dibuang, karena terdapat kecenderungan pada PRGA, RC4 tetap memiliki kelemahan yang dapat dieksploitasi.

Banyak contoh lain penggunaan RC4 yang tidak aman. Jika tidak hati-hati, memang sangat mudah untuk menggunakan *stream cipher* secara tidak aman. Itulah sebabnya, tidak dianjurkan untuk menggunakan *stream cipher*, apalagi jika tidak paham dengan kelemahan algoritma yang digunakan. Daftar penggunaan RC4 secara tidak aman dimasa lalu sangat panjang. Semua produk Microsoft yang diketahui pernah menggunakan RC4, pernah menggunakannya secara tidak aman. Browser Netscape juga pernah mengimplementasikan RC4 secara tidak aman untuk keperluan SSL (*Secure Socket Layer*).

6.2 Ringkasan

Bab ini telah membahas *stream cipher* sebagai teknik enkripsi yang menyerupai enkripsi *one-time pad* dengan *keystream* yang dibuat menggunakan kunci enkripsi. Contoh yang dibahas adalah RC4 karena merupakan *stream cipher* yang banyak digunakan. Sangat mudah untuk menggunakan *stream cipher* secara tidak aman, contohnya penggunaan RC4 dalam WEP. Oleh sebab itu buku ini tidak merekomendasikan penggunaan *stream cipher*.

Bab 7

Kriptografi Block Cipher

Di bab-bab sebelumnya, kita melihat bagaimana enkripsi yang bersifat linear (termasuk *affine transformation*) rentan terhadap analisa statistik. Claude Shannon [sha49], yang dianggap bapak dari teori informasi, dalam *paper* yang diterbitkannya tahun 1949, menganjurkan dua metode untuk mempersulit analisa statistik: *diffusion* dan *confusion*.

Analisa statistik kerap berandalkan pengetahuan mengenai struktur statistik dari naskah asli dan "sisa" dari struktur ini dalam naskah acak. Untuk enkripsi sangat sederhana seperti *Caesar cipher*, "sisa" dari struktur[1] ini masih sangat besar, bahkan masih utuh dan terlokalisir, jadi sangat mudah untuk mencari struktur dalam naskah acak. Efek *diffusion* bertujuan memperlemah "sisa" struktur dengan menyebarnya secara merata ke bagian yang cukup besar dari naskah acak meliputi banyak karakter. Struktur statistik (dalam teori informasi kerap disebut *redundancy*) masih ada tetapi sudah tersebar. Satu-satunya cara menghilangkan struktur tanpa menghilangkan informasi adalah dengan kompresi.

Analisa statistik juga mencari hubungan antara kunci dengan naskah acak. Untuk *simple substitution cipher*, pengetahuan *a priori* mengenai statistik naskah asli memperkecil ruang kunci yang perlu diselidiki secara drastis, karena dapat digunakan untuk menghubungkan kunci dengan struktur statistik dari naskah acak. Efek *confusion* bertujuan untuk menghilangkan atau membuat tidak jelas hubungan antara naskah acak dengan kunci.

Block cipher seperti DES/3DES, CAST, IDEA dan AES menggunakan efek *diffusion* dan *confusion* untuk mempersulit analisa statistik. Ini dilakukan menggunakan apa yang disebut *Feistel network* atau *substitution permutation network*. Walaupun DES sudah dianggap lemah untuk ukuran sekarang, kita akan mulai dengan pembahasan DES karena DES merupakan standard enkripsi

[1] Dalam hal ini struktur statistik didapat dari statistik frekuensi penggunaan huruf.

pertama yang menggunakan konsep *Feistel network* dan algoritma DES masih digunakan oleh 3DES yang dianggap masih cukup kuat untuk penggunaan masa kini.

7.1 DES

DES (Data Encryption Standard) [nis99] pertama dijadikan standard FIPS (Federal Information Processing Standards) oleh NIST (National Institute of Standards and Technology) tahun 1977 untuk digunakan oleh semua instansi pemerintahan Amerika Serikat, dan semua kontraktor dan penyedia jasa untuk pemerintahan Amerika Serikat. DES dirancang oleh tim IBM yang dipimpin Horst Feistel dengan bantuan dari NSA (National Security Agency). DES adalah teknik enkripsi pertama (selain *one-time pad*) yang tahan terhadap *linear cryptanalysis* dan *differential cryptanalysis*.

DES menggunakan kunci sebesar 64 bit untuk mengenkripsi blok juga sebesar 64 bit. Akan tetapi karena 8 bit dari kunci digunakan sebagai *parity*, kunci efektif hanya 56 bit. Gambar 7.1 secara garis besar menunjukkan proses enkripsi DES. Dalam DES, penomoran bit adalah dari kiri kekanan dengan bit 1 menjadi *most significant bit*, jadi untuk 64 bit, bit 1 mempunyai nilai 2^{63}.

Permutasi menggunakan *initial permutation* dilakukan terhadap input sebesar 64 bit. Hasil permutasi dibagi menjadi dua blok L0 dan R0, masing-masing sebesar 32 bit, dimana L0 merupakan 32 bit pertama dari hasil permutasi dan R0 merupakan 32 bit sisanya (bit 33 hasil permutasi menjadi bit 1 R0). Sebanyak 16 putaran enkripsi dilakukan menggunakan fungsi *cipher* f dan setiap putaran menggunakan kunci 48 bit yang berbeda dan dibuat berdasarkan kunci DES. Efeknya adalah setiap blok secara bergantian dienkripsi, masing-masing sebanyak 8 kali.

Pada setiap putaran, blok sebesar 32 bit dienkripsi menggunakan rumus:

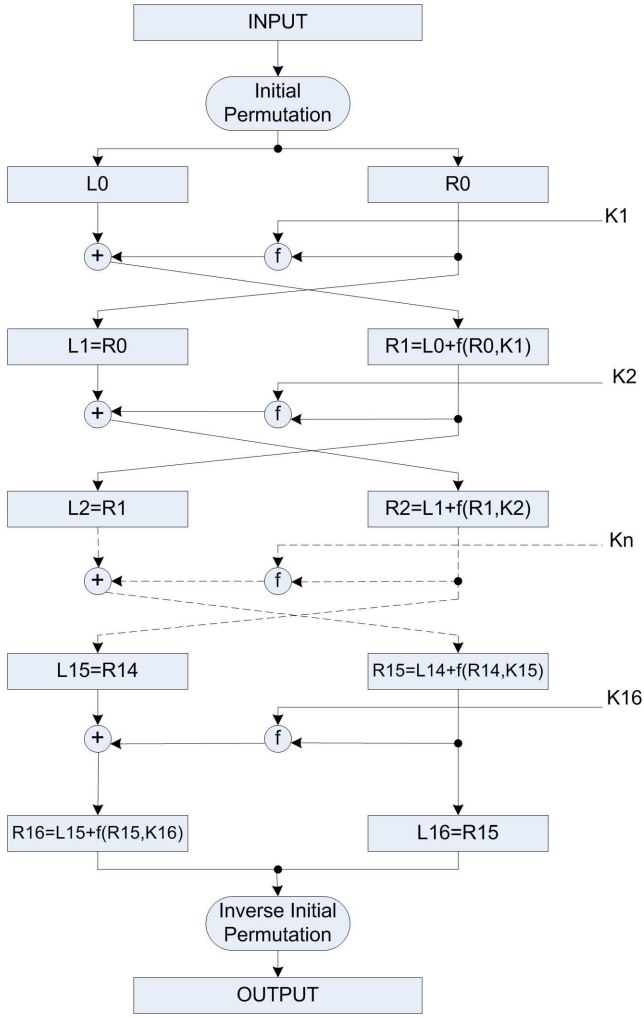
$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n) \tag{7.1}$$

dan blok juga sebesar 32 bit tidak dienkripsi:

$$L_n = R_{n-1} \tag{7.2}$$

dimana
L_{n-1} adalah blok yang sedang giliran tidak dienkripsi,
$\oplus$ adalah operasi *exclusive or* secara bitwise,
f adalah fungsi *cipher* yang akan dijelaskan,
R_{n-1} adalah blok yang sedang giliran dienkripsi, dan
K_n adalah kunci untuk putaran n.

Setelah putaran terahir, kedua blok digabung lagi tetapi bertukaran tempat, jadi R16 menjadi blok pertama dan L16 menjadi blok kedua. Ini dilakukan



Gambar 7.1: Enkripsi DES

untuk menyederhanakan proses dekripsi. Setelah itu permutasi menggunakan *inverse permutation* dilakukan terhadap blok yang sudah digabung menjadi 64 bit memberikan hasil ahir enkripsi DES.

Untuk proses dekripsi, rumus 7.1 dan 7.2 memberikan:

$$\begin{aligned} L_{n-1} &= R_n \oplus f(R_{n-1}, K_n) \\ &= R_n \oplus f(L_n, K_n). \end{aligned}$$

Jadi tanpa harus mengetahui fungsi f, kita tahu bahwa operasi *bitwise exclusive or* R_n dengan $f(R_{n-1}, K_n) = f(L_n, K_n)$ akan mendapatkan kembali L_{n-1} (lihat 2.2 - penjelasan enkripsi *one-time pad*).

Juga karena

$$\begin{aligned} \text{output} &= \text{IP}^{-1}(\text{R16L16}) \text{ dan} \\ \text{L0R0} &= \text{IP}(\text{input}) \end{aligned}$$

dimana IP adalah *initial permutation*, maka

$$\begin{aligned} \text{R16L16} &= \text{IP}(\text{output}) \text{ dan} \\ \text{input} &= \text{IP}^{-1}(\text{L0R0}). \end{aligned}$$

Jadi proses dekripsi DES dapat menggunakan algoritma yang sama dengan enkripsi, asalkan *schedule* kunci dibalik (mulai dari K16 dan berahir dengan K1). Blok yang terahir dienkripsi harus didekripsi pertama, itulah sebabnya proses enkripsi membuat L16 dan R16 bertukaran tempat.

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabel 7.1: Tabel *Initial Permutation*

Tabel 7.1 adalah tabel untuk *Initial Permutation*. Bit 1 hasil permutasi berasal dari bit 58 input, bit 2 dari bit 50, dan seterusnya hingga bit 64 dari bit 7.

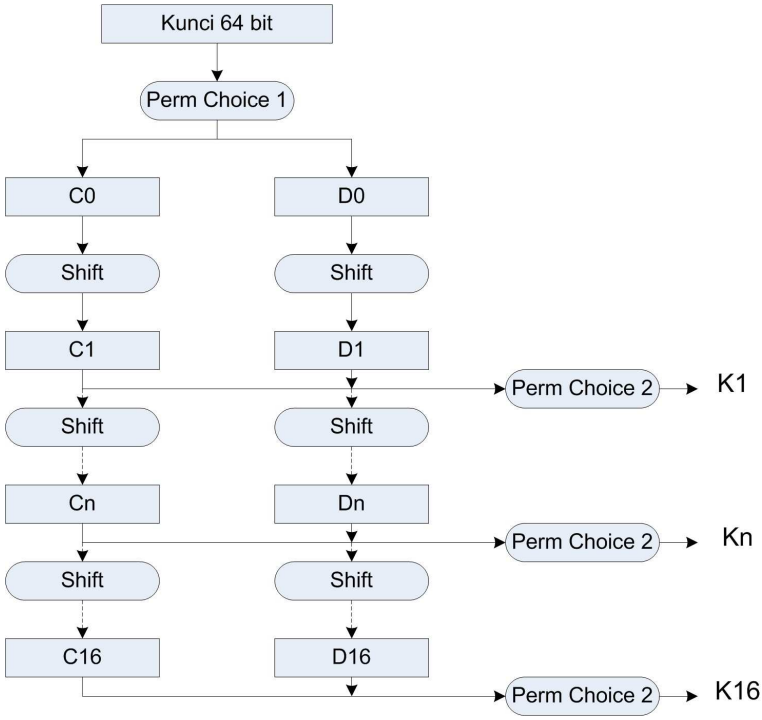
Tabel 7.2 adalah tabel untuk *Inverse Initial Permutation*. Transformasi *Inverse Initial Permutation* adalah *inverse* dari transformasi *Initial Permutation*:

- dalam IP bit 1 berasal dari bit 58, dalam IP^{-1} bit 58 berasal dari bit 1,
- dalam IP bit 2 berasal dari bit 50, dalam IP^{-1} bit 50 berasal dari bit 2,
- dalam IP bit 3 berasal dari bit 42, dalam IP^{-1} bit 42 berasal dari bit 3, dan seterusnya.

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Tabel 7.2: Tabel *Inverse Initial Permutation*



Gambar 7.2: Algoritma *Key Schedule* DES

Jadi jika bit 58 dipindahkan oleh IP menjadi bit 1, IP^{-1} akan mengembalikannya ke posisi bit 58, demikian juga bit lainnya.

Gambar 7.2 menunjukkan secara garis besar algoritma *key schedule* pembuatan 16 kunci putaran. Transformasi *permuted choice* 1 membuang 8 bit untuk *parity* dan melakukan permutasi terhadap 56 bit yang tersisa, yang kemudian dibagi menjadi dua blok C0 dan D0, masing-masing 28 bit.

PC1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Tabel 7.3: Tabel *Permuted Choice* 1

Untuk setiap putaran n:

- operasi *shift* dilakukan terhadap blok C_{n-1} dan blok D_{n-1} menghasilkan masing-masing blok C_n dan blok D_n, dan

- operasi *permuted choice* 2 dilakukan terhadap gabungan blok C_n dan blok D_n, membuang 8 dari 56 bit, lalu melakukan permutasi terhadap 48 bit yang tersisa, menghasilkan kunci putaran K_n.

Tabel 7.3 menunjukkan operasi *permuted choice* 1. Tabel mempunyai dua bagian, bagian pertama untuk membuat blok C0 dan bagian kedua untuk membuat blok D0. Jadi bit 1 blok C0 didapat dari bit 57 kunci DES, bit 2 blok C0 didapat dari bit 49 kunci DES, dan seterusnya. Bit 1 blok D0 didapat dari bit 63 kunci DES, bit 2 blok D0 didapat dari bit 55 kunci DES, dan seterusnya. Bit 8, 16, 24, 32, 40, 48, 56 dan 64 kunci DES tidak digunakan.

Operasi *shift* merotasi blok 28 bit satu atau dua posisi kekiri tergantung pada putaran. Merotasi satu posisi kekiri berarti bit 1 menjadi bit 28 (karena bit 1 adalah bit paling kiri), bit 2 menjadi bit 1, bit 3 menjadi bit 2, dan seterusnya. Merotasi dua posisi berarti bit 1 menjadi bit 27, bit 2 menjadi bit 28, bit 3 menjadi bit 1, dan seterusnya.

Tabel 7.4 menunjukkan berapa besar *shift* yang harus dilakukan untuk setiap putaran. Jadi untuk putaran 1, 2, 9 dan 16 blok C dan blok D masing-masing dirotasi 1 posisi, sedangkan untuk putaran 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14 dan 15 besar rotasi adalah 2 posisi.

Putaran	Besar Shift
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Tabel 7.4: Tabel *Shift*

Tabel 7.5 menunjukkan operasi *permuted choice* 2. Bit 9, 18, 22, 25, 35, 38, 43 dan 54 dibuang, bit 1 kunci putaran didapat dari bit 14 gabungan blok C dan D, bit 2 kunci putaran didapat dari bit 17 gabungan blok C dan D, dan seterusnya.

Komponen terahir dari algoritma DES yang perlu dijelaskan adalah fungsi *cipher f*. Gambar 7.3 menunjukkan fungsi *cipher f* secara garis besar.

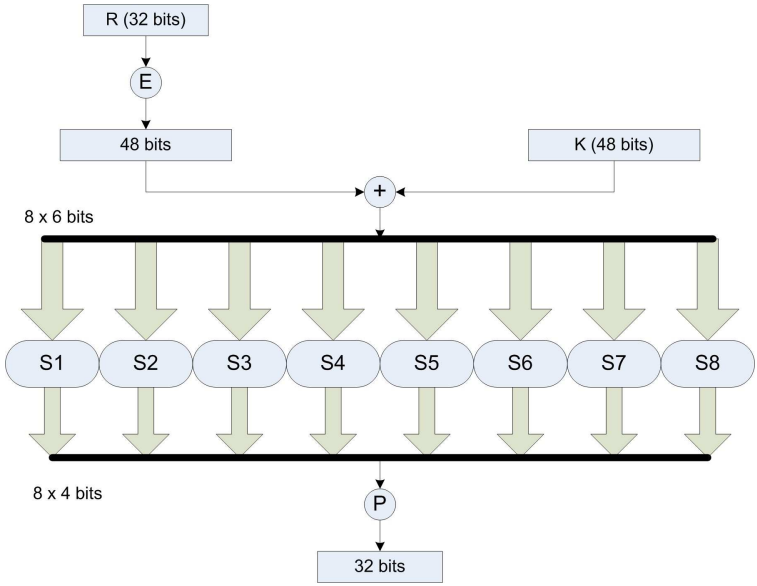
Blok sebesar 32 bit diekspansi menjadi 48 bit menggunakan transformasi E (lihat tabel B.1 di appendix B untuk tabel transformasi E). Operasi *exclusive or* dilakukan terhadap hasil ekspansi dan kunci putaran (yang juga 48 bit). Hasil *exclusive or* lalu dibagi menjadi 8 bagian, masing-masing sebesar 6 bit. Setiap bagian disubstitusi menghasilkan 4 bit, menggunakan fungsi substitusi S1 sampai dengan S8 (lihat tabel B.5 di appendix B). Karena operasi ini merupakan bagian dari fungsi *cipher f*, ini bukan operasi langsung terhadap naskah (lihat gambar 7.1, naskah direpresentasikan oleh L), jadi tidak ada informasi yang hilang dari naskah asli dengan dilakukannya substitusi 6 bit menjadi 4 bit. Gabungan hasil substitusi sebesar 32 bit kemudian dipermutasi menggunakan P yang menghasilkan 32 bit hasil ahir (lihat tabel B.2 di appendix B untuk tabel permutasi P).

Enkripsi DES menggunakan permutasi dan substitusi secara berulang untuk mendapatkan efek *diffusion* dan *confusion*. Operasi permutasi lebih mengarah

PC2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Tabel 7.5: Tabel *Permuted Choice* 2



Gambar 7.3: Fungsi *Cipher f*

pada efek *diffusion*, karena permutasi berefek menyebar struktur informasi naskah keseluruh blok sebesar 64 bit. Operasi substitusi dengan *S-box* membuat hubungan antara kunci dengan naskah enkripsi tidak jelas, jadi menghasilkan efek *confusion*.

Meskipun semula ada kecurigaan terhadap ketangguhan enkripsi DES (ada yang berspekulasi bahwa NSA, yang ikut berperan dalam merancang DES, tidak akan memperbolehkan enkripsi tangguh digunakan oleh pihak lain), ter-

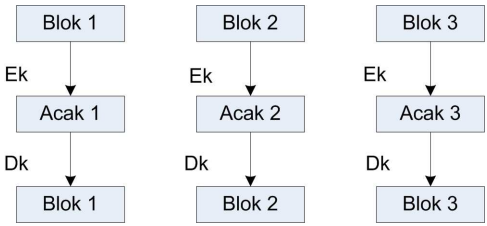
nyata DES cukup tahan terhadap analisa statistik. Masih ada kecurigaan terhadap fungsi *S-box* karena tidak adanya penjelasan mengenai bagaimana *S-box* dirancang. Akan tetapi, kelemahan DES adalah besar kunci efektif hanya 56 bit, yang membuatnya rentan terhadap *brute force attack*. Pada bulan Mei 2005, DES sebagai standard FIPS telah dicabut.

7.2 Mode Operasi DES

Standard FIPS-81, yang pernah dikeluarkan oleh NIST, memberi petunjuk untuk mode operasi DES. Standard tersebut telah dicabut seiring dengan dicabutnya DES sebagai standard FIPS, namun mode operasi dalam standard dapat digunakan untuk *block cipher* lainnya. Mode operasi dalam FIPS-81 adalah:

- Electronic Code Book (ECB).
- Cipher Block Chaining (CBC).
- Cipher Feedback (CFB).
- Output Feedback (OFB).

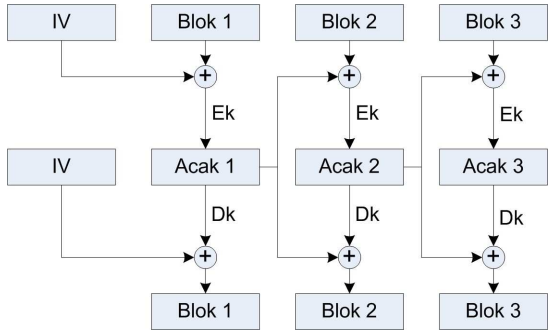
Dengan mode operasi ECB, data dienkripsi secara berurutan per blok menggunakan satu kunci DES. Jika naskah asli berisi blok yang berulang, maka naskah acak juga berisi blok yang berulang di posisi yang sama. Gambar 7.4 menunjukkan penggunaan DES dengan mode operasi ECB.



Gambar 7.4: DES dengan mode ECB

Dengan mode operasi CBC, setiap blok data dikombinasikan dahulu dengan hasil enkripsi blok sebelumnya menggunakan operasi *exclusive or*, lalu dienkripsi menggunakan kunci DES. Untuk blok pertama, blok dikombinasikan dengan *initialization vector* (IV) sebesar 64 bit, yang sebaiknya dibuat secara acak. Gambar 7.5 menunjukkan penggunaan DES dengan mode CBC.

Dengan mode operasi CBC, naskah yang berisi blok yang berulang akan dienkripsi menjadi sesuatu yang tidak berulang[2]. Akibatnya mode CBC mempersulit analisa untuk memecahkan enkripsi. Mode CBC adalah mode terpopuler untuk penggunaan DES dan *block cipher* lainnya karena menghasilkan enkripsi yang teraman diantara semua mode.



Gambar 7.5: DES dengan mode CBC

Mode CFB menghasilkan apa yang dinamakan *stream cipher* (lihat bab 6) dimana operasi *exclusive or* dilakukan terhadap naskah asli dengan *keystream* menghasilkan naskah acak, jadi meniru enkripsi *one-time pad*. Namun naskah acak dijadikan *feedback* untuk ikut menentukan *keystream*, jadi agak lebih aman dibandingkan OFB. Gambar 7.6 menunjukkan penggunaan DES dengan mode CFB.

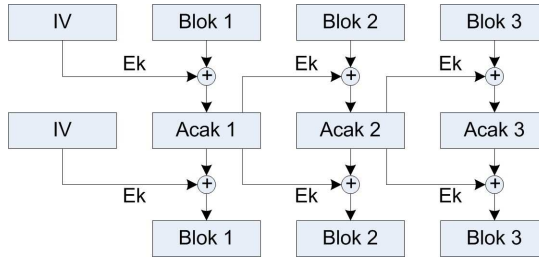
Dengan mode operasi CFB, *keystream* dibuat dengan mengenkripsi IV dan naskah acak menggunakan algoritma enkripsi DES. Jadi untuk blok pertama, bagian *keystream* yang digunakan adalah hasil enkripsi IV, untuk blok kedua, bagian *keystream* yang digunakan adalah hasil enkripsi blok acak pertama, dan seterusnya (naskah acak dijadikan *feedback*). Jadi enkripsi DES dapat dianggap sebagai *pseudo-random number generator* dengan IV atau blok acak sebelumnya sebagai *seed*.

Selain menggunakan blok sebesar 64 bit, mode CFB dapat juga digunakan dengan besar blok kurang dari 64 bit (sebut saja k bit dengan $1 \leq k < 64$) sebagai berikut:

- IV sebesar k bit digunakan sebagai *least significant bits* untuk *seed* blok pertama, dengan *most significant bits* diisi 0.
- *Seed* lalu dienkripsi menggunakan DES, dengan k *most significant bits* hasil enkripsi digunakan sebagai bagian *keystream* untuk blok pertama.

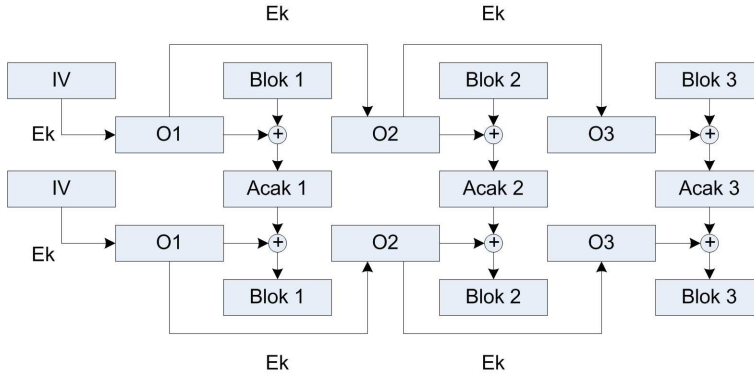
[2]Ini mungkin tidak pasti, namun jika ada kemungkinan blok berulang juga dalam naskah acak, kemungkinan ini sangat kecil.

- Blok acak didapat dari operasi *exclusive or* antara blok asli dengan bagian *keystream* untuk blok.
- Untuk blok-blok berikutnya, *seed* didapat dengan membuang k *most significant bits* dari *seed* blok sebelumnya, menggeser sisa *seed* kekiri k posisi menjadi *most significant bits*, dan mengisi k *least significant bits* dengan blok acak sebelumnya.



Gambar 7.6: DES dengan mode CFB

Mode OFB juga menghasilkan *stream cipher*, tetapi pembuatan *keystream* berbeda dengan mode CFB. *Keystream* hanya ditentukan oleh IV dan kunci enkripsi, jadi OFB lebih lemah dibandingkan CFB. Gambar 7.7 menunjukkan penggunaan DES dengan mode OFB.



Gambar 7.7: DES dengan mode OFB

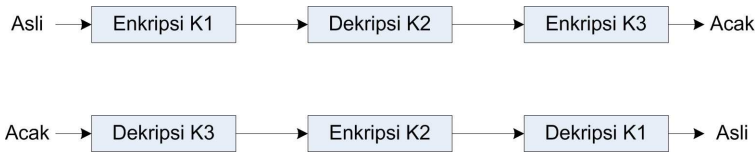
Beda antara OFB dengan CFB terletak pada *feedback*. Dengan OFB, yang dijadikan *feedback* adalah *keystream*, bukan naskah acak. Sama dengan CFB,

selain menggunakan blok sebesar 64 bit, OFB dapat juga digunakan dengan besar blok kurang dari 64 bit (sebut saja k bit dengan $1 \leq k < 64$) sebagai berikut:

- IV sebesar k bit digunakan sebagai *least significant bits* untuk *seed* blok pertama, dengan *most significant bits* diisi 0.
- *Seed* lalu dienkripsi menggunakan DES, dengan k *most significant bits* hasil enkripsi digunakan sebagai bagian *keystream* untuk blok pertama.
- Blok acak didapat dari operasi *exclusive or* antara blok asli dengan bagian *keystream* untuk blok.
- Untuk blok-blok berikutnya, *seed* didapat dengan membuang k *most significant bits* dari *seed* blok sebelumnya, menggeser sisa *seed* kekiri k posisi menjadi *most significant bits*, dan mengisi k *least significant bits* dengan bagian *keystream* blok sebelumnya.

7.3 3DES

Enkripsi dengan DES sudah dianggap tidak memadai karena kunci efektif sebesar 56 bit sudah terlalu kecil. Standard Triple DES menggunakan algoritma DES dengan 3 kunci seperti dalam gambar 7.8.



Gambar 7.8: Enkripsi dan Dekripsi 3DES

Dengan 3 kunci DES K1, K2 dan K3, enkripsi 3DES dilakukan sebagai berikut:

1. Enkripsi DES dengan kunci K1 dilakukan terhadap naskah asli.
2. Dekripsi DES dengan kunci K2 dilakukan terhadap hasil pertama.
3. Enkripsi DES dengan kunci K3 dilakukan terhadap hasil kedua.

Jadi enkripsi 3DES mempunyai rumus:

$$C = E3_{k1,k2,k3}(P) = E_{k3}(D_{k2}(E_{k1}(P))) \tag{7.3}$$

dimana
$E3_{k1,k2,k3}(P)$ adalah enkripsi 3DES terhadap P dengan kunci $k1$, $k2$ dan $k3$,
$E_{kn}(P)$ adalah enkripsi DES terhadap P dengan kunci kn, dan
$D_{kn}(P)$ adalah dekripsi DES terhadap P dengan kunci kn.

Dekripsi 3DES mempunyai rumus:

$$P = D3_{k1,k2,k3}(C) = D_{k1}(E_{k2}(D_{k3}(C))) \tag{7.4}$$

dimana
$D3_{k1,k2,k3}(C)$ adalah dekripsi 3DES terhadap C dengan kunci $k1$, $k2$ dan $k3$.

Triple DES dapat digunakan dengan satu kunci k sebagai berikut:

$$\begin{aligned} E3_{k,k,k}(P) &= E_k(D_k(E_k(P))) \\ &= E_k(P) \text{ dan} \\ D3_{k,k,k}(C) &= D_k(E_k(D_k(C))) \\ &= D_k(C). \end{aligned}$$

Jadi 3DES dengan 1 kunci ekuivalen dengan DES.

Dengan dua kunci $k1$ dan $k2$, penggunaan 3DES adalah sebagai berikut:

$$\begin{aligned} E3_{k1,k2,k1}(P) &= E_{k1}(D_{k2}(E_{k1}(P))) \text{ dan} \\ D3_{k1,k2,k1}(C) &= D_{k1}(E_{k2}(D_{k1}(C))). \end{aligned}$$

Enkripsi 3DES dengan dua atau tiga kunci masih cukup tangguh untuk penggunaan saat ini. Walaupun enkripsi 3DES agak lamban komputasinya dibandingkan dengan enkripsi *block cipher* lainnya yang masih cukup tangguh, 3DES tergolong populer.

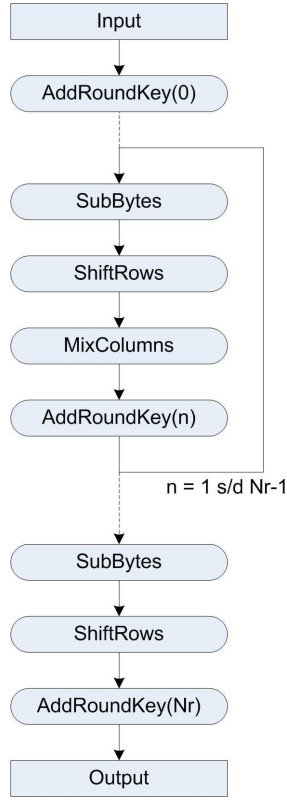
Bagi pembaca yang ingin mendapatkan rekomendasi penggunaan 3DES secara rinci, dipersilahkan untuk membaca [nis08].

7.4 AES

AES (Advanced Encryption Standard) [nis01] adalah teknik enkripsi yang dijadikan standard FIPS oleh NIST tahun 2001. AES dimaksudkan akan, secara bertahap, menggantikan DES sebagai standard enkripsi di Amerika Serikat untuk abad ke 21. (DES sebagai standard FIPS telah dicabut, Mei 2005.)

AES menjadi standard melalui proses seleksi. Dari beberapa teknik enkripsi yang dicalonkan untuk menjadi AES, yang terpilih adalah enkripsi Rijndael. Teknik enkripsi ini termasuk jenis *block cipher* seperti halnya dengan DES. Perbedaan utama antara teknik enkripsi AES dan teknik enkripsi DES adalah AES juga menggunakan substitusi (menggunakan S-boxes) secara langsung terhadap naskah, sedangkan substitusi S-box digunakan DES hanya dalam fungsi

cipher f yang hasilnya kemudian dioperasikan terhadap naskah menggunakan *exclusive or*, jadi DES tidak menggunakan substitusi secara langsung terhadap naskah. AES juga menggunakan kunci enkripsi yang lebih besar yaitu 128 bit, 192 bit, atau 256 bit.



Gambar 7.9: Enkripsi AES

Gambar 7.9 menunjukkan, secara garis besar, enkripsi AES. Input berupa naskah asli sebesar 128 bit, sedangkan output adalah naskah acak sebesar 128 bit. Setiap transformasi dilakukan secara langsung terhadap naskah, mulai dengan transformasi $AddRoundKey(0)$. Jadi setiap transformasi harus mempunyai *inverse* agar naskah acak dapat didekripsi. Pada putaran 1 sampai dengan $Nr - 1$, transformasi *SubBytes*, *ShiftRows*, *MixColumns* dan *AddRoundKey* dilakukan terhadap naskah. Pada putaran terahir (Nr), transformasi *SubBytes*, *ShiftRows* dan *AddRoundKey* dilakukan terhadap naskah (transformasi *MixColumns* tidak dilakukan). Total ada Nr putaran.

Jumlah putaran (Nr) tergantung pada besar kunci yang digunakan. Tabel 7.6 menunjukkan jumlah putaran (Nr) untuk kunci sebesar 128 bit, 192 bit dan 256 bit. Jadi untuk kunci sebesar 128 bit, besar kunci (Nk) adalah 4 *word* (setiap *word* mempunyai 32 bit), besar blok (Nb) adalah 4 *word*, dan jumlah putaran (Nr) adalah 10.

	Besar Kunci (Nk) dalam *words*	Besar Blok (Nb) dalam *words*	Jumlah Putaran (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Tabel 7.6: Tabel untuk Jumlah Putaran

Penjelasan AES menggunakan konvensi urutan indeks untuk bit dan byte sebagai berikut:

- Urutan indeks bit sebagai input adalah $input_0$, $input_1$, $input_2$, dan seterusnya.
- Urutan indeks byte sebagai input adalah a_0, a_1, a_2, dan seterusnya.
- Dalam byte, bit menggunakan urutan indeks berlawanan dengan input: b_7, b_6, b_5, dan seterusnya sampai dengan b_0. *Most significant bit* dalam byte adalah b_7.

input	0	1	2	3	4	5	6	7	8	9	...	...
a	0								1			...
b	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

Tabel 7.7: Index bit dan byte AES

Tabel 7.7 memperlihatkan konvensi indeks bit dan byte untuk AES. Tabel menunjukkan bahwa untuk byte kedua (a_1), $b_7 = input_8$, $b_6 = input_9$, $b_5 = input_{10}$, dan seterusnya sampai dengan $b_0 = input_{15}$.

Algoritma enkripsi AES beroperasi terhadap *state* dari naskah yang dipandang sebagai matrik terdiri dari 16 byte. Setiap kolom dari *state* merepresentasikan satu *word*, dengan $s_{0,i}$ sebagai *most significant byte* untuk kolom i. Tabel 7.8 menunjukkan bagaimana *state* didapat dari input dan bagaimana *state* dijadikan output. Transformasi *AddRoundKey*, *SubBytes*, *ShiftRows* dan *MixColumns* semua dilakukan terhadap *state*.

input bytes

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_10	in_{14}
in_3	in_7	in_11	in_{15}

⇓

state

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

⇓

output bytes

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_10	out_{14}
out_3	out_7	out_11	out_{15}

Tabel 7.8: Input, State dan Output

Beberapa transformasi yang dilakukan dalam enkripsi AES memperlakukan byte seolah *polynomial* dalam *polynomial field* $\mathbf{GF}(2^8)$ (lihat bagian 5.7). Setiap bit dalam byte merepresentasikan koefisien suku *polynomial* sebagai berikut:

- bit b_7 merupakan koefisien untuk x^7,
- bit b_6 merupakan koefisien untuk x^6,
- dan seterusnya sampai dengan bit b_0 yang merupakan koefisien untuk x^0 (konstan).

Operasi *inverse polynomial* terhadap byte dapat dilakukan (dalam $\mathbf{GF}(2^8)$) jika byte $\neq 0$. *Irreducible polynomial* yang digunakan AES untuk $\mathbf{GF}(2^8)$ adalah

$$x^8 + x^4 + x^3 + x + 1$$

AES juga melakukan transformasi *affine* (dalam $\mathbf{GF}(2)$) terhadap byte menggunakan rumus

$$b'_i = b_i + b_{i+4 \bmod 8} + b_{i+5 \bmod 8} + b_{i+6 \bmod 8} + b_{i+7 \bmod 8} + c_i \qquad (7.5)$$

dengan $0 \leq i \leq 7$ dan byte $c = 01100011$. Jika dijabarkan, rumus menjadi

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Operasi *SubByte* (substitusi S-box) terhadap byte terdiri dari operasi *inverse polynomial* (jika byte $\neq 0$) diikuti oleh transformasi *affine* menggunakan rumus 7.5. Jika byte $= 0$ maka hanya transformasi *affine* yang dilakukan, menghasilkan 01100011 (63 dalam notasi hexadecimal). Operasi *SubByte* (substitusi S-box) dan *inverse* operasi *SubByte* dalam bentuk tabel kami berikan di appendix C.

Operasi *AddRoundKey* melakukan *bitwise exclusive or* menggunakan kunci putaran sebesar 128 bit terhadap *state* (yang juga 128 bit). Untuk setiap putaran, kunci putaran berbeda dari *key schedule* yang didapat dari ekspansi kunci enkripsi digunakan. Total ada $Nr+1$ kunci putaran yang digunakan, satu untuk operasi awal sebelum putaran pertama, dan satu untuk setiap putaran. Jadi ekspansi kunci menghasilkan $Nb(Nr+1) = 4(Nr+1)$ *words* menggunakan algoritma sebagai berikut:

1. $i \leftarrow 0$,
2. $w[i] \leftarrow [k[4i], k[4i+1], k[4i+2], k[4i+3]]$,
3. $i \leftarrow i+1$, jika $i < Nk$ kembali ke langkah 2,
4. $t \leftarrow w[i-1]$,
5. jika $i \bmod Nk = 0$ maka $t \leftarrow SubWord(RotWord(t)) \oplus Rcon[i/Nk]$, kalau tidak, jika $Nk > 6$ dan $i \bmod Nk = 4$ maka $t \leftarrow SubWord(t)$,
6. $w[i] \leftarrow w[i-Nk] \oplus t$,
7. $i \leftarrow i+1$, jika $i < Nb(Nr+1)$ kembali ke langkah 4

dimana

- k adalah kunci enkripsi,
- w adalah *array* terdiri dari $Nb(Nr+1)$ *words* dengan indeks mulai dari 0,

- $\oplus$ adalah operasi *bitwise exclusive or*,
- *SubWord* adalah operasi terhadap *word* dimana substitusi *SubByte* dilakukan terhadap setiap byte dalam *word*,
- *RotWord* adalah operasi terhadap *word* dimana urutan byte dalam *word* diubah sebagai berikut:

$$[a_0, a_1, a_2, a_3] \Rightarrow [a_1, a_2, a_3, a_0],$$

dan

- *Rcon*[*j*] adalah word sebagai berikut:

$$[x^{j-1}, 0, 0, 0]$$

dimana $x^0 = 00000001$, $x = 00000010$, $x^2 = 00000100$ dan seterusnya sampai dengan $x^7 = 10000000$, dan untuk x dengan pangkat > 7 aritmatika *polynomial field* digunakan.

Jadi Nk *words* pertama untuk w didapatkan langsung dari kunci enkripsi, sedangkan *words* selanjutnya untuk w ditentukan oleh *word* 1 posisi sebelumnya dan *word* Nk posisi sebelumnya. Kunci putaran diambil dari w sebagai berikut:

$$k_i \leftarrow [w[4i], w[4i+1], w[4i+2], w[4i+3]]$$

dengan $0 \leq i \leq Nr$. Transformasi *AddRoundKey*(i) terhadap *state* adalah sebagai berikut:

$$state' \leftarrow state \oplus k_i.$$

Transformasi *AddRoundKey* mempunyai *inverse* yaitu transformasi *AddRoundKey* sendiri (ingat sifat operasi *exclusive or*).

Untuk transformasi *SubBytes*, substitusi *SubByte* dilakukan terhadap setiap byte dalam *state*. Untuk *inverse* transformasi *SubBytes*, substitusi dilakukan menggunakan *inverse SubByte* (lihat appendix C).

Transformasi *ShiftRows* menggeser byte dalam baris *state* sebagai berikut:

- baris 0 tidak digeser,
- baris 1 digeser 1 posisi kekiri dengan byte terkiri menjadi byte terkanan,
- baris 2 digeser 2 posisi kekiri, jadi dua byte sebelah kiri ditukar dengan dua byte sebelah kanan, dan
- baris 3 digeser 3 posisi kekiri, yang efeknya sama dengan menggeser 1 posisi kekanan.

state

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$\Downarrow$

state'

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

Tabel 7.9: ShiftRows

Tabel 7.9 memperlihatkan efek *ShiftRows* terhadap *state*. Tidak terlalu sulit untuk menunjukkan bahwa transformasi *ShiftRows* mempunyai *inverse*.

Transformasi terahir yang perlu dijelaskan untuk enkripsi AES adalah transformasi *MixColumns*. Untuk *MixColumns*, kolom dalam *state* (yang merupakan *word*) diperlakukan sebagai *polynomial* dengan koefisien dalam $\mathbf{GF}(2^8)$. *Word*

$$[a_0, a_1, a_2, a_3]$$

diperlakukan sebagai *polynomial*

$$a_3x^3 + a_2x^2 + a_1x + a_0.$$

Pertambahan *polynomial* dilakukan sebagaimana pertambahan dalam *polynomial ring*. Agar hasil tetap sebesar *word*, perkalian dilakukan modulo *polynomial*

$$g(x) = x^4 + 1.$$

Tidak terlalu sulit untuk menunjukkan bahwa

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4} \tag{7.6}$$

karena $-1 = 1$ dalam $\mathbf{GF}(2^8)$. Karena $x^4 + 1$ bukan merupakan *irreducible polynomial* dalam $K[x]$ dimana $K = \mathbf{GF}(2^8)$, maka $K[x]/g(x)K[x]$ bukan merupakan *field*: tidak semua *polynomial* mempunyai *inverse*. Akan tetapi *polynomial*

$$a(x) = 03x^3 + 01x^2 + 01x + 02$$

mempunyai *inverse*

$$a^{-1}(x) = 0\mathrm{b}x^3 + 0\mathrm{d}x^2 + 09x + 0\mathrm{e}.$$

Transformasi *MixColumns* mengalikan (modulo *polynomial* $g(x)$) setiap kolom dalam *state* (diperlakukan sebagai *polynomial*) dengan *polynomial* $a(x)$. Transformasi *MixColumns* terhadap *state* dapat dirumuskan efeknya terhadap setiap kolom c sebagai berikut:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}.$$

Tidak terlalu sulit untuk meyakinkan kebenaran rumus ini menggunakan rumus 7.6. Transformasi *MixColumns* mempunyai *inverse* yaitu mengalikan setiap kolom dalam *state* dengan $a^{-1}(x)$ modulo *polynomial* $g(x)$.

Karena semua transformasi yang dilakukan dalam enkripsi AES mempunyai *inverse*, dekripsi dapat dilakukan menggunakan *inverse* transformasi dengan urutan kebalikan dari urutan transformasi enkripsi. Proses dekripsi dimulai dengan transformasi *AddRoundKey*(Nr) terhadap naskah acak, diikuti dengan transformasi *inverse ShiftRows*, *inverse SubBytes* dan seterusnya.

AES telah menggantikan DES sebagai standard enkripsi untuk keperluan instansi pemerintahan Amerika Serikat. Ada yang meragukan ketangguhan AES karena semua transformasi yang dilakukan dalam enkripsi AES mempunyai rumus aljabar yang elegan. Akan tetapi rumus yang elegan tidak berarti parameter kunci dapat dipecahkan dengan mudah dari persamaan. Kriptografi *public key* menggunakan rumus aljabar yang elegan, namun parameter kunci privat sulit untuk dipecahkan. Kembali ke AES, karena enkripsi adalah manipulasi berbagai bit, pemecahan parameter kunci dapat dirumuskan sebagai masalah SAT dalam aljabar Boolean, namun masalah SAT adalah masalah yang bersifat NP-*complete* yang pada umumnya tidak dapat dipecahkan secara efisien (untuk pemecahan kunci enkripsi AES, ruang pencarian terlalu besar sehingga dalam prakteknya pemecahan tidak dapat dilakukan).

7.5 Ringkasan

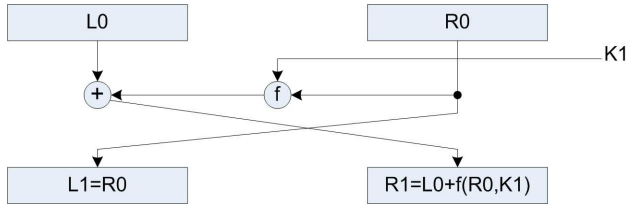
Dalam bab ini kita telah bahas berbagai enkripsi *block cipher* antara lain DES dan AES. Meskipun dianggap sudah tidak memadai untuk ukuran sekarang, DES dibahas karena merupakan *block cipher* pertama yang menjadi standard internasional dan merupakan contoh dari enkripsi yang tahan terhadap *differential cryptanalysis* dan *linear cryptanalysis*. Mode penggunaan DES juga dibahas, dengan mode CBC (*cipher block chaining*) sebagai mode yang direkomendasikan, sedangkan CFB dan OFB adalah mode-mode penggunaan DES sebagai *building block* dari suatu *stream cipher*. DES juga digunakan sebagai *building block* untuk 3DES (*triple*-DES). AES dibahas karena merupakan standard baru yang telah menggantikan DES. Berbeda dengan *block cipher*

lainnya, AES menggunakan transformasi *affine* (tetapi dalam suatu *polynomial field*) untuk fungsi S-box.

Bab 8

Analisa Block Cipher

Enkripsi *block cipher* menggunakan fungsi *cipher* yang agak lemah berulang kali untuk mendapatkan fungsi *cipher* yang kuat. Setiap putaran menambahkan efek *confusion* dan *diffusion* terhadap proses enkripsi. Semakin banyak putaran yang digunakan, semakin besar efek *confusion* dan *diffusion* dalam proses enkripsi, selama efek *confusion* dan *diffusion* belum "jenuh." Jadi, agar optimal, jumlah putaran adalah jumlah terkecil yang mengakibatkan efek *confusion* dan *diffusion* menjadi "jenuh."



Gambar 8.1: 1 Putaran DES

Mari kita ambil contoh enkripsi DES. Gambar 8.1 menunjukkan 1 putaran dalam DES. Kita coba dapatkan kunci putaran K_1 jika L_0, R_0 dan L_1, R_1 diketahui. Berdasarkan rumus

$$R_1 = L_0 \oplus f(R_0, K_1)$$

kita dapat mengkalkulasi output fungsi *cipher* f:

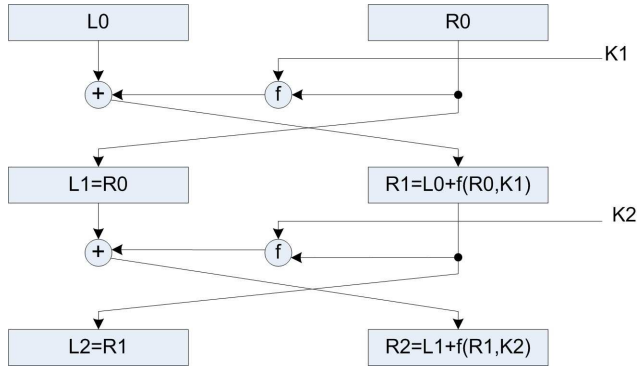
$$f(R_0, K_1) = R_1 \oplus L_0.$$

Gambar 7.3 menunjukkan fungsi *cipher* f untuk DES. Jika output fungsi *cipher* f diketahui, maka output dari S-boxes S1 sampai dengan S8 dapat dikalkulasi

menggunakan *inverse* permutasi P. Untuk setiap S-box, jika output diketahui, maka ada 4 kandidat untuk input S-box. Jadi untuk 8 S-boxes, ada 4^8 atau 2^{16} kandidat input S-boxes yang dapat menghasilkan output yang diketahui. Menggunakan rumus

$$K_1 = E(R_0) \oplus S_I$$

dimana S_I adalah kandidat input S-boxes, terdapat 2^{16} kandidat kunci putaran K_1.



Gambar 8.2: 2 Putaran DES

Dengan 2 putaran DES, jika L_0, R_0 dan L_2, R_2 diketahui, maka kita dapat gunakan rumus

$$\begin{aligned} f(R_0, K_1) &= L_0 \oplus R_1 \\ &= L_0 \oplus L_2 \end{aligned}$$

untuk mencari kandidat K_1 dan rumus

$$\begin{aligned} f(L_2, K_2) &= f(R_1, K_2) \\ &= L_1 \oplus R_2 \\ &= R_0 \oplus R_2 \end{aligned}$$

untuk mencari kandidat K_2. Seperti dengan 1 putaran, terdapat 2^{16} kandidat kunci putaran K_1. Untuk kunci putaran K_2 juga terdapat 2^{16} kandidat. Jika dependensi K_2 terhadap K_1 tidak digunakan maka kita harus mencoba 2^{32} kombinasi K_1 dan K_2, yang masih jauh lebih kecil dari jumlah kemungkinan kunci DES yaitu 2^{56}.

Sampai dengan 2 putaran, efek *confusion* dan *diffusion* belum terlalu besar sehingga masih dapat dianalisa secara naif. Akan tetapi untuk lebih dari

2 putaran, efek *confusion* dan *diffusion* semakin besar dan analisa menjadi semakin rumit. Nilai L dan R untuk putaran bukan pertama dan bukan terahir tidak dapat ditentukan dengan pasti dari nilai L dan R untuk putaran pertama dan terahir. Oleh karena itu diperlukan analisa yang lebih canggih seperti *differential cryptanalysis* atau *linear cryptanalysis*.

Cryptanalysis kerap digunakan untuk mencari kelemahan algoritma enkripsi. Tetapi, meskipun suatu algoritma dianggap dapat "dipecahkan" dengan *cryptanalysis*, ini belum tentu berarti bahwa pemecahan dapat dipraktekkan untuk mencari kunci enkripsi. Ini hanya berarti bahwa algoritma mempunyai "kelemahan." Sebagai contoh, dengan *linear cryptanalysis*, DES memerlukan 2^{47} pencarian menggunakan *known plaintext attack* dimana kita harus mempunyai naskah acak untuk 2^{47} naskah yang diketahui (secara *brute force* DES memerlukan 2^{55} pencarian). Akan tetapi dalam prakteknya sangat mustahil kita bisa mendapatkan sedemikian banyak naskah acak yang dienkripsi "musuh" untuk naskah yang kita ketahui.

8.1 Differential Cryptanalysis

Secara garis besar, *differential cryptanalysis* adalah teknik untuk mencari kunci enkripsi *block cipher* dari analisa efek perbedaan naskah asli terhadap perbedaan naskah acak. *Differential cryptanalysis* ditemukan oleh Eli Biham dan Adi Shamir, dan dalam bentuk dasarnya merupakan apa yang disebut *chosen-plaintext attack*. Namun jika cukup banyak naskah asli (*plaintext*) yang diketahui, maka teknik ini dapat digunakan untuk *known-plaintext attack* (lihat bagian 2.3.1).

Teknik *differential cryptanalysis* pada awalnya digunakan untuk menganalisa DES, dan DES dijadikan contoh untuk menjelaskan *differential cryptanalysis*. Pembaca dapat meninjau kembali bagian 7.1 mengenai DES. Disini kami hanya akan menjelaskan konsep-konsep dasar dari *differential cryptanalysis*. Bagi pembaca yang ingin memperdalam pengetahuan mengenai *differential cryptanalysis*, dianjurkan untuk membaca [bih91].

Konsep perbedaan dalam *differential cryptanalysis* dirumuskan dengan operasi *exclusive or*. Jadi perbedaan antara dua naskah asli P_1 dan P_2 adalah

$$P_1 \oplus P_2$$

dimana operasi *exclusive or* dilakukan secara *bitwise*. Jika C_1 dan C_2 adalah naskah acak untuk P_1 dan P_2, maka efek $P_1 \oplus P_2$ terhadap $C_1 \oplus C_2$ dapat memberikan informasi mengenai kunci enkripsi. Analisa mencoba mengeksploitasi kecenderungan fungsi *cipher* dan didasarkan pada sifat aljabar operasi *exclusive or*.

Efek dari permutasi seperti *initial permutation* (IP) adalah linear dengan

$$IP(P_1) \oplus IP(P_2) = IP(P_1 \oplus P_2),$$

jadi efek permutasi terhadap perbedaan tidak terlalu rumit. Permutasi yang dilakukan diluar putaran seperti IP dan IP^{-1} sama sekali tidak mempersulit analisa.

Mari kita lihat efek dari fungsi *cipher* f yang beroperasi terhadap setengah dari naskah sebesar 32 bit. Efek dari ekspansi E juga linear dengan

$$E(P_1) \oplus E(P_2) = E(P_1 \oplus P_2),$$

jadi ekspansi juga tidak membuat rumit perbedaan, jadi tidak mempengaruhi analisa satu putaran. Akan tetapi, ekspansi, yang selain mengekspansi juga melakukan permutasi, mempengaruhi tingkat kesulitan analisa lebih dari dua putaran karena efek *avalanche* yang ditimbulkannya. Efek *avalanche* terjadi karena perbedaan 1 bit dalam input setelah melewati S-box menjadi perbedaan sedikitnya 2 bit. Karena efek ekspansi, perbedaan 2 bit akan menjadi input 3 S-boxes dua putaran kemudian yang oleh 3 S-boxes dijadikan perbedaan 6 bit, dan seterusnya. Jadi dengan setiap putaran, efek perbedaan semakin besar bagaikan *avalanche*.

Efek dari operasi *exclusive or* dengan kunci putaran adalah

$$(P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2$$

yang berarti tidak ada efek terhadap perbedaan. Efek dari permutasi P juga linear, jadi yang sangat menentukan dalam *differential cryptanalysis* adalah efek dari substitusi S-box yang diketahui sebagai tidak linear.

Penjelasan konsep-konsep dasar *differential cryptanalysis* kami bagi menjadi 3 bagian:

1. Analisa 1 putaran.
2. Mekanisme n-round characteristic.
3. Penggunaan n-round characteristic.

8.1.1 Analisa 1 Putaran

Bagaimana kita mencari bits kunci putaran dari XOR pasangan input dan XOR pasangan output suatu S-box? Sebagai contoh kita umpamakan bahwa XOR pasangan input adalah $0x34$ (hexadecimal 34) dan XOR pasangan output adalah $0xd$ (hexadecimal d) dan S-box adalah S1. (Kita gunakan notasi $0x34 \rightarrow 0xd$ untuk menandakan bahwa XOR input $0x34$ dapat menghasilkan XOR output $0xD$.) Kita umpamakan juga bahwa bits pasangan hasil ekspansi E adalah $0x35$ dan $0x01$. Bits input untuk S1 didapat dari XOR bits hasil ekspansi E dengan bits kunci k_1. Jadi pasangan input S1, sebut saja x dan y mempunyai rumus:

$$\begin{aligned} x &= 0x35 \oplus k_1, \\ y &= 0x01 \oplus k_1, \end{aligned}$$

yang berarti

$$k_1 = 0x35 \oplus x = 0x01 \oplus y.$$

Jadi bits kunci putaran didapat dari XOR pasangan hasil ekspansi dengan pasangan input S1. Namun tidak semua pasangan input S1 dapat menghasilkan $0xd$ sebagai XOR output S1. Hanya ada 8 pasangan input x dan y dengan XOR $0x34$ yang menghasilkan XOR output $0xd$, oleh karena itu hanya ada 8 kandidat nilai bits kunci yang dimungkinkan seperti terlihat dalam tabel 8.1.

Pasangan input S1	Bits kunci putaran
$0x06$, $0x32$	$0x33$
$0x32$, $0x06$	$0x07$
$0x10$, $0x24$	$0x25$
$0x24$, $0x10$	$0x11$
$0x16$, $0x22$	$0x23$
$0x22$, $0x16$	$0x17$
$0x1c$, $0x28$	$0x29$
$0x28$, $0x1c$	$0x1d$

Tabel 8.1: Bits kunci untuk $0x34 \rightarrow 0xd$ dan ekspansi ($0x35$, $0x01$).

Jadi dengan menganalisa hasil transformasi S1 terhadap pasangan ekspansi $0x35$ dan $0x01$, ruang pencarian bits kunci putaran diperkecil dari 64 kandidat menjadi 8 kandidat.

Jika kita mempunyai pasangan ekspansi lain (mungkin dengan hasil XOR yang berbeda) yang menghasilkan tabel lain, kita dapat memperoleh informasi tambahan mengenai bits kunci putaran. Bits kunci putaran harus berada dalam semua tabel yang dihasilkan, jadi setelah mendapatkan tabel 8.2, kandidat untuk bits kunci putaran tinggal dua yaitu

$$0x23 \text{ dan } 0x17.$$

Analisa dapat dilanjutkan menggunakan pasangan ekspansi lainnya sampai kandidat bits kunci putaran tinggal satu sehingga bits kunci putaran dapat ditentukan.

Jika proses pencarian bits kunci putaran menggunakan analisa efek S-box tidak selesai, hasil analisa dapat digunakan untuk menentukan probabilitas berbagai kandidat bits kunci putaran. Pendekatan probabilistik inilah sebenarnya yang digunakan dalam *differential cryptanalysis*.

Secara garis besar, metode yang digunakan *differential cryptanalysis* untuk mencari kunci putaran adalah sebagai berikut:

1. Kita pilih XOR untuk naskah asli.

Pasangan input S1	Bits kunci putaran
$0x01$, $0x35$	$0x20$
$0x35$, $0x01$	$0x14$
$0x02$, $0x36$	$0x23$
$0x36$, $0x02$	$0x17$
$0x15$, $0x21$	$0x34$
$0x21$, $0x15$	$0x00$

Tabel 8.2: Bits kunci untuk $0x34 \rightarrow 0x3$ dan ekspansi ($0x21$, $0x15$).

2. Kita buat beberapa pasangan naskah asli dengan XOR yang dipilih, kita lakukan enkripsi terhadap pasangan, dan simpan pasangan terenkripsi.

3. Untuk setiap pasangan, cari XOR output yang diharapkan untuk sebanyak mungkin S-boxes untuk putaran terahir dari XOR naskah asli dan pasangan terenkripsi (XOR input fungsi *cipher f* untuk putaran terahir diketahui karena merupakan XOR bagian dari pasangan terenkripsi).

4. Untuk setiap kandidat kunci putaran, hitung pasangan yang menghasilkan XOR yang diharapkan jika menggunakan kandidat kunci putaran.

5. Kunci putaran yang terpilih adalah kandidat kunci putaran yang mempunyai hitungan terbesar.

8.1.2 Mekanisme n-round Characteristic

Sebelum melanjutkan penjelasan mekanisme yang digunakan, kita bahas dahulu model probabilistik yang digunakan dalam *differential cryptanalysis*. Untuk input, semua input yang dimungkinkan mempunyai probabilitas yang sama. Demikian juga dengan kunci putaran, semua kunci putaran mempunyai probabilitas a priori yang sama dan setiap kunci putaran adalah independen dari semua kunci putaran sebelumnya (jadi algoritma untuk *key scheduling* tidak berperan dalam model probabilistik yang digunakan). Meskipun kunci putaran DES sebenarnya tidak independen (algoritma *key scheduling* membuat relasi antar kunci putaran deterministik), asumsi ini menyederhanakan model probabilistik yang digunakan, tetapi ruang pencarian menjadi lebih besar. Dengan asumsi kunci putaran independen terdapat 2^{768} kombinasi kunci putaran untuk 16 putaran DES, sedangkan DES sebenarnya hanya mempunyai 2^{56} kombinasi kunci putaran. Tentunya dalam praktek *differential cryptanalysis*, pengetahuan mengenai algoritma *key scheduling* dapat digunakan untuk mempermudah analisa secara keseluruhan.

Kita mulai penjelasan mekanisme pada tingkat S-box. Setiap S-box mempunyai input 6 bit dan output 4 bit, jadi ada 64 nilai XOR input dan 16 nilai

XOR output. Setiap nilai 6 bit XOR input dapat berasal dari 64 pasangan yang berbeda, sebagai contoh

$$\begin{aligned} 000010 &\oplus 000001, \\ 000001 &\oplus 000010, \\ 000110 &\oplus 000101, \\ 000101 &\oplus 000110 \end{aligned}$$

dan ada 60 pasangan lainnya semua menghasilkan 000011 sebagai XOR input. Kecuali jika nilai XOR input adalah 000000, S-box dapat menghasilkan nilai XOR output yang berbeda untuk nilai XOR input yang sama tetapi dari pasangan yang berbeda. Sebagai contoh, dengan nilai $0x03$ (000011) sebagai XOR input, S1 akan menghasilkan XOR output $0x4$ (0100) untuk pasangan input $0x01$ dan $0x02$ sedangkan untuk pasangan input $0x21$ dan $0x22$ S1 akan menghasilkan XOR output $0xe$ (1110). Tabel 8.3 menunjukkan distribusi XOR output untuk S1 untuk nilai XOR input $0x00$, $0x01$, $0x02$ dan $0x03$ ($0x03$ berarti 03 dalam notasi hexadecimal, yang dalam notasi biner menjadi 000011). Semua pasangan input dengan $0x00$ sebagai nilai XOR input (ada 64 pasangan)

XOR	XOR output															
input	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
02	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
03	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
...	...															

Tabel 8.3: Tabel parsial distribusi XOR output untuk S1

menghasilkan $0x0$ sebagai XOR output. Untuk nilai $0x01$ sebagai XOR input, 6 pasangan input menghasilkan $0x3$ sebagai XOR output, 2 pasangan input menghasilkan $0x5$, 4 pasangan input menghasilkan $0x6$, dan tidak ada pasangan input yang dapat menghasilkan $0x0$, $0x1$, $0x2$, $0x4$ dan $0x8$. Secara rerata, hanya sekitar 80 persen dari nilai XOR output dimungkinkan oleh S1 dan S-box lainnya untuk setiap nilai XOR input. Tabel lengkap untuk distribusi XOR untuk semua S-box DES terdapat dalam [bih91], dan tabel juga dapat dikalkulasi berdasarkan spesifikasi S-box DES.

Jika suatu nilai XOR input, sebut saja X, dapat menghasilkan suatu nilai XOR output, sebut saja Y, untuk suatu S-box, maka kita katakan bahwa X dapat menyebabkan Y ($X \rightarrow Y$) dengan probabilitas $\frac{n_i}{64}$ dimana n_i adalah jumlah pasangan dengan XOR input X yang menghasilkan XOR output Y (n_i dapat diambil dari tabel distribusi XOR).

Konsep "menyebabkan" ($\rightarrow$) juga berlaku untuk transformasi fungsi *cipher* f dimana besar input dan output adalah 32 bit. Dengan X dan Y masing-masing sebesar 32 bit, kita katakan bahwa $X \rightarrow Y$ menggunakan fungsi *cipher* f dengan probabilitas

$$p = \frac{n_Y}{n}$$

dimana

- n_Y adalah jumlah semua kombinasi pasangan input dan kunci putaran dengan XOR input X dan menghasilkan XOR output Y, dan
- n adalah jumlah semua kombinasi pasangan input dan kunci putaran dengan XOR input X.

Ada 2^{32} pasangan input dengan XOR X karena input 32 bit, dan ada 2^{48} kunci putaran karena kunci putaran 48 bit, oleh sebab itu $n = 2^{80}$, jadi

$$p = \frac{n_Y}{2^{80}}.$$

Teorema 28 *Untuk DES, jika $X \rightarrow Y$ menggunakan fungsi cipher f dengan probabilitas p, maka untuk setiap pasangan input dengan XOR X, p juga merupakan probabilitas bahwa pasangan input akan menghasilkan XOR output Y. Jadi p adalah rasio jumlah kunci putaran yang mengakibatkan fungsi cipher f menghasilkan pasangan output dengan XOR Y, dibagi dengan jumlah semua kunci putaran.*

Untuk membuktikan teorema 28, kita mengetahui bahwa kunci putaran tidak mengubah XOR hasil ekspansi, jadi XOR input S-boxes sama dengan XOR hasil ekspansi. Kunci putaran mengubah pasangan tetapi tetap mempertahankan XOR pasangan. Jika i_1, i_2 adalah pasangan input fungsi *cipher* f dengan

$$i_1 \oplus i_2 = X$$

dan

$$\begin{aligned} e_1 &= E(i_1), \\ e_2 &= E(i_2) \end{aligned}$$

dan s_1, s_2 adalah pasangan input S-boxes yang menghasilkan pasangan o_1, o_2 sebagai output fungsi *cipher* f dengan

$$o_1 \oplus o_2 = Y$$

dan

$$s_1 \oplus s_2 = e_1 \oplus e_2$$

maka kunci putaran

$$k = e_1 \oplus s_1$$

dapat digunakan sehingga

$$\begin{aligned} s_1 &= e_1 \oplus k \text{ dan} \\ s_2 &= e_2 \oplus k. \end{aligned}$$

Karena pasangan output o_1, o_2 hanya tergantung pada pasangan s_1, s_2, setiap pasangan output dengan XOR Y mempunyai satu kunci putaran yang menghasilkan pasangan tersebut dari i_1, i_2. Karena setiap pasangan menggunakan kunci yang berbeda, banyaknya kunci putaran yang menghasilkan pasangan output dengan XOR Y dari pasangan i_1, i_2 sama dengan banyaknya pasangan output dengan XOR Y yang dapat dihasilkan dari pasangan i_1, i_2. Jadi

$$p = \frac{n_k}{2^{48}}$$

dimana n_k adalah banyaknya kunci putaran yang menghasilkan pasangan output dengan XOR Y dari pasangan i_1, i_2. Probabilitas ini konstan untuk semua pasangan input dengan XOR X jadi sama dengan probabilitas untuk semua pasangan input dengan XOR X secara rerata.

Kita dapat gabungkan probabilitas kedelapan S-box sebagai berikut. Jika

$$X_i \rightarrow Y_i \text{ dengan probabilitas } p_i$$

untuk $1 \leq i \leq 8$, dimana

$$\begin{aligned} X_1X_2X_3X_4X_5X_6X_7X_8 &= E(X) \text{ dan} \\ P(Y_1Y_2Y_3Y_4Y_5Y_6Y_7Y_8) &= Y \end{aligned}$$

maka

$$X \rightarrow Y$$

dengan probabilitas $p_1p_2p_3p_4p_5p_6p_7p_8$, jadi probabilitas untuk setiap S-box dikalikan untuk mendapatkan probabilitas untuk fungsi *cipher* f.

Mekanisme untuk menggabungkan hasil putaran (butir 3 sampai dengan 5 pencarian kunci putaran) menjadi hasil dari semua putaran dinamakan *n-round characteristic* oleh Biham dan Shamir. Suatu *n-round characteristic* Ω adalah

$$\Omega = (\Omega_P, \Omega_\Lambda, \Omega_T)$$

dimana

- Ω_P adalah XOR dari naskah asli sebesar m bit (64 bit untuk DES),

- $\Omega_\Lambda = (\Lambda_1, \Lambda_2, \ldots \Lambda_n)$,
  $\Lambda_i = (\lambda_I^i, \lambda_O^i)$ untuk $1 \leq i \leq n$,
  λ_I^i adalah XOR input fungsi *cipher* f sebesar $\frac{m}{2}$ bit untuk putaran i,
  λ_O^i adalah XOR output fungsi *cipher* f sebesar $\frac{m}{2}$ bit untuk putaran i, dan

- Ω_T adalah XOR dari naskah acak sebesar m bit.

Untuk setiap *n-round characteristic* Ω,

$$\begin{aligned}
\lambda_I^1 &= \text{belahan kanan dari } \Omega_P, \\
\lambda_I^2 &= \text{belahan kiri dari } \Omega_P \oplus \lambda_O^1, \\
\lambda_I^n &= \text{belahan kanan dari } \Omega_T, \\
\lambda_I^{n-1} &= \text{belahan kiri dari } \Omega_T \oplus \lambda_O^n, \text{ dan} \\
\lambda_O^i &= \lambda_I^{i-1} \oplus \lambda_I^{i+1} \text{ untuk } 2 \leq i \leq n-1.
\end{aligned}$$

Suatu *n-round characteristic* mempunyai probabilitas bahwa sembarang pasangan naskah asli yang mempunyai XOR sama dengan Ω_P memenuhi *n-round characteristic*, dengan asumsi sembarang kunci putaran yang independen digunakan untuk setiap putaran. Sebagai contoh, *1-round characteristic*

$$((L,0),((0,0)),(L,0))$$

mempunyai probabilitas 1 (dengan sembarang L), dan merupakan satu-satunya *n-round characteristic* yang mempunyai probabilitas lebih dari $\frac{1}{4}$.

Probabilitas suatu putaran i adalah probabilitas dari Λ_i (dengan notasi $P(\Lambda_i)$), yaitu probabilitas λ_I^i akan menghasilkan λ_O^i. Probabilitas ini telah dibahas dan menjadi subyek dari teorema 28. Probabilitas ini sama dengan probabilitas untuk

$$E(\lambda_I^i) \rightarrow P^{-1}(\lambda_O^i)$$

dimana E adalah ekspansi dalam fungsi *cipher* f dan P^{-1} adalah *inverse* dari permutasi P.

Probabilitas untuk *n-round characteristic* adalah

$$P(\Omega) = P(\Lambda_1)P(\Lambda_2)\ldots P(\Lambda_n).$$

Jadi untuk $\Omega = ((L,0),((0,0)),(L,0))$

$$P(\Omega) = P(0,0) = 1$$

karena probabilitas 0 akan menghasilkan 0 adalah 1 (jika tidak ada perbedaan dalam input maka dapat dipastikan tidak akan ada perbedaan dalam output).

Suatu pasangan input disebut *right pair* untuk *n-round characteristic* Ω dan kunci independen K (terdiri dari n kunci putaran yang independen), jika XOR

dari pasangan tersebut adalah Ω_P, dan untuk n putaran menggunakan kunci independen K, XOR input putaran i adalah λ_I^i dan XOR output fungsi *cipher* F adalah λ_O^i sesuai dengan Ω.

Teorema 29 *Probabilitas untuk characteristic Ω yang telah didefinisikan merupakan probabilitas aktual bahwa suatu pasangan input dengan XOR Ω_P adalah right pair jika sembarang kunci independen digunakan.*

Untuk membuktikan teorema 29, probabilitas bahwa pasangan input dengan XOR Ω_P adalah *right pair* merupakan probabilitas bahwa untuk setiap putaran i: $\lambda_I^i \rightarrow \lambda_O^i$. Probabilitas untuk setiap putaran bersifat independen dari bentuk persisnya pasangan input (asalkan menghasilkan XOR Ω_P, ini dibuktikan oleh teorema 28) dan independen dari apa yang telah terjadi di putaran sebelumnya (karena kunci independen mengacak input ke S-boxes, meskipun XOR input S-boxes tetap sama). Jadi probabilitas bahwa pasangan input merupakan *right pair* merupakan produk dari setiap probabilitas $\lambda_I^i \rightarrow \lambda_O^i$, jadi merupakan probabilitas untuk *characteristic* Ω.

8.1.3 Penggunaan n-round Characteristic

Untuk analisa 1 putaran (lihat bagian 8.1.1), kunci putaran dapat dicari dari perpotongan himpunan-himpunan kunci kandidat yang dimungkinkan berbagai pasangan input dan pasangan output. Untuk *n-round characteristic*, ini tidak dapat dilakukan karena perpotongan himpunan-himpunan tersebut biasanya kosong (himpunan yang dihasilkan pasangan input yang bukan *right pair* mungkin tidak memiliki kunci putaran yang sebenarnya sebagai elemen).

Menurut teorema 29, setiap pasangan input yang berupa *right pair*, yang muncul dengan probabilitas *characteristic*, akan menghasilkan kunci putaran yang sebenarnya sebagai kandidat. Kandidat lainnya terdistribusi secara acak, jadi jika semua kandidat dihitung kemunculannya, diharapkan kandidat dengan hitungan terbesar merupakan kunci putaran yang sebenarnya. Kunci putaran yang sebenarnya bagaikan *signal* sedangkan kandidat lainnya adalah *noise*. Semakin tinggi *signal-to-noise ratio*, semakin mudah analisa yang dibutuhkan, dimana *signal-to-noise ratio* S/N didefinisikan sebagai berikut:

$$S/N = \frac{\text{jumlah pasangan yang merupakan } \textit{right pair}}{\text{rerata hitungan per kandidat}}$$

Untuk mendapatkan kunci putaran yang sebenarnya, kita membutuhkan probabilitas *characteristic* yang cukup tinggi, dan pasangan input dan pasangan output yang cukup banyak untuk menjamin cukupnya jumlah pasangan input yang merupakan *right pair*. Dalam prakteknya, *n-round characteristic* dapat digunakan untuk mencari sebagian bits dari kunci putaran, tidak harus

sekaligus mencari semua bits kunci putaran. Banyaknya pasangan yang diperlukan tergantung pada jumlah bits kunci putaran yang dicari (k), probabilitas *characteristic* (p), dan jumlah pasangan yang dapat diabaikan karena bukan merupakan *right pairs*. Jika m adalah jumlah pasangan, α adalah rerata hitungan untuk semua pasangan yang dihitung, dan β adalah rasio pasangan yang dihitung dibandingkan dengan total pasangan, karena ada 2^k kandidat yang dihitung, maka rerata hitungan untuk setiap kandidat adalah

$$\frac{m \cdot \alpha \cdot \beta}{2^k}.$$

Rumus untuk S/N menjadi

$$\begin{aligned} S/N &= \frac{m \cdot p}{m \cdot \alpha \cdot \beta / 2^k} \\ &= \frac{2^k \cdot p}{\alpha \cdot \beta}. \end{aligned}$$

Jumlah pasangan *right pair* yang diperlukan untuk mendapatkan kunci putaran yang benar tergantung pada *signal-to-noise ratio*. Hasil empiris menurut Biham dan Shamir menunjukkan untuk rasio S/N sekitar $1 - 2$, 20 s/d 40 pasangan *right pair* diperlukan. Untuk rasio S/N jauh lebih besar, hanya 3 atau 4 pasangan *right pair* diperlukan. Untuk rasio S/N jauh lebih kecil, pasangan *right pair* yang diperlukan terlalu banyak sehingga tidak praktis.

Dalam penggunaannya, jumlah putaran n dalam *n-round characteristic* tidak harus sama dengan jumlah putaran dalam algoritma enkripsi yang dianalisa. Biasanya, untuk algoritma dengan m putaran,

$$n < m.$$

Sebagai contoh, Biham dan Shamir menggunakan *1-round characteristic* $\Omega = ((0x20000000, 0), ((0, 0)), (0x20000000, 0))$ untuk menganalisa DES yang diperlemah dari 16 putaran menjadi 4 putaran. Pembaca yang ingin melihat berbagai contoh *differential cryptanalysis* dipersilahkan untuk membaca [bih91].

8.1.4 Differential Cryptanalysis DES

Biham dan Shamir menggunakan teknik *differential cryptanalysis* terhadap DES dengan berbagai putaran. Tabel 8.4 memperlihatkan hasil dari analisa mereka. Untuk DES dengan 16 putaran (DES penuh), *differential cryptanalysis* lebih sukar daripada *brute force search*, jadi DES tahan terhadap *differential cryptanalysis*.

Putaran	Kompleksitas
4	2^4
6	2^8
8	2^{16}
9	2^{26}
10	2^{35}
11	2^{36}
12	2^{43}
13	2^{44}
14	2^{51}
15	2^{52}
16	2^{58}

Tabel 8.4: Hasil Differential Cryptanalysis DES

8.2 Linear Cryptanalysis

Linear cryptanalysis adalah teknik memecahkan enkripsi dengan cara membuat perkiraan linear untuk algoritma enkripsi. Kita harus mencari persamaan dalam bentuk

$$P[i_1] \oplus P[i_2] \oplus \ldots \oplus P[i_a] \oplus C[j_1] \oplus C[j_2] \oplus \ldots \oplus C[j_b] = K[k_1] \oplus K[k_2] \oplus \ldots \oplus K[k_c] \tag{8.1}$$

dimana $i_1, i_2 \ldots i_a$, $j_1, j_2 \ldots j_b$, dan $k_1, k_2 \ldots k_c$ adalah posisi bits tertentu, dan persamaan 8.1 mempunyai probabilitas $p \neq 0.5$ untuk sembarang naskah asli P dengan naskah acaknya C. Besar dari $|p - 0.5|$ merepresentasikan efektivitas dari persamaan 8.1. Setelah persamaan 8.1 ditemukan, kita dapat mencari informasi ekuivalen dengan satu bit kunci $K[k_1] \oplus K[k_2] \ldots K[k_c]$ menggunakan algoritma berdasarkan metode kemungkinan maksimal (*maximal likelihood method*) sebagai berikut (Algoritma 1):

1. Kita gunakan T sebagai representasi berapa kali ekspresi sebelah kiri persamaan 8.1 sama dengan 0.

2. Jika $T > N/2$ dimana N adalah jumlah naskah asli yang digunakan, maka kita perkirakan $K[k_1] \oplus K[k_2] \ldots K[k_c] = 0$ (jika $p > 0.5$) atau 1 (jika $p < 0.5$). Jika tidak maka kita perkirakan $K[k_1] \oplus K[k_2] \ldots K[k_c] = 1$ (jika $p > 0.5$) atau 0 (jika $p < 0.5$).

Tingkat kesuksesan metode ini jelas akan meningkat jika N atau $|p - 0.5|$ meningkat. Jadi persamaan yang paling efektif adalah persamaan yang mempunyai nilai maksimal untuk $|p - 0.5|$. Bagaimana kita mencari persamaan

yang efektif? Dalam papernya [mat93], Mitsuru Matsui memberi contoh bagaimana menemukan persamaan efektif untuk enkripsi DES. Contoh tersebut akan dicoba untuk dijelaskan disini.

Known-plaintext attack terhadap DES dapat dilakukan dengan persamaan paling efektif untuk $n-1$ putaran, jadi putaran terahir dianggap telah didekripsi menggunakan K_n dan persamaan dengan probabilitas terbaik ($|p-0.5|$ maksimal) dan mengandung F (fungsi *cipher f* DES) adalah sebagai berikut:

$$P[i_1, i_2, \ldots, i_a] \oplus C[j_1, j_2, \ldots, j_b] \oplus F_n(C_L, K_n)[l_1, l_2, \ldots, l_d] = K[k_1, k_2, \ldots, k_c] \tag{8.2}$$

dimana $P[i_1, i_2, \ldots, i_a]$ adalah singkatan untuk $P[i_1] \oplus P[i_2] \oplus \ldots \oplus P[i_a]$.

Efektivitas persamaan 8.2 tergantung pada pilihan untuk K_n (jika K_n yang salah dipilih maka efektivitas persamaan menurun drastis). Jadi metode kemungkinan maksimal dapat digunakan sebagai berikut (Algoritma 2):

1. Untuk setiap kandidat $K_n^{(i)}$ $(i = 1, 2, \ldots)$ dari K_n, kita gunakan T_i untuk merepresentasikan jumlah naskah asli yang mengakibatkan sebelah kiri dari persamaan 8.2 menjadi 0.

2. Jika T_{max} adalah nilai maksimal dari semua T_i dan T_{min} adalah nilai minimal dari semua T_i, maka

   - Jika $|T_{max} - N/2| > |T_{min} - N/2|$ maka kita pilih kandidat kunci yang mengakibatkan T_{max} dan kita pilih $K[k_1, k_2, \ldots k_c] = 0$ (jika $p > 0.5$) atau 1 (jika $p < 0.5$).
   - Jika $|T_{max} - N/2| < |T_{min} - N/2|$ maka kita pilih kandidat kunci yang mengakibatkan T_{min} dan kita pilih $K[k_1, k_2, \ldots k_c] = 1$ (jika $p > 0.5$) atau 0 (jika $p < 0.5$).

8.2.1 Perkiraan Linear untuk S-boxes

Untuk mendapatkan persamaan yang optimal dengan bentuk persamaan 8.2, kita analisa terlebih dahulu perkiraan linear untuk S-boxes. Ini dilakukan dengan mencari kecenderungan pada S-boxes.

Untuk setiap S-box S_a dengan $(a = 1, 2, \ldots, 8), 1 \leq \alpha \leq 63$ dan $1 \leq \beta \leq 15$, kita definisikan $NS_a(\alpha, \beta)$ sebagai berapa kali dari 64 kemungkinan pola input S_a XOR dari bits input yang di*mask* terlebih dahulu menggunakan α, sama dengan XOR dari bits output yang di*mask* terlebih dahulu menggunakan β. Jadi:

$$NS_a(\alpha, \beta) = \sharp\{x | 0 \leq x < 64, (\bigoplus_{s=0}^{5}(x[s] \bullet \alpha[s])) = (\bigoplus_{t=0}^{3}(S_a(x)[t] \bullet \beta[t]))\}$$

dimana • adalah *bitwise and* (yang digunakan untuk proses *masking*) dan ♯ adalah simbol untuk *cardinality* (besar dari himpunan). Notasi himpunan diatas menyeleksi semua bilangan bulat non-negatif dan lebih kecil dari 64 yang, jika dijadikan input untuk S_5, mengakibatkan persamaan antara kedua XOR terpenuhi.

Jika $NS_a(\alpha, \beta)$ tidak sama dengan 32, maka kita katakan bahwa ada korelasi antara input dan output S_a. Yang dicari adalah $NS_a(\alpha, \beta)$ yang memaksimalkan nilai $|NS_a(\alpha, \beta) - 32|$ (tabel untuk semua $NS_a(\alpha, \beta)$ dapat digunakan untuk mencari nilai maksimal). Sebagai contoh,

$$NS_5(16, 15) = 12$$

yang berarti bit input nomor 4 dari S_5 sama dengan XOR semua bit output S_5, dengan probabilitas $12/64 \approx 0.19$. Ini nilai optimal untuk $NS_a(\alpha, \beta)$. Dengan melibatkan ekspansi E dan permutasi P dalam fungsi *cipher* f DES, kita dapatkan persamaan

$$X[15] \oplus F(X, K)[7, 18, 24, 29] = K[22] \tag{8.3}$$

dengan probabilitas 0.19 untuk suatu kunci K yang ditetapkan dan suatu X yang terpilih secara acak. Karena $NS_a(16, 15)$ merupakan nilai optimal, persamaan 8.3 merupakan persamaan linear dengan probabilitas terbaik untuk S_5, jadi merupakan persamaan yang optimal.

8.2.2 Perkiraan Linear untuk DES

Setelah mendapatkan perkiraan linear yang baik untuk S-boxes, sekarang kita harus mencari perkiraan linear untuk algoritma DES secara keseluruhan. Sebagai contoh, kita gunakan algoritma DES dengan 3 putaran. Dengan mengaplikasikan persamaan 8.3 pada putaran pertama, kita dapatkan:

$$X_2[7, 18, 24, 29] \oplus P_H[7, 18, 24, 29] \oplus P_L[15] = K_1[22] \tag{8.4}$$

dengan probabilitas 12/64, dimana
X_2 merupakan input putaran kedua,
K_1 merupakan kunci putaran pertama,
P_L merupakan 32 bit pertama (*low 32 bits*) dari P, dan
P_H merupakan 32 bit terahir(*high 32 bits*) dari P.
Demikian juga pada putaran terahir, kita dapatkan:

$$X_2[7, 18, 24, 29] \oplus C_H[7, 18, 24, 29] \oplus C_L[15] = K_3[22] \tag{8.5}$$

juga dengan probabilitas 12/64. Kita kombinasikan kedua persamaan diatas untuk mendapatkan:

$$P_H[7, 18, 24, 29] \oplus C_H[7, 18, 24, 29] \oplus P_L[15] \oplus C_L[15] = K_1[22] \oplus K_3[22]. \tag{8.6}$$

Probabilitas bahwa persamaan 8.6 berlaku untuk sembarang naskah asli P dan naskah acaknya C adalah

$$(12/64)^2 + (1 - 12/64)^2 \approx 0.70.$$

Karena persamaan 8.3 merupakan perkiraan linear terbaik untuk fungsi *cipher* F, persamaan 8.6 merupakan persamaan terbaik untuk DES 3 putaran. Kita dapat gunakan algoritma 1 untuk mencari $K_1[22] \oplus K_3[22]$.

Seberapakah tingkat kesuksesan algoritma 1? Jika $|p - 1/2|$ cukup kecil, maka tingkat kesuksesan algoritma 1 dapat dirumuskan sebagai berikut:

$$\int_{-2\sqrt{N}|p-1/2|}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx.$$

Mari kita lihat bagaimana kita dapatkan rumus diatas. Algoritma 1 dapat dipandang sebagai eksperimen statistika dengan distribusi binomial:

- jumlah percobaan N adalah jumlah naskah asli yang digunakan, dan
- kita harapkan bahwa Np percobaan mematuhi persamaan 8.1.

Kita umpamakan bahwa $p > 0.5$ (untuk $p < 0.5$ analisanya sangat mirip karena simetris). Kita dapatkan

$$\begin{aligned} \mu &= Np \\ \sigma^2 &= Np(1-p). \end{aligned}$$

Algoritma 1 (dengan $p > 0.5$) dianggap sukses jika mayoritas naskah asli yang digunakan mematuhi persamaan 8.1, dengan kata lain jumlah naskah asli yang mematuhi persamaan 8.1 melebihi $N/2$. Tingkat kesuksesan algoritma 1 adalah rasio eksperimen yang sukses dari semua kemungkinan eksperimen. Jadi kita harus melihat distribusi μ untuk semua kemungkinan eksperimen (distribusi ini istilahnya *sampling distribution*). Menurut teori probabilitas, *sampling distribution* untuk kasus ini adalah distribusi normal dengan deviasi standard

$$\begin{aligned} \frac{\sigma}{\sqrt{N}} &= \frac{\sqrt{Np(1-p)}}{\sqrt{N}} \\ &= \sqrt{p(1-p)} \\ &\approx 1/2 \end{aligned}$$

untuk nilai $|p - 1/2|$ yang kecil. Kurva distribusi untuk kasus ini berbentuk lonceng, dan untuk keperluan integral, kita dapat menggeser kurva lonceng sehingga μ terletak pada titik nol. Titik representasi untuk $N/2$ harus dibagi dengan $\sqrt{N}$ dan digeser menjadi

$$\frac{N/2 - Np}{\sqrt{N}} = \sqrt{N}(1/2 - p).$$

Supaya integral dapat menggunakan rumus distribusi normal standard dengan deviasi standard 1, titik representasi $N/2$ harus dibagi lagi dengan deviasi standard diatas menjadi:

$$\begin{aligned}\frac{\sqrt{N}(1/2-p)}{1/2} &= 2\sqrt{N}(1/2-p)\\ &= -2\sqrt{N}|p-1/2|.\end{aligned}$$

Kurva lonceng standard dengan deviasi standard 1, μ terletak pada titik 0, dan representasi untuk $N/2$ terletak pada titik $-2\sqrt{N}|p-1/2|$, merepresentasikan distribusi hasil eksperimen untuk semua kemungkinan eksperimen yang menggunakan N naskah asli. Jadi representasi hasil eksperimen yang sukses (dan hanya eksperimen yang sukses) akan terletak disebelah kanan titik representasi untuk $N/2$, dan tingkat kesuksesan algoritma 1 adalah area dibawah kurva lonceng mulai dari titik representasi untuk $N/2$ kekanan. Oleh sebab itu integral dimulai dari titik $-2\sqrt{N}|p-1/2|$.

N	$\frac{1}{4}\|p-1/2\|^{-2}$	$\frac{1}{2}\|p-1/2\|^{-2}$	$\|p-1/2\|^{-2}$	$2\|p-1/2\|^{-2}$
Sukses	84.1%	92.1%	97,7%	99.8%

Tabel 8.5: Tingkat kesuksesan algoritma 1

Tabel 8.5 menunjukkan tingkat kesuksesan algoritma 1. Untuk DES 3 putaran menggunakan persamaan 8.6 sebagai perkiraan, tingkat kesuksesan 97.7% membutuhkan

$$\begin{aligned}N &= |p-1/2|^{-2}\\ &= |0.7-0.5|^{-2}\\ &= 25\end{aligned}$$

naskah asli.

Untuk DES 5 putaran, kita aplikasikan persamaan 8.3 pada putaran kedua dan keempat. Kita aplikasikan persamaan dibawah ini (yang mempunyai probabilitas 22/64)

$$X[27,28,30,31] \oplus F(X,K)[15] = K[42,43,45,46] \tag{8.7}$$

pada putaran pertama dan terahir. Kita dapat gabungkan keempat persamaan untuk menghasilkan

$$\begin{aligned}&P_H[15] \oplus P_L[7,18,24,27,28,29,30,31]\\ &\oplus C_H[15] \oplus C_L[7,18,24,27,28,29,30,31]\\ &= K_1[42,43,45,46] \oplus K_2[22] \oplus K_4[22] \oplus K_5[42,43,45,46].\end{aligned} \tag{8.8}$$

Bagaimana kita mencari probabilitas bahwa persamaan 8.8 berlaku untuk sembarang naskah asli P dengan naskah acaknya C? Kita gunakan rumus probabilitas penggabungan variabel acak biner yang independen

$$X_1 \oplus X_2 \oplus \ldots \oplus X_n = 0$$

dimana setiap X_i mempunyai probabilitas p_i untuk menjadi 0 (jadi mempunyai probabilitas $(1 - p_i)$ untuk menjadi 1). Rumusnya adalah sebagai berikut:

$$1/2 + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2).$$

Jadi untuk penggabungan empat persamaan diatas, kita dapatkan probabilitas

$$\begin{aligned} 1/2 + 2^{4-1} \prod_{i=1}^{4} (p_i - 1/2) &= 1/2 + 2^3 (12/64 - 32/64)^2 (22/64 - 32/64)^2 \\ &= 0.519 \end{aligned}$$

bahwa persamaan 8.8 berlaku untuk sembarang naskah asli P dengan naskah acaknya C. Untuk DES 5 putaran menggunakan persamaan 8.8 sebagai perkiraan, tingkat kesuksesan 97.7% membutuhkan

$$\begin{aligned} N &= |p - 1/2|^{-2} \\ &= |0.519 - 0.5|^{-2} \\ &= 2770 \end{aligned}$$

naskah asli.

Dalam papernya [mat93], Mitsuru Matsui memberikan tabel persamaan terbaik untuk DES dengan berbagai jumlah putaran. Untuk DES 16 putaran, diperlukan $|1.49 \times 2^{-24}|^{-2} \approx 2^{47}$ naskah asli untuk mencari 2 bit kunci. Di bagian berikut akan ditunjukkan cara untuk mendapatkan beberapa bit kunci sekaligus.

8.2.3 Known Plaintext Attack DES

Beberapa bit sekaligus dapat ditemukan untuk DES N putaran menggunakan algoritma 2 dengan persamaan paling efektif untuk $N - 1$ putaran. Sebagai contoh, untuk DES 8 putaran, kita gunakan persamaan paling efektif DES 7 putaran dikombinasikan dengan fungsi *cipher* F menjadi:

$$\begin{aligned} P_H[7, 18, 24] \oplus P_L[12, 16] \oplus C_H[15] \oplus C_L[7, 18, 24, 29] \oplus F_8(C_L, K_8)[15] \\ = K_1[19, 23] \oplus K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22]. \end{aligned} \tag{8.9}$$

Meskipun K_8 terdiri dari 48 bit, jumlah bit K_8 yang mempengaruhi hasil dari $F_8(C_L, K_8)[15]$ hanya ada 6, yaitu bit 42 sampai dengan 47, jadi diperlukan $2^6 = 64$ *counters* (yang merepresentasikan 64 kandidat kunci parsial 6 bit) untuk algoritma 2.

Analisa tingkat kesuksesan algoritma 2 agak sulit dan tidak akan dibahas disini. Pembaca yang ingin mendalami analisa tersebut dipersilahkan untuk membaca [mat93] yang memuat tabel 8.6 sebagai tingkat kesuksesan algoritma 2.

N	$2\|p-1/2\|^{-2}$	$4\|p-1/2\|^{-2}$	$8\|p-1/2\|^{-2}$	$16\|p-1/2\|^{-2}$
Sukses	48.6%	78.5%	96.7%	99.9%

Tabel 8.6: Tingkat kesuksesan algoritma 2

Jadi untuk tingkat kesuksesan 96.7% diperlukan sekitar $8|1.95 \times 2^{-10}|^{-2} \approx 2^{21}$ naskah asli. Karena sifat simetris yang terdapat pada putaran proses enkripsi DES, terdapat 2 persamaan paling efektif yang dapat digunakan untuk mendapatkan 12 bit kunci. Ditambah dengan 2 bit kunci yang didapatkan menggunakan algoritma 1, total 14 bit kunci didapatkan dari *linear cryptanalysis*. Sisanya, $56 - 14 = 42$ bit kunci dapat dicari dengan cara *exhaustive search*.

Untuk DES 16 putaran penuh, diperlukan $8|1.19 \times 2^{-22}|^{-2} \approx 2^{47}$ naskah asli menggunakan persamaan:

$$\begin{aligned} P_H[7,18,24] \oplus P_L[12,16] \oplus C_H[15] \oplus C_L[7,18,24,29] \oplus F_{16}(C_L, K_{16})[15] \\ = K_1[19,23] \oplus K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] \oplus K_8[44] \\ \oplus K_9[22] \oplus K_{11}[22] \oplus K_{12}[44] \oplus K_{13}[22] \oplus K_{15}[22]. \end{aligned} \tag{8.10}$$

8.3 Pelajaran dari Cryptanalysis DES

Meskipun secara teoritis *linear cryptanalysis* menunjukkan sedikit kelemahan DES, secara umum dan praktis DES cukup tahan terhadap *differential cryptanalysis* dan *linear cryptanalysis*. Ini membantu menghilangkan kecurigaan bahwa NSA sengaja membuat DES lemah agar mereka dapat memecahkannya. (Tetapi dengan kemajuan teknologi komputer sekarang DES rentan terhadap serangan *brute force search*.) Sebaliknya, berbagai macam *block cipher* yang dirancang sebelum DES ternyata sangat rentan terhadap *differential* dan *linear cryptanalysis*. Pelajaran ini diterapkan dalam merancang *block cipher* modern yang mewajibkan *strict avalanche criterion* untuk setiap fungsi dalam enkripsi dimana perubahan 1 bit input menyebabkan setiap bit output mempunyai probabilitas 0.5 untuk berubah, independen dari bit lainnya.

8.4 Ringkasan

Dalam bab ini kita telah bahas dua teknik *cryptanalysis*. Teknik pertama untuk mencari kecenderungan dalam fungsi enkripsi adalah *differential cryptanalysis*. Teknik kedua adalah *linear cryptanalysis*. Perancangan *block cipher* kini memperhatikan ketahanan terhadap dua teknik *cryptanalysis* tersebut.

Bab 9

Cryptographically Secure Hashing

Cryptographically secure hashing adalah proses pembuatan suatu "sidik jari" (*fingerprint* atau kerap juga disebut *digest*) untuk suatu naskah. Sidik jari relatif tidak terlalu besar, antara 128 hingga 512 bit tergantung algoritma yang digunakan, sedangkan besar naskah tidak terbatas. Contoh penggunaan sidik jari adalah untuk *digital signature* (lihat bab 16). Agar proses pembuatan sidik jari aman dari segi kriptografi, kemungkinan *collision*, dimana dua naskah yang berbeda mempunyai sidik jari yang sama, harus sangat kecil. Kriteria yang sudah menjadi standard untuk keamanan algoritma *secure hashing* adalah:

- *Preimage resistance.* Untuk suatu nilai *hash* yang sembarang (tidak diketahui asal-usulnya), sangat sukar untuk mencari naskah yang mempunyai nilai *hash* tersebut.

- *Second preimage resistance.* Untuk suatu naskah m_1, sangat sukar untuk mencari naskah lain m_2 ($m_1 \neq m_2$) yang mempunyai nilai *hash* yang sama ($hash(m_1) = hash(m_2)$). Persyaratan ini kerap disebut juga *weak collision resistance.*

- *Collision resistance.* Sangat sukar untuk mencari dua naskah m_1 dan m_2 yang berbeda ($m_1 \neq m_2$) yang mempunyai nilai *hash* yang sama ($hash(m_1) = hash(m_2)$). Persyaratan ini kerap disebut juga *strong collision resistance.*

Algoritma untuk *secure hashing* biasanya membagi naskah sehingga terdiri dari beberapa blok, setiap blok biasanya 512 atau 1024 bit. Naskah diberi *padding* meskipun besarnya merupakan kelipatan dari besarnya blok dan

padding diahiri dengan *size* dari naskah. Algoritma biasanya terdiri dari dua tahap:

- *preprocessing* dan
- *hashing*.

Tahap *preprocessing* biasanya terdiri dari *padding* dan *parameter setup*. Tahap *hashing* membuat sidik jari dengan mengkompresi naskah yang sudah diberi *padding*. Kompresi dilakukan dengan setiap blok secara berurut diproses dan hasilnya dijadikan *feedback* untuk proses blok berikutnya. Konstruksi seperti ini dinamakan konstruksi Merkle-Damgård (lihat [mer79] bab II), dan digunakan oleh MD5 dan SHA, dua algoritma *secure hashing* yang akan dibahas.

Tentunya keamanan algoritma *secure hashing* sangat tergantung pada proses kompresi yang digunakan. Selain 3 kriteria *resistance* yang telah dibahas, dewasa ini *secure hashing* juga diharapkan memiliki resistensi terhadap *length extension attack*. *Length extension attack* adalah *attack* dimana dari mengetahui $hash(m_1)$ dan panjangnya naskah m_1, tanpa mengetahui m_1, seseorang dapat membuat naskah m_2 dan $hash(m_1 \circ m_2)$, dimana $\circ$ adalah operasi penyambungan naskah (*concatenation*). *Attack* ini digunakan terhadap *authentication* yang menggunakan *hashing* yang lemah. Sementara ini, untuk mengatasi masalah *length extension attack*, mekanisme HMAC (lihat bagian 9.3) kerap digunakan. Ditingkat yang lebih rinci, kriteria keamanan yang digunakan untuk merancang *block cipher* seperti *strict avalanche criterion*, dimana perbedaan input 1 bit menyebabkan setiap bit output mempunyai probabilitas 0.5 untuk berubah independen dari bit lainnya, sebaiknya juga berlaku untuk proses kompresi.

Penggunaan utama *secure hashing* memang untuk *digest*, contohnya untuk *digital signature*. Namun selain untuk keperluan *digest*, *secure hashing* kerap juga digunakan sebagai *pseudorandom function*. Salah satu contoh aplikasi *pseudorandom function* adalah untuk *key derivation*, seperti yang dilakukan oleh protokol Kerberos (lihat bagian 22.1) dan juga protokol IKE dalam IPsec (lihat bagian 20.3).

Saat bab ini ditulis, NIST sedang mengadakan sayembara untuk pembuatan standard SHA-3. Salah satu sebab NIST merasa standard SHA baru perlu dibuat adalah ditemukannya *collision* untuk MD5 oleh Xiaoyun Wang dan koleganya (lihat [wan05] dan juga bagian 9.1). Waktu komputer yang dibutuhkan untuk membuat *collision* tidak terlalu lama, jadi MD5 tidak memenuhi kriteria diatas. Xiaoyun Wang dan koleganya juga berhasil menemukan kelemahan pada SHA-1. NIST mengharapkan bahwa SHA-3 akan memenuhi 4 kriteria *resistance* yang telah dibahas, ditambah dengan persyaratan bahwa jika hanya sebagian dari semua bit digunakan (tentunya banyaknya bit yang digunakan harus cukup besar), maka algoritma tetap memenuhi keempat kriteria.

9.1 MD5

Algoritma *secure hashing* MD5 dirancang oleh Ron Rivest dan penggunaannya sangat populer dikalangan komunitas *open source* sebagai *checksum* untuk *file* yang dapat di *download*. MD5 juga kerap digunakan untuk menyimpan password dan juga digunakan dalam *digital signature* dan *certificate*.

Spesifikasi lengkap untuk algoritma MD5 ada pada suatu RFC (*request for comment*, lihat [riv92]). Besarnya blok untuk MD5 adalah 512 bit sedangkan *digest size* adalah 128 bit. Karena *word size* ditentukan sebesar 32 bit, satu blok terdiri dari 16 *word* sedangkan *digest* terdiri dari 4 *word*. Indeks untuk bit dimulai dari 0. *Preprocessing* dimulai dengan *padding* sebagai berikut:

1. Bit dengan nilai 1 ditambahkan setelah ahir naskah.

2. Deretan bit dengan nilai 0 ditambahkan setelah itu sehingga besar dari naskah mencapai nilai 448 (mod 512) (sedikitnya 0 dan sebanyaknya 511 bit dengan nilai 0 ditambahkan sehingga tersisa 64 bit untuk diisi agar besar naskah menjadi kelipatan 512).

3. 64 bit yang tersisa diisi dengan besar naskah asli dalam bit. Jika besar naskah asli lebih dari 2^{64} bit maka hanya 64 *lower order* bit yang dimasukkan. *Lower order word* untuk besar naskah asli dimasukkan sebelum *high-order word*.

Setelah *padding*, naskah terdiri dari n *word* $M[0 \ldots n-1]$ dimana n adalah kelipatan 16. Langkah berikutnya dalam *preprocessing* adalah menyiapkan MD *buffer* sebesar 4 word:

$$(A, B, C, D)$$

dimana A merupakan *lower order word*. *Buffer* diberi nilai awal sebagai berikut (nilai dalam hexadecimal dimulai dengan *lower order byte*).

A:	01	23	45	67
B:	89	*ab*	*cd*	*ef*
C:	*fe*	*dc*	*ba*	98
D:	76	54	32	10.

Proses *hashing* dilakukan per blok, dengan setiap blok melalui 4 putaran. Proses *hashing* menggunakan 4 fungsi $F, G, H,$ dan I yang masing-masing mempunyai input 3 *word* dan output 1 *word*:

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X,Y,Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) \\ H(X,Y,Z) &= X \oplus Y \oplus Z \\ I(X,Y,Z) &= Y \oplus (X \vee \neg Z) \end{aligned}$$

dimana $\wedge$ adalah *bitwise and*, $\vee$ adalah *bitwise or*, $\oplus$ adalah *bitwise exclusive or*, dan $\neg$ adalah *bitwise not* (*one's complement*). Selain keempat fungsi diatas, proses *hashing* juga menggunakan tabel dengan 64 *word*, $T[1]$ sampai dengan $T[64]$, yang berada dalam tabel D.1 di appendix D.

Secara garis besar, algoritma untuk *hashing* untuk satu blok adalah sebagai berikut (menggunakan array 16 *word* $X[0]$ sampai dengan $X[15]$ yang dapat menyimpan satu blok):

1. *Copy* satu blok (*word* $M[16i]$ sampai dengan $M[16i+15]$) ke $X[0]$ sampai dengan $X[15]$.
2. Simpan A, B, C, D dalam A', B', C', D'.
3. Lakukan putaran 1 pada A, B, C, D.
4. Lakukan putaran 2 pada A, B, C, D.
5. Lakukan putaran 3 pada A, B, C, D.
6. Lakukan putaran 4 pada A, B, C, D.
7. Tambahkan nilai simpanan pada A, B, C, D:

$$\begin{aligned} A &\leftarrow (A + A') \bmod 2^{32}, \\ B &\leftarrow (B + B') \bmod 2^{32}, \\ C &\leftarrow (C + C') \bmod 2^{32}, \\ D &\leftarrow (D + D') \bmod 2^{32}. \end{aligned}$$

Algoritma diatas diulang hingga semua blok dalam M terproses (dimulai dengan $i = 0$ sampai dengan $i = n/16 - 1$).

Sekarang kita jelaskan putaran 1 sampai dengan 4 yang dilakukan pada A, B, C, D. Setiap putaran menggunakan operasi berbeda. Untuk putaran 1, operasi $P_1(A, B, C, D, k, s, i)$ didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + F(B, C, D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi X kekiri s bit. Putaran 1 terdiri dari:

$$\begin{aligned} P_1(A, B, C, D, 0, 7, 1), \quad & P_1(D, A, B, C, 1, 12, 2), \\ P_1(C, D, A, B, 2, 17, 3), \quad & P_1(B, C, D, A, 3, 22, 4), \\ P_1(A, B, C, D, 4, 7, 5), \quad & P_1(D, A, B, C, 5, 12, 6), \\ P_1(C, D, A, B, 6, 17, 7), \quad & P_1(B, C, D, A, 7, 22, 8), \\ P_1(A, B, C, D, 8, 7, 9), \quad & P_1(D, A, B, C, 9, 12, 10), \\ P_1(C, D, A, B, 10, 17, 11), \quad & P_1(B, C, D, A, 11, 22, 12), \\ P_1(A, B, C, D, 12, 7, 13), \quad & P_1(D, A, B, C, 13, 12, 14), \\ P_1(C, D, A, B, 14, 17, 15), \quad & P_1(B, C, D, A, 15, 22, 16). \end{aligned}$$

Untuk putaran 2, operasi $P_2(A, B, C, D, k, s, i)$ didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + G(B, C, D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi X kekiri s bit. Putaran 2 terdiri dari:

$$\begin{array}{rr}
P_2(A, B, C, D, 1, 5, 17), & P_2(D, A, B, C, 6, 9, 18), \\
P_2(C, D, A, B, 11, 14, 19), & P_2(B, C, D, A, 0, 20, 20), \\
P_2(A, B, C, D, 5, 5, 21), & P_2(D, A, B, C, 10, 9, 22), \\
P_2(C, D, A, B, 15, 14, 23), & P_2(B, C, D, A, 4, 20, 24), \\
P_2(A, B, C, D, 9, 5, 25), & P_2(D, A, B, C, 14, 9, 26), \\
P_2(C, D, A, B, 3, 14, 27), & P_2(B, C, D, A, 8, 20, 28), \\
P_2(A, B, C, D, 13, 5, 29), & P_2(D, A, B, C, 2, 9, 30), \\
P_2(C, D, A, B, 7, 14, 31), & P_2(B, C, D, A, 12, 20, 32).
\end{array}$$

Untuk putaran 3, operasi $P_3(A, B, C, D, k, s, i)$ didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + H(B, C, D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi X kekiri s bit. Putaran 3 terdiri dari:

$$\begin{array}{rr}
P_3(A, B, C, D, 5, 4, 33), & P_3(D, A, B, C, 8, 11, 34), \\
P_3(C, D, A, B, 11, 16, 35), & P_3(B, C, D, A, 14, 23, 36), \\
P_3(A, B, C, D, 1, 4, 37), & P_3(D, A, B, C, 4, 11, 38), \\
P_3(C, D, A, B, 7, 16, 39), & P_3(B, C, D, A, 10, 23, 40), \\
P_3(A, B, C, D, 13, 4, 41), & P_3(D, A, B, C, 0, 11, 42), \\
P_3(C, D, A, B, 3, 16, 43), & P_3(B, C, D, A, 6, 23, 44), \\
P_3(A, B, C, D, 9, 4, 45), & P_3(D, A, B, C, 12, 11, 46), \\
P_3(C, D, A, B, 15, 16, 47), & P_3(B, C, D, A, 2, 23, 48).
\end{array}$$

Untuk putaran 4, operasi $P_4(A, B, C, D, k, s, i)$ didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + I(B, C, D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi X kekiri s bit. Putaran 4 terdiri dari:

$$\begin{array}{rr}
P_4(A, B, C, D, 0, 6, 49), & P_4(D, A, B, C, 7, 10, 50), \\
P_4(C, D, A, B, 14, 15, 51), & P_4(B, C, D, A, 5, 21, 52), \\
P_4(A, B, C, D, 12, 6, 53), & P_4(D, A, B, C, 3, 10, 54), \\
P_4(C, D, A, B, 10, 15, 55), & P_4(B, C, D, A, 1, 21, 56), \\
P_4(A, B, C, D, 8, 6, 57), & P_4(D, A, B, C, 15, 10, 58), \\
P_4(C, D, A, B, 6, 15, 59), & P_4(B, C, D, A, 13, 21, 60), \\
P_4(A, B, C, D, 4, 6, 61), & P_4(D, A, B, C, 11, 10, 62), \\
P_4(C, D, A, B, 2, 15, 63), & P_4(B, C, D, A, 9, 21, 64).
\end{array}$$

Setelah semua blok diproses, maka hasil ahir A, B, C, D menjadi MD5 *digest* dari naskah asli. Urutan byte untuk *digest* dimulai dengan *lower order byte* dari A dan diahiri oleh *higher order byte* dari D.

Beberapa peneliti telah berhasil membuat *collision* untuk MD5. Xiaoyun Wang dan koleganya berhasil membuat *collision* untuk sepasang naskah yang masing-masing terdiri dari 2 blok (lihat [wan05]). Kita akan bahas esensi dari metode yang digunakan untuk membuat *collision* tersebut.

Wang menggunakan teknik *differential cryptanalysis* dengan dua macam perbedaan:

- perbedaan bit (*exclusive or*) dan
- selisih (menggunakan pengurangan).

Jika *differential cryptanalysis* terhadap DES berfokus pada perbedaan bit (lihat bagian 8.1), metode Wang menggunakan kombinasi selisih dan perbedaan bit, tetapi lebih berfokus pada selisih. Kombinasi ini menghasilkan perbedaan yang lebih rinci. Sebagai contoh, jika dua *word* X dan X', yang masing-masing besarnya 32 bit, mempunyai selisih $X' - X = 2^6$, perbedaan bit antara X dan X' bisa berupa

- perbedaan 1 bit ($X'_7 = 1$ dan $X_7 = 0$), atau
- perbedaan 2 bit ($X'_{8-7} = 10$ dan $X_{8-7} = 01$), atau
- perbedaan 3 bit ($X'_{9-7} = 100$ dan $X_{9-7} = 011$),
- dan seterusnya.

Differential dari X dan X' didefinisikan sebagai

$$\Delta X = X' - X.$$

Jika naskah $M = (M_0, M_1, \ldots, M_{k-1})$ dan naskah $M' = (M'_0, M'_1, \ldots, M'_{k-1})$, maka *full differential* dari fungsi hash H untuk M dan M' adalah

$$\Delta H_0 \overset{(M_0, M'_0)}{\longrightarrow} \Delta H_1 \overset{(M_1, M'_1)}{\longrightarrow} \ldots \Delta H_{k-1} \overset{(M_{k-1}, M'_{k-1})}{\longrightarrow} \Delta H,$$

dimana ΔH_0 adalah perbedaan awal (jadi sama dengan 0). ΔH adalah perbedaan output antara M dan M', dan $\Delta H_i = \Delta IV_i$ adalah perbedaan output untuk tahap i yang juga merupakan nilai awal untuk tahap berikutnya. Cukup jelas bahwa jika $\Delta H = 0$, maka kita dapatkan *collision*. *Differential* yang mengakibatkan *collision* disebut *collision differential*. Untuk lebih rinci, *differential* tahap i, $\Delta H_i \overset{(M_i, M'_i)}{\longrightarrow} \Delta H_{i+1}$, dapat diuraikan menjadi

$$\Delta H_i \xrightarrow{P_1} \Delta R_{i+1,1} \xrightarrow{P_2} \Delta R_{i+1,2} \xrightarrow{P_3} \Delta R_{i+1,3} \xrightarrow{P_4} \Delta R_{i+1,4} = \Delta H_{i+1},$$

dimana $\Delta R_{i+1,j}$ adalah perbedaan output untuk putaran j. Untuk lebih rinci lagi, *differential* putaran j, $\Delta R_{j-1} \xrightarrow{P_j} \Delta R_j$ dengan probabilitas P_j, dimana $j = 1, 2, 3, 4$, dapat diuraikan menjadi

$$\Delta R_{j-1} \xrightarrow{P_{j1}} \Delta X_1 \xrightarrow{P_{j2}} \ldots \xrightarrow{P_{j16}} \Delta X_{16} = \Delta R_j,$$

dimana $\Delta_{t-1} \xrightarrow{P_{jt}} \Delta X_t$ merupakan *differential characteristic* untuk langkah t dalam putaran j, $t = 1, 2, \ldots, 16$. Jika P adalah probabilitas untuk *differential* $\Delta H_i \xrightarrow{(M_i, M_i')} \Delta H_{i+1}$, maka

$$P \geq \prod_{i=1}^{4} P_i \text{ dan } \forall j \in \{1, 2, 3, 4\} : P_j \geq \prod_{t=1}^{16} P_{jt}.$$

Metode Wang menggunakan *collision differential* dua tahap

$$\Delta H_0 \xrightarrow{(M_0, M_0')} \Delta H_1 \xrightarrow{(M_1, M_1')} \Delta H,$$

dimana

$$\begin{aligned}
\Delta M_0 = M_0' - M_0 &= (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, 2^{15}, 0, 0, 2^{31}, 0), \\
\Delta M_1 = M_1' - M_1 &= (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, -2^{15}, 0, 0, 2^{31}, 0), \\
\Delta H_1 &= (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25}).
\end{aligned}$$

Nilai awal buffer, yaitu (a_0, b_0, c_0, d_0), digunakan untuk IV_0, dimana

$$\begin{aligned}
a_0 &= 0x67452301, \\
b_0 &= 0xefcdab89, \\
c_0 &= 0x98badcfe, \\
d_0 &= 0x10325476.
\end{aligned}$$

Untuk meningkatkan probabilitas *differential characteristic* dalam penggunaannya, naskah yang dibuat secara acak dapat dimodifikasi. Untuk itu Wang pertama membuat persyaratan yang cukup (*sufficient condition*) agar berbagai *characteristic* dijamin berlaku dengan probabilitas 1. Kemudian teknik untuk melakukan modifikasi naskah agar memenuhi berbagai persyaratan dibuat.

Untuk membuat persyaratan yang cukup agar suatu *characteristic* dijamin berlaku dengan probabilitas 1, diperlukan *data flow analysis*. Sebagai contoh, *differential characteristic* yang digunakan Wang untuk langkah 8 pada putaran pertama tahap pertama adalah

$$(\Delta a_2, \Delta b_1, \Delta c_2, \Delta d_2) \longrightarrow (\Delta a_2, \Delta b_2, \Delta c_2, \Delta d_2),$$

dimana perbedaan variabel mematuhi persamaan-persamaan sebagai berikut:

$$\begin{aligned} b'_1 &= b_1, \\ a'_2 &= a_2[7, \ldots, 22, -23], \\ d'_2 &= d_2[-7, 24, 32], \\ c'_2 &= c_2[7, 8, 9, 10, -12, -24, -25, -26, 27, 28, 29, 30, 31, 32, 1, 2, 3, 4, 5, -6], \\ b'_2 &= b_2[1, 16, -17, 18, 19, 20, -21, -24]. \end{aligned}$$

Sedikit penjelasan mengenai notasi diatas, $a'_2 = a_2[7, \ldots, 22, -23]$ berarti bit 7 sampai dengan 22 dari a_2 adalah 0 sedangkan bit 7 sampai dengan 22 dari a'_2 adalah 1, dan bit 23 dari a_2 adalah 1 sedangkan bit 23 dari a'_2 adalah 0. Bit yang tidak disebut, nilainya sama pada a'_2 dan a_2. Efek dari langkah ke 8 adalah sebagai berikut;

$$\begin{aligned} b_2 &= c_2 + ((b_1 + F(c_2, d_2, a_2) + m_7 + T[8]) <<< 22), \\ b'_2 &= c'_2 + ((b_1 + F(c'_2, d'_2, a'_2) + m'_7 + T[8]) <<< 22), \end{aligned}$$

dimana m_7 dan m'_7 adalah *word* ke 8 untuk masing-masing blok. Kita gunakan notasi ϕ_7:

$$\phi_7 = F(c_2, d_2, a_2) = (c_2 \wedge d_2) \vee (\neg c_2 \wedge a_2).$$

Pencarian persyaratan didasarkan pada fakta bahwa $\Delta b_1 = 0$ dan $\Delta m_7 = 0$ (berdasarkan *differential characteristic* ΔM_0 dan ΔM_1, hanya Δm_4, Δm_{11} dan Δm_{14} yang tidak sama dengan nol). Jadi

$$\Delta b_2 = \Delta c_2 + (\Delta \phi_7 <<< 22).$$

Persyaratan untuk bit pertama dari $\Delta b2$ ($\Delta b_{2,1}$) adalah $d_{2,11} = \overline{a_{2,11}} = 1$ dan $b_{2,1} = 0$ dikarenakan

1. Jika $d_{2,11} = \overline{a_{2,11}} = 1$, maka $\Delta \phi_{7,11} = 1$.
2. $\Delta \phi_{7,11} = (\Delta \phi_7 <<< 22)_{11}$.
3. Karena $\Delta c_{2,1} = 0$, maka $\Delta b_{2,1} = 0 + 1 = 1$.

Persyaratan untuk semua bit Δb_2, baik yang 0 maupun 1 dirumuskan oleh Wang. Wang merumuskan semua persyaratan agar

$$\Delta H_0 \xrightarrow{1} \Delta R_{1,1}$$

dan

$$\Delta H_1 \xrightarrow{1} \Delta R_{2,1},$$

dengan kata lain *characteristic* untuk 1 putaran MD5 dijamin dengan probabilitas 1.

Agar persyaratan *characteristic* dapat dipenuhi, Wang melakukan modifikasi terhadap naskah yang dibuat secara acak. Sebagai contoh, salah satu persyaratan agar *characteristic* 1 putaran terjamin adalah untuk c_1 yaitu

$$c_{1,7} = 0, c_{1,12} = 0, c_{1,20} = 0.$$

Untuk memenuhi persyaratan tersebut, m_2 dimodifikasi sebagai berikut:

$$\begin{aligned} c_1^{new} &\leftarrow c_1^{old} - c_{1,7}^{old} - c_{1,12}^{old} \cdot 2^{11} - c_{1,20}^{old} \cdot 2^{19}, \\ m_2^{new} &\leftarrow ((c_1^{new} - c_1^{old}) >>> 17) + m_2^{old}, \end{aligned}$$

dimana m_2^{old} adalah nilai m_2 sebelum modifikasi, m_2^{new} adalah nilai m_2 setelah modifikasi, c_1^{old} adalah nilai c_1 sebelum modifikasi, dan c_1^{new} adalah nilai c_1 setelah modifikasi.

Modifikasi naskah dilakukan agar *characteristic* putaran pertama dijamin mempunyai probabilitas 1, baik untuk tahap 1 maupun tahap 2. Modifikasi naskah juga dilakukan agar sebagian dari persyaratan untuk putaran kedua terpenuhi. Dengan melakukan berbagai modifikasi tersebut, probabilitas *characteristic* tahap pertama ditingkatkan menjadi 2^{-37} sedangkan probabilitas *characteristic* tahap kedua ditingkatkan menjadi 2^{-30}. Menggunakan metode ini, Wang berhasil menemukan *collision* untuk MD5 dalam waktu kurang dari 1 jam dengan komputer.

Bersama Arjen Lenstra dan Benne de Weger, Xiaoyun Wang berhasil membuat *collision* untuk sepasang X.509 *certificate* dimana bagian *certificate* yang ditanda-tangan berbeda tetapi mempunyai MD5 *digest* yang sama [len05]. Sebagai konsekuensinya, kebenaran dari suatu *certificate* dapat diragukan karena bagian yang ditanda-tangan dapat ditukar dengan nilai lain (termasuk kunci publik lain) tanpa perubahan pada nilai *digital signature*. Jadi sebaiknya *digital signature* menggunakan algoritma *secure hashing* lain seperti SHA-1[1].

9.2 SHA

Algoritma *secure hashing* SHA dirancang oleh National Security Agency (NSA) dan dijadikan standard FIPS (lihat [sha02]). Ada 4 varian SHA dalam standard FIPS-180-2 dengan parameter yang berbeda (lihat tabel 9.1).

Keamanan algoritma didasarkan pada fakta bahwa *birthday attack* pada *digest* sebesar n bit menghasilkan *collision* dengan faktor kerja sekitar $2^{n/2}$. Kita hanya akan membahas SHA-1 disini karena SHA-256, SHA-384 dan SHA-512 algoritmanya mirip dengan SHA-1 dan dijelaskan oleh [sha02]. SHA-1

[1] X.509 memperbolehkan kita untuk memilih algoritma SHA-1 untuk *secure hashing*.

Algoritma	Naskah (bit)	Blok (bit)	Word (bit)	Digest (bit)	Keamanan (bit)
SHA-1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

Tabel 9.1: 4 Varian SHA

menggunakan fungsi f_t, dimana $0 \leq t < 79$, dengan input 3 *word* masing-masing sebesar 32 bit dan menghasilkan output 1 *word*:

$$f_t = \begin{cases} Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \leq t \leq 19 \\ Parity(x,y,z) = x \oplus y \oplus z & 20 \leq t \leq 39 \\ Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \leq t \leq 59 \\ Parity(x,y,z) = x \oplus y \oplus z & 60 \leq t \leq 79. \end{cases}$$

SHA-1 menggunakan konstan k_t sebagai berikut (dengan nilai dalam hexadecimal):

$$k_t = \begin{cases} 5a827999 & 0 \leq t \leq 19 \\ 6ed9eba1 & 20 \leq t \leq 39 \\ 8f1bbcdc & 40 \leq t \leq 59 \\ ca62c1d6 & 60 \leq t \leq 79. \end{cases}$$

Seperti halnya dengan MD5, SHA-1 terdiri dari dua tahap yaitu *preprocessing* dan *hashing*. *Preprocessing* dimulai dengan *padding* yang prosesnya persis sama dengan MD5 (lihat bagian 9.1), yaitu setelah ahir naskah, 1 bit dengan nilai 1 ditambahkan, disusul oleh bit dengan nilai 0 sebanyak 0 sampai dengan 511 tergantung panjang naskah, dan diahiri dengan 64 bit yang merepresentasikan besar naskah asli. Setelah *padding*, naskah terdiri dari n *word* $M[0 \ldots n-1]$ dimana n adalah kelipatan 16. Langkah berikutnya dalam *preprocessing* adalah menyiapkan SHA-1 *buffer* sebesar 5 word:

$$(H_0^{(0)}, H_1^{(0)}, H_2^{(0)}, H_3^{(0)}, H_4^{(0)}).$$

Buffer diberi nilai awal sebagai berikut (nilai dalam hexadecimal):

$$\begin{aligned} H_0^{(0)} &\leftarrow 67452301 \\ H_1^{(0)} &\leftarrow efcdab89 \\ H_2^{(0)} &\leftarrow 98badcfe \\ H_3^{(0)} &\leftarrow 10325476 \\ H_4^{(0)} &\leftarrow c3d2e1f0. \end{aligned}$$

Setelah *buffer* diberi nilai awal, tahap kedua yaitu *hashing* dilakukan terhadap setiap blok $(M^{(1)}, M^{(2)}, \ldots, M^{(n)})$ sebagai berikut $(i = 1, 2, \ldots, n)$:

1. Siapkan *message schedule* $\{W_t\}$:

$$W_t \leftarrow \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1 & 16 \le t \le 79. \end{cases}$$

2. Berikan nilai awal untuk variable a, b, c, d, dan e:

$$\begin{aligned} a &\leftarrow H_0^{(i-1)} \\ b &\leftarrow H_1^{(i-1)} \\ c &\leftarrow H_2^{(i-1)} \\ d &\leftarrow H_3^{(i-1)} \\ e &\leftarrow H_4^{(i-1)}. \end{aligned}$$

3. Untuk $t = 0, 1, 2, \ldots, 79$:

$$\begin{aligned} T &\leftarrow (a <<< 5) + f_t(b, c, d) + e + K_t + W_t \pmod{2^{32}} \\ e &\leftarrow d \\ d &\leftarrow c \\ c &\leftarrow b <<< 30 \\ b &\leftarrow a \\ a &\leftarrow T. \end{aligned}$$

4. Lakukan kalkulasi *hash value* tahap i:

$$\begin{aligned} H_0^i &\leftarrow a + H_0^i \\ H_1^i &\leftarrow b + H_1^i \\ H_2^i &\leftarrow c + H_2^i \\ H_3^i &\leftarrow d + H_3^i \\ H_4^i &\leftarrow e + H_4^i. \end{aligned}$$

Setelah *hashing* dilakukan pada semua blok $(M^{(1)}, M^{(2)}, \ldots, M^{(n)})$, kita dapatkan *digest* sebagai berikut:

$$(H_0^{(n)}, H_1^{(n)}, H_2^{(n)}, H_3^{(n)}, H_4^{(n)}).$$

Beberapa ahli kriptografi telah mencoba mencari kelemahan SHA-1. Xiaoyun Wang dan koleganya berhasil memperkecil ruang pencarian untuk *collision*

SHA-1 dari 2^{80} operasi SHA-1 (yang merupakan "kekuatan" teoritis SHA-1 jika SHA-1 tidak memiliki kelemahan, berdasarkan *birthday attack*) menjadi 2^{69} operasi (lihat [wyy05]). Walaupun demikian, penggunaan SHA-1 masih dianggap cukup aman, dan jika ingin lebih aman lagi, maka SHA-256, SHA-384 atau SHA-512 dapat digunakan.

SHA-256, SHA-384 dan SHA-512, bersama dengan SHA-224 secara kolektif masuk dalam standard SHA-2. Saat bab ini ditulis, NIST sedang mengadakan sayembara pembuatan standard SHA-3 yang diharapkan akan lebih tangguh dari SHA-2.

9.3 Hash Message Authentication Code

Hash message authentication code atau HMAC adalah metode *authentication* untuk pesan atau naskah menggunakan *secure hashing* dan kunci rahasia. Jika k adalah kunci rahasia dan m adalah pesan atau naskah, maka rumus yang digunakan untuk HMAC adalah

$$HMAC(k, m) = H((k \oplus p_o) \circ H((k \oplus p_i) \circ m)).$$

dimana H adalah fungsi *secure hashing* (contohnya MD5 atau SHA-1), $\circ$ adalah operasi penyambungan (*concatenation*), p_o dan p_i masing-masing merupakan *padding* sebesar blok yang digunakan H. *Padding* p_o berisi byte $0x5c$ (hexadecimal) yang diulang untuk memenuhi blok, dan *padding* p_i berisi byte $0x36$ (hexadecimal) yang diulang untuk memenuhi blok. Jika kunci k lebih kecil dari blok, maka k di*padding* dengan 0 sampai memenuhi blok. Jika k lebih besar dari blok, maka k dipotong belakangnya hingga besarnya sama dengan blok. Metode HMAC digunakan karena metode yang menggunakan rumus berikut:

$$MAC(k, m) = H(k \circ m)$$

mempunyai kelemahan yaitu kelemahan terhadap *length extension attack*. Tergantung dari fungsi H, seseorang dapat menambahkan sesuatu (misalnya m_a) ke m:

$$m' = m \circ m_a$$

dan tanpa mengetahui k dapat membuat

$$MAC(k, m').$$

HMAC digunakan oleh SSL/TLS (lihat bagian 20.1) dan IPsec (lihat bagian 20.3). Metode HMAC menggunakan MD5 dinamakan HMAC-MD5. Demikian juga, metode HMAC menggunakan SHA-1 dinamakan HMAC-SHA-1.

9.4 Ringkasan

Bab ini telah membahas *secure hashing* yaitu proses pembuatan *fingerprint* atau *digest* untuk suatu naskah. Dua contoh algoritma *secure hashing* dijelaskan: MD5 dan SHA-1. MD5 telah dianggap tidak aman penggunaannya untuk *digital signature*. SHA-1, meskipun memiliki kelemahan, masih dianggap cukup aman. Untuk lebih aman lagi, SHA-256, SHA-384 atau SHA-512 dapat digunakan. Metode *authentication* menggunakan kunci, HMAC, juga dijelaskan.

Bab 10

Matematika III - Dasar untuk PKC

Di bab ini kita akan bahas berbagai topik matematika yang merupakan dasar dari kriptografi *public key* (PKC).

10.1 Fermat's Little Theorem

Teorema kecil Fermat (*Fermat's little theorem*) adalah teorema sangat penting dalam teori bilangan yang menjadi dasar dari berbagai teknik enkripsi asimetris.

Teorema 30 (Fermat's Little Theorem) *Untuk bilangan prima p dan bilangan bulat a, $a^p \equiv a \pmod{p}$ dan jika a tidak dapat dibagi oleh p, maka $a^{p-1} \equiv 1 \pmod{p}$.*

Untuk membuktikan teorema ini, pertama kita tunjukkan bahwa jika $p \not| a$, maka

$$\{0a, 1a, 2a, 3a, \ldots, (p-1)a\}$$

merupakan himpunan lengkap dari *residue classes modulo p*. Dengan kata lain, setiap elemen merupakan representasi dari satu kelas yang unik (elemen yang berbeda merepresentasikan kelas yang berbeda), dan setiap kelas mempunyai representasi dalam himpunan. Untuk itu, kita ambil $0 \leq i < p$ dan $0 \leq j < p$ dengan $i \neq j$. Jika ia dan ja berada dalam kelas yang sama, maka

$$ia \equiv ja \pmod{p},$$

yang berarti $p|(i-j)a$, dan karena $p \not| a$ maka $p|(i-j)$. Karena $i < p$ dan $j < p$, maka ini hanya bisa terjadi jika $i = j$, suatu kontradiksi karena $i \neq j$.

Jadi setiap elemen merepresentasikan kelas yang unik, dan karena ada p kelas yang berbeda, maka semua kelas ada dalam himpunan, jadi himpunan adalah himpunan lengkap dari *residue classes modulo p*. Selanjutnya, jelas bahwa $0a = 0$, jadi $1, 2, 3, \ldots, p-1$ dan $1a, 2a, 3a, \ldots (p-1)a$ hanya berbeda urutan jika setiap bilangan dianggap modulo p, dan produk dari deretan menjadi ekuivalen modulo p:

$$a^{p-1}(p-1)! \equiv (p-1)! \pmod{p},$$

jadi $p|((p-1)!(a^{p-1}-1))$. Karena p tidak membagi $(p-1)!$, maka $p|a^{p-1}-1$, yang berarti

$$a^{p-1} \equiv 1 \pmod{p}$$

untuk $p \not| a$. Jika kita kalikan kedua sisi dengan a kita dapatkan

$$a^p \equiv a \pmod{p}$$

untuk $p \not| a$. Untuk $p|a$ pembuktian

$$a^p \equiv a \pmod{p}$$

sangat mudah karena $a^p \equiv 0 \equiv a \pmod{p}$. Lengkaplah pembuktian teorema 30 (*Fermat's Little Theorem*).

Teorema 30 dapat digunakan untuk mempermudah kalkulasi pemangkatan modulo bilangan prima. Sebagai contoh, kita coba kalkulasi $2^{58} \pmod{19}$. Karena 19 adalah bilangan prima dan 2 tidak dapat dibagi 19, maka teorema 30 dapat digunakan untuk mengkalkulasi

$$\begin{aligned} 2^{18} &\equiv 2^{19-1} \pmod{19} \\ &\equiv 1 \pmod{19}. \end{aligned}$$

Jadi

$$2^{58} = (2^{18})^3 \times 2^4 \equiv 1^3 \times 2^4 \equiv 16 \pmod{19}.$$

Meskipun dapat digunakan untuk mempermudah kalkulasi, dalam kriptografi, peran terpenting dari *Fermat's little theorem* adalah sebagai dasar dari berbagai teknik enkripsi asimetris.

10.2 Chinese Remainder Theorem

Seperti halnya dengan algoritma Euclid, *Chinese Remainder Theorem* merupakan penemuan penting dibidang teori bilangan yang telah berumur ribuan tahun. Aplikasi jaman dahulu mungkin untuk astronomi dimana r *events* berulang secara periodis, setiap *event* i mempunyai periode m_i, *event* i akan muncul secepatnya dalam waktu a_i, dan kita ingin mengetahui kapan semua *events* akan muncul secara bersamaan. Suatu contoh aplikasi ini adalah untuk memprediksi kapan akan terjadi gerhana.

Teorema 31 (Chinese Remainder Theorem) *Jika kita mempunyai beberapa persamaan dengan modulus berbeda sebagai berikut*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1}, \\ x &\equiv a_2 \pmod{m_2}, \\ &\dots \\ x &\equiv a_r \pmod{m_r} \end{aligned}$$

dimana setiap pasangan modulus adalah koprima ($\gcd(m_i, m_j) = 1$ *untuk* $i \neq j$)*, maka terdapat solusi untuk* x*. Jika* x_1 *dan* x_2 *merupakan solusi untuk* x*, maka* $x_1 \equiv x_2 \pmod{M}$ *dimana* $M = m_1 m_2 \dots m_r$*.*

Pembuktian bahwa sistem persamaan seperti diatas mempunyai solusi untuk x bersifat konstruktif, jadi menghasilkan algoritma untuk mencari solusi. Kita definisikan $M_i = M/m_i$, jadi M_i merupakan produk dari semua modulus kecuali m_i. Karena $\gcd(m_i, M_i) = 1$, maka terdapat bilangan bulat N_i (*inverse* yang dapat dicari menggunakan *extended Euclidean algorithm*) dimana $M_i N_i \equiv 1 \pmod{m_i}$. Maka suatu solusi untuk x adalah

$$x = \sum_{j=1}^{r} a_j M_j N_j.$$

Untuk setiap i, karena semua suku kecuali suku i dapat dibagi dengan m_i, maka hanya suku i yang tidak $\equiv 0 \pmod{m_i}$, jadi

$$x \equiv a_i M_i N_i \equiv a_i \pmod{m_i}$$

seperti yang dikehendaki. Untuk menunjukkan bahwa solusi x unik modulo M, kita tunjukkan bahwa jika x_1 dan x_2 adalah solusi untuk x, maka $x_1 \equiv x_2 \pmod{M}$. Untuk setiap i, $x_1 \equiv x_2 \equiv a_i \pmod{m_i}$, atau $x_1 - x_2 \equiv 0 \pmod{m_i}$. Jadi $x_1 - x_2 \equiv 0 \pmod{M}$, yang berarti $x_1 \equiv x_2 \pmod{M}$.

Sebagai contoh kalkulasi dengan angka menggunakan *Chinese Remainder Theorem*, kita gunakan

$$x \equiv 2 \pmod{3}, \quad x \equiv 3 \pmod{5}, \quad x \equiv 2 \pmod{7}.$$

Setelah kita periksa bahwa

$$\gcd(3,5) = 1,\ \gcd(3,7) = 1,\ \gcd(5,7) = 1,$$

kita lanjutkan kalkulasi dan dapatkan

$$M = 105,\ M_1 = 35,\ M_2 = 21,\ M_3 = 15.$$

Kita dapatkan masing-masing *inverse* (bisa menggunakan *extended Euclidean algorithm*):

$$N_1 \equiv 2 \pmod 3, \quad N_2 \equiv 1 \pmod 5, \quad N_3 \equiv 1 \pmod 7.$$

Solusinya:

$$x = 2 \cdot 35 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1 = 140 + 63 + 30 = 233 \equiv 23 \pmod{105},$$

atau

$$x = 23 + 105n$$

dimana n adalah bilangan bulat apa saja.

10.3 Fungsi Euler

Fermat's little theorem (teorema 30) berlaku untuk modulus prima. Euler berhasil membuat generalisasi teorema 30 dengan menggunakan fungsi Euler phi (ϕ). Definisi fungsi ϕ, untuk $n > 0$, adalah sebagai berikut:

Definisi 17 (Fungsi Euler)

$$\phi(n) = \sharp\{0 \leq b < n | \gcd(b, n) = 1\}.$$

Dengan kata lain hasil fungsi adalah banyaknya bilangan bulat non-negatif dan lebih kecil dari n yang koprima dengan n. Jadi $\phi(1) = 1$ dan untuk bilangan prima p, $\phi(p) = p - 1$ karena $1, 2, 3, \ldots, p - 1$ semua koprima terhadap p, sedangkan $\gcd(p, 0) = p \neq 1$. Untuk bilangan prima p,

$$\phi(p^\alpha) = p^\alpha - p^{\alpha-1}.$$

Untuk membuktikan ini, perhatikan bahwa bilangan antara 0 dan p^α yang tidak koprima dengan p^α adalah yang dapat dibagi dengan p, yang banyaknya adalah $p^{\alpha-1}$, jadi $p^{\alpha-1}$ harus dikurangkan dari p^α.

Dengan menggunakan konsep fungsi Euler ϕ, *Fermat's little theorem* dapat digeneralisasi sebagai berikut:

Teorema 32

$$a^{\phi(m)} \equiv 1 \pmod m$$

jika $\gcd(a, m) = 1$.

Untuk $m = p$ bilangan prima, $\phi(p) = p - 1$, dan $\gcd(a, p) = 1$ berarti a tidak dapat dibagi oleh p, jadi kita dapatkan *Fermat's little theorem* dalam bentuk orisinil. Jadi teorema jelas berlaku jika m adalah bilangan prima. Untuk m

bukan bilangan prima, pembuktiannya menggunakan sifat *multiplicative* fungsi ϕ:

$$\phi(mn) = \phi(m)\phi(n)$$

jika $\gcd(m, n) = 1$. Untuk membuktikan sifat *multiplicative*, kita harus hitung semua bilangan bulat antara 0 dan $mn - 1$ yang koprima dengan mn (jadi tidak ada faktor bilangan yang lebih besar dari 1 yang juga merupakan faktor mn). Kita beri label j untuk bilangan yang kita hitung. Kita beri label j_1 untuk *residue* non-negatif terkecil j modulo m dan j_2 untuk *residue* non-negatif terkecil j modulo n, jadi $0 \leq j_1 < m$, $0 \leq j_2 < n$,

$$\begin{aligned} j &\equiv j_1 \pmod{m}, \\ j &\equiv j_2 \pmod{n}. \end{aligned}$$

Berdasarkan teorema 31 (*Chinese Remainder Theorem*), untuk setiap pasangan j_1, j_2, hanya ada satu j antara 0 dan $mn - 1$ yang mengakibatkan kedua persamaan diatas berlaku. Juga perhatikan bahwa j koprima dengan mn jika dan hanya jika j koprima dengan m dan n. Jadi banyaknya j yang harus dihitung sama dengan banyaknya kombinasi pasangan j_1, j_2. Banyaknya j_1 yang koprima dengan m dimana $0 \leq j_1 < m$ adalah $\phi(m)$, sedangkan banyaknya j_2 yang koprima dengan n dimana $0 \leq j_2 < n$ adalah $\phi(n)$. Jadi banyaknya j adalah $\phi(m)\phi(n)$. Selesailah pembuktian sifat *multiplicative* ϕ. Kembali ke teorema 32, kita sudah buktikan untuk m prima. Kita ingin buktikan juga untuk m berupa bilangan prima p dipangkatkan ($m = p^\alpha$). Pembuktian kita lakukan dengan cara induksi. Untuk $\alpha = 1$, kita dapatkan bentuk asli teorema 30, jadi sudah terbukti. Tinggal kita buktikan bahwa jika teorema 32 berlaku untuk $\alpha - 1$, maka ia juga berlaku untuk α (dengan $\alpha \geq 2$). Untuk $\alpha - 1$ kita dapatkan

$$\begin{aligned} a^{\phi(p^{\alpha-1})} &\equiv 1 \pmod{p^{\alpha-1}}, \text{ jadi} \\ a^{p^{\alpha-1}-p^{\alpha-2}} &\equiv 1 \pmod{p^{\alpha-1}}, \text{ yang berarti} \\ a^{p^{\alpha-1}-p^{\alpha-2}} &= 1 + p^{\alpha-1}b \text{ untuk suatu } b. \end{aligned}$$

Kita pangkatkan kedua sisi persamaan dengan p. Untuk sisi kiri persamaan kita dapatkan $a^{p^\alpha - p^{\alpha-1}}$ atau $a^{\phi(p^\alpha)}$. Untuk sisi kanan, $(1+p^{\alpha-1}b)^p$ mempunyai koefisien binomial yang dapat dibagi oleh p kecuali untuk 1 dan $(p^{\alpha-1}b)^p$. Jadi semua suku kecuali 1 dapat dibagi dengan p^α, dan sisi kanan menjadi $1 + bp^\alpha$ untuk suatu b. Persamaan menjadi

$$\begin{aligned} a^{\phi(p^\alpha)} &= 1 + bp^\alpha, \text{ yang berarti} \\ a^{\phi(p^\alpha)} &\equiv 1 \pmod{p^\alpha}. \end{aligned}$$

Selesailah pembuktian untuk $m = p^\alpha$. Selanjutnya, berdasarkan *fundamental theorem of arithmetic*, setiap bilangan dapat diuraikan menjadi

$$m = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$$

dimana untuk $1 \leq i \leq n$ setiap p_i merupakan bilangan prima unik. Dengan memangkatkan kedua sisi dari

$$a^{\phi(p_i^{\alpha_i})} \equiv 1 \pmod{p_i^{\alpha_i}}$$

dengan pangkat yang sesuai, kita dapatkan

$$a^{\phi(m)} \equiv 1 \pmod{p_i^{\alpha_i}}$$

untuk $1 \leq i \leq n$. Karena untuk $i \neq j$, $p_i^{\alpha_i}$ koprima dengan $p_j^{\alpha_j}$, maka kita dapatkan

$$a^{\phi(m)} \equiv 1 \pmod{m}.$$

Selesailah pembuktian teorema 32. Satu lagi teorema mengenai fungsi ϕ Euler adalah sebagai berikut:

Teorema 33

$$\sum_{d|n} \phi(d) = n.$$

Untuk membuktikan teorema 33, kita beri notasi $f(n) = \sum_{d|n} \phi(d)$, jadi $f(n)$ merupakan penjumlahan $\phi(d)$ untuk semua d yang membagi n. Kita harus tunjukkan bahwa $f(n) = n$, tetapi pertama kita tunjukkan lebih dahulu bahwa $f(n)$ bersifat *multiplicative*, yaitu $f(mn) = f(m)f(n)$ jika $\gcd(m, n) = 1$. Kita mengetahui bahwa pembagi $d|mn$ dapat diuraikan menjadi $d_1 \cdot d_2$ jika $\gcd(m, n) = 1$, dimana $d_1|m$ dan $d_2|n$. Karena $\gcd(d_1, d_2) = 1$, kita dapatkan $\phi(d) = \phi(d_1)\phi(d_2)$. Untuk mencari semua $d|mn$, kita harus mencari semua kombinasi d_1, d_2 dimana $d_1|m$ dan $d_2|n$. Jadi

$$\begin{aligned} f(mn) &= \sum_{d_1|m} \sum_{d_2|n} \phi(d_1)\phi(d_2) \\ &= (\sum_{d_1|m} \phi(d_1))(\sum_{d_2|n} \phi(d_2)) \\ &= f(m)f(n). \end{aligned}$$

Kembali ke pembuktian teorema 33, berdasarkan *fundamental theorem of arithmetic*, setiap n dapat diuraikan dalam bentuk $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_r^{\alpha_r}$, dimana setiap p_i adalah bilangan prima yang unik. Karena f bersifat *multiplicative*, $f(n) = f(p_1^{\alpha_1}) f(p_2^{\alpha_2}) \cdots f(p_r^{\alpha_r})$. Jadi kita cukup menunjukkan bahwa $f(p^\alpha) = p^\alpha$. Karena p merupakan bilangan prima, pembagi dari p^α adalah p^j untuk $0 \leq j \leq \alpha$. Jadi

$$f(p^\alpha) = \sum_{j=0}^{\alpha} \phi(p^j)$$

$$\begin{aligned} &= 1 + \sum_{j=1}^{\alpha} p^j - p^{j-1} \\ &= p^\alpha. \end{aligned}$$

Selesailah pembuktian teorema 33.

10.4 Group of Units

Kerap untuk suatu *finite ring* $\mathbf{Z}/n\mathbf{Z}$ (yang merupakan *finite field* jika n prima), kita ingin fokus pada elemen-elemen yang mempunyai *inverse* (elemen-elemen yang merupakan *unit*). Ternyata elemen-elemen tersebut membentuk suatu *multiplicative group* $(\mathbf{Z}/n\mathbf{Z})^*$ karena

- untuk *unit* $[a]$ dan $[b]$, $[a][b] = [ab]$ merupakan *unit* karena jika $[a]$ mempunyai *inverse* $[a]^{-1} = [u]$ dan $[b]$ mempunyai *inverse* $[b]^{-1} = [v]$ maka $[ab][uv] = [au][bv] = 1$, jadi $[ab]$ mempunyai *inverse* $[uv]$ (*closure*);
- untuk *unit* $[a]$, $[b]$ dan $[c]$, $([a][b])[c] = [a]([b][c])$ (*associativity*);
- terdapat elemen *identity* $[1]$ (*identity*); dan
- setiap elemen mempuyai *inverse*.

Patut diperhatikan bahwa untuk bilangan prima p dan bilangan $n > 1$, $\mathbf{Z}/p^n\mathbf{Z}$ tidak sama dengan $\mathbf{GF}(p^n)$: $\mathbf{Z}/p^n\mathbf{Z}$ merupakan *finite ring* tetapi bukan *finite field*, sedangkan $\mathbf{GF}(p^n)$ merupakan *finite field*. $\mathbf{GF}(p^n)^*$ akan dibahas di bagian 10.7.

Untuk setiap elemen $[a] \in (\mathbf{Z}/n\mathbf{Z})^*$, kita ketahui bahwa $\gcd(a, n) = 1$, jadi banyaknya elemen dalam $(\mathbf{Z}/n\mathbf{Z})^*$ adalah $\phi(n)$. Suatu *multiplicative group* $(\mathbf{Z}/n\mathbf{Z})^*$ disebut *cyclic* jika mempunyai elemen a dengan *order* (pangkat positif terkecil dari a yang menghasilkan 1) $\phi(n)$, dan elemen a disebut *generator* karena setiap elemen dalam *group* merupakan pemangkatan dari a.

Teorema 34 *Jika a adalah elemen dari $(\mathbf{Z}/n\mathbf{Z})^*$ untuk suatu bilangan n, maka order dari a (yang kita beri label d) membagi $\phi(n)$.*

Pembuktian teorema 34 menggunakan teorema 32 yang mengatakan bahwa $a^{\phi(n)} = 1$ (dalam $(\mathbf{Z}/n\mathbf{Z})$). Jika d tidak membagi $\phi(n)$, maka terdapat $0 < r < d$ dimana

$$bd + r = \phi(n)$$

untuk suatu b, dan

$$a^r = a^{\phi(n)-bd} \equiv 1 \pmod{n}.$$

Ini adalah suatu kontradiksi karena d merupakan pangkat positif a terkecil yang menghasilkan 1. Jadi d membagi $\phi(n)$, dan selesailah pembuktian teorema 34.

Karena setiap elemen a dalam $(\mathbf{Z}/n\mathbf{Z})^*$ mempunyai *order* (sebut saja d), a menjadi *generator* untuk suatu *cyclic subgroup*:

$$G_a = \{a^1, a^2, \ldots, a^d = a^0\}$$

karena kita dapatkan:

- jika $x, y \in G_a$ (jadi terdapat $1 \leq i \leq d$ dan $1 \leq j \leq d$ dimana $x = a^i, y = a^j$), maka $xy \in G_a$ (karena $xy = a^{ij} = a^{ij \bmod d}$), jadi kita dapatkan *closure*;
- jika $x, y, z \in G_a$ (jadi terdapat $1 \leq i \leq d$, $1 \leq j \leq d$ dan $1 \leq k \leq d$ dimana $x = a^i, y = a^j, z = a^k$), maka $(xy)z = x(yz)$ (karena $(xy)z = (a^i a^j)a^k = a^i(a^j a^k) = x(yz)$), jadi kita dapatkan *associativity*;
- kita dapatkan $a^d = a^0 = 1 \in G_a$, jadi G_a memiliki *identity*; dan
- untuk a^i dengan $1 \leq i \leq d$, kita dapatkan a^{d-i} yang juga merupakan elemen dari G_a dengan $a^i a^{d-i} = a^d = 1$, jadi setiap elemen dalam G_a mempunyai *inverse* dalam G_a.

Tentunya *order* dari *generator* untuk suatu *cyclic group* sama dengan banyaknya elemen dalam *group* tersebut. Istilah *order* juga kerap digunakan untuk banyaknya elemen dalam *group*. Untuk *cyclic group*, *order* dari *group* sama dengan *order* dari *generator*.

Kita mulai pembahasan struktur $(\mathbf{Z}/n\mathbf{Z})^*$ dengan membahas struktur dari $(\mathbf{Z}/2^e\mathbf{Z})^*$.

Teorema 35 *Multiplicative group* $(\mathbf{Z}/2^e\mathbf{Z})^*$ *cyclic hanya untuk* $e = 1, 2$.

Untuk $e = 1$ kita dapatkan $(\mathbf{Z}/2^1\mathbf{Z})^* = \{1\}$ *cyclic* karena elemen 1 mempunyai *order* $\phi(2) = 1$. Untuk $e = 2$ kita dapatkan $(\mathbf{Z}/2^2\mathbf{Z})^* = \{1, 3\}$ *cyclic* karena elemen 3 mempunyai *order* $\phi(4) = 2$. Untuk $e > 2$ kita harus tunjukkan bahwa $(\mathbf{Z}/2^e\mathbf{Z})^*$ tidak mempunyai elemen dengan *order* $\phi(2^e) = 2^e - 2^{e-1} = 2^{e-1}$. Ini kita lakukan dengan menunjukkan bahwa

$$a^{2^{e-2}} \equiv 1 \pmod{2^e} \tag{10.1}$$

untuk setiap bilangan ganjil a (untuk a genap, $\gcd(a, 2^e) > 1$, jadi bukan elemen *group*). Kita buktikan ini dengan cara induksi. Untuk *base case* $e = 3$ kita harus buktikan

$$a^2 \equiv 1 \pmod{8}$$

untuk setiap bilangan ganjil a. Karena terdapat bilangan b dimana $a = 2b + 1$, kita dapatkan

$$a^2 = (2b+1)^2 = 4b^2 + 4b + 1 = 4b(b+1) + 1 \equiv 1 \pmod{8}.$$

Jika persamaan 10.1 berlaku untuk suatu $e \geq 3$, maka untuk setiap bilangan ganjil a kita dapatkan

$$a^{2^{e-2}} = 1 + 2^e k$$

untuk suatu bilangan k. Kita kuadratkan:

$$\begin{aligned} a^{2^{(e+1)-2}} &= (1 + 2^e k)^2 \\ &= 1 + 2^{e+1} k + 2^{2e} k^2 \\ &= 1 + 2^{e+1}(k + 2^{2e-1} k^2) \\ &\equiv 1 \pmod{2^{e+1}} \end{aligned}$$

yang berarti persamaan 10.1 berlaku untuk $e+1$. Selesailah pembuktian induksi untuk persamaan 10.1. Walaupun $(\mathbf{Z}/2^e\mathbf{Z})^*$ untuk $e \geq 3$ bukan merupakan *cyclic group*, kita akan tunjukkan bahwa $(\mathbf{Z}/2^e\mathbf{Z})^*$ merupakan produk dari dua *cyclic group*.

Teorema 36 *Untuk $e \geq 3$, $(\mathbf{Z}/2^e\mathbf{Z})^*$ merupakan produk dari dua cyclic group dengan generator 5 dan -1.*

Dengan menggunakan persamaan 10.1 kita dapatkan untuk $e \geq 3$:

$$5^{2^{e-2}} \equiv 1 \pmod{2^e}. \tag{10.2}$$

Kita ingin juga tunjukkan bahwa untuk $e \geq 3$:

$$5^{2^{e-3}} \equiv 1 + 2^{e-1} \pmod{2^e}. \tag{10.3}$$

Ini kita lakukan dengan induksi. Untuk $e = 3$ kita dapatkan

$$\begin{aligned} 5^{2^{e-3}} &= 5^{2^{3-3}} \\ &= 5^{2^0} \\ &= 5 \\ &= 1 + 2^{3-1} \\ &= 1 + 2^{e-1}. \end{aligned}$$

Untuk langkah induksi kita umpamakan

$$5^{2^{e-3}} \equiv 1 + 2^{e-1} \pmod{2^e}$$

jadi

$$5^{2^{e-3}} = 1 + n2^{e-1}$$

untuk suatu bilangan ganjil n (karena jika n genap maka $5^{2^{e-3}} = 1 + n2^{e-1}$ berarti $5^{2^{e-3}} \equiv 1 \pmod{2^e}$, bukan $5^{2^{e-3}} \equiv 1 + 2^{e-1} \pmod{2^e}$). Kita harus tunjukkan bahwa

$$5^{2^{e-2}} \equiv 1 + 2^e \pmod{2^{e+1}}.$$

Kita dapatkan

$$\begin{aligned} 5^{2^{e-2}} &= (5^{2^{e-3}})^2 \\ &= (1+n2^{e-1})^2 \\ &= 1+2n2^{e-1}+n^2 2^{2e-2} \\ &= 1+n2^e+n^2 2^{2e-2} \\ &\equiv 1+2^e \pmod{2^{e+1}} \end{aligned}$$

karena 2^{2e-2} dapat dibagi oleh 2^{e+1} untuk $e \geq 3$ dan $n2^e$ menyisakan 2^e jika dibagi oleh 2^{e+1} (karena n ganjil). Selesailah pembuktian persamaan 10.3 dengan induksi. Dari persamaan 10.2 dan 10.3 kita dapatkan 2^{e-2} sebagai *order* dari 5, jadi 5 adalah *generator* untuk *cyclic subgroup* dengan 2^{e-2} elemen:

$$\{5 \pmod{2^e}, 5^2 \pmod{2^e}, \ldots, 5^{2^{e-2}} \equiv 1 \pmod{2^e}\}.$$

Meneruskan pembuktian teorema 36, -1 merupakan *generator* untuk *cyclic subgroup* $\{-1, 1\}$. Kita ingin tunjukkan bahwa produk dua *cyclic group*, berupa himpunan semua elemen berbentuk $5^i(-1)^j$ dengan $0 \leq i < 2^{e-2}$ dan $0 \leq j < 2$, menghasilkan *group* $(\mathbf{Z}/2^e\mathbf{Z})^*$ yang merupakan himpunan bilangan ganjil $\{1, 3, 5, 2^e - 1\}$. Karena $5 \equiv 1 \pmod 4$, maka $\{5^i \pmod{2^e} | 0 \leq i < 2^{e-2}\}$ membentuk setengah dari $(\mathbf{Z}/2^e\mathbf{Z})^*$, sedangkan sisanya setengah lagi dibentuk oleh $\{-(5^i) \pmod{2^e} | 0 \leq i < 2^{e-2}\}$, jadi

$$(\mathbf{Z}/2^e\mathbf{Z})^* = \{5^i(-1)^j \pmod{2^e} | 0 \leq i < 2^{e-2}, 0 \leq j < 2\}.$$

Selesailah pembuktian teorema 36.

Teorema 37 *Jika p adalah bilangan prima, maka terdapat $\phi(d)$ elemen dengan order d dalam $(\mathbf{Z}/p\mathbf{Z})^*$, untuk setiap d yang membagi $p-1$.*

Untuk setiap $d|p-1$ kita definisikan

$$\Omega_d = \{a \in (\mathbf{Z}/p\mathbf{Z})^* | a \text{ mempunyai } \textit{order } d\} \text{ dan } \omega(d) = |\Omega_d|$$

jadi $\omega(d)$ adalah banyaknya elemen yang mempunyai *order* d dalam $(\mathbf{Z}/p\mathbf{Z})^*$. Jadi kita harus tunjukkan bahwa

$$\omega(d) = \phi(d)$$

untuk setiap $d|p-1$. Teorema 34 mengatakan bahwa setiap elemen mempunyai *order* yang membagi $\phi(p) = p-1$, jadi himpunan-himpunan Ω_d mempartisi $(\mathbf{Z}/p\mathbf{Z})^*$, oleh karena itu

$$\sum_{d|p-1} \omega(d) = p-1.$$

Teorema 33 dengan $p-1$ menggantikan n menghasilkan

$$\sum_{d|p-1} \phi(d) = p-1,$$

jadi

$$\sum_{d|p-1} (\phi(d) - \omega(d)) = 0.$$

Jika kita dapat tunjukkan bahwa $\omega(d) \leq \phi(d)$ untuk setiap $d|p-1$, karena total penjumlahan adalah 0, maka $\omega(d) = \phi(d)$. Untuk Ω_d yang kosong jelas $\omega(d) \leq \phi(d)$, jadi kita dapat mengumpamakan terdapat elemen $a \in \Omega_d$. Berdasarkan definisi Ω_d, setiap hasil pemangkatan

$$a^1, a^2, \ldots a^d$$

berbeda, dan

$$(a^i)^d = 1$$

untuk $i = 1, 2, \ldots, d$. jadi setiap a^i merupakan akar dari *polynomial* $f(x) = x^d - 1$ dalam $(\mathbf{Z}/p\mathbf{Z})$. Karena banyaknya akar untuk $f(x)$ maksimum d, maka $a^1, a^2, \ldots a^d$ membentuk himpunan akar yang komplit. Kita ingin tunjukkan bahwa Ω_d terdiri dari akar-akar a^i dimana $\gcd(i,d) = 1$. Jika $b \in \Omega_d$, maka $b = a^i$ untuk suatu $1 \leq i \leq d$. Dengan $j = \gcd(i,d)$,

$$b^{d/j} = a^{id/j} = (a^d)^{i/j} = 1^{i/j} = 1$$

dalam $(\mathbf{Z}/p\mathbf{Z})$. Tetapi b mempunyai *order* d, jadi tidak ada pangkat dari b yang lebih kecil dari d yang menghasilkan 1, jadi $j = 1$. Jadi setiap elemen $b \in \Omega_d$ mempunyai bentuk a^i dengan $1 \leq i \leq d$ dan $\gcd(i,d) = 1$, jadi terdapat $\phi(d)$ elemen dalam Ω_d. Jadi kita sudah tunjukkan bahwa $\omega(d) = \phi(d)$ dan selesailah pembuktian teorema 37. Sebagai konsekuensi dari teorema 37, kita dapatkan $(\mathbf{Z}/p\mathbf{Z})^*$ merupakan *cyclic group* karena terdapat $\phi(p-1) = \phi(\phi(p))$ elemen dengan *order* $p-1 = \phi(p)$.

Teorema 38 *Untuk bilangan prima p yang ganjil dan $e \geq 1$, $(\mathbf{Z}/p^e\mathbf{Z})^*$ merupakan cyclic group.*

Kita sudah tunjukkan ini untuk $e = 1$. Untuk $e \geq 2$ kita akan buktikan teorema 38 dengan membuktikan bahwa terdapat *generator* untuk $(\mathbf{Z}/p^e\mathbf{Z})^*$. Kita mulai dengan *generator* untuk $(\mathbf{Z}/p^2\mathbf{Z})^*$. Karena $(\mathbf{Z}/p\mathbf{Z})^*$ *cyclic* maka terdapat *generator* g untuk $(\mathbf{Z}/p\mathbf{Z})^*$ dimana

$$g^{p-1} \equiv 1 \pmod{p}$$

tetapi

$$g^i \not\equiv 1 \pmod{p}$$

untuk $1 \leq i < p-1$. Karena $\gcd(g,p) = 1$, maka $\gcd(g^2,p) = 1$, jadi g juga merupakan elemen (tetapi belum tentu *generator*) untuk $(\mathbf{Z}/p^2\mathbf{Z})^*$. Jika d adalah *order* dari g dalam $(\mathbf{Z}/p^2\mathbf{Z})^*$, maka teorema 34 mengatakan bahwa $d|\phi(p^2)$, jadi d membagi $p(p-1)$. Karena $g^d \equiv 1 \pmod{p^2}$ maka $g^d \equiv 1 \pmod{p}$. Tetapi g mempunyai *order* $p-1$ dalam $(\mathbf{Z}/p\mathbf{Z})^*$, jadi $p-1|d$. Karena $d|p(p-1)$, $p-1|d$ dan p adalah bilangan prima, maka hanya ada dua kemungkinan:

$$\begin{aligned} d &= p-1 \text{ atau} \\ d &= p(p-1). \end{aligned}$$

Jika $d = p(p-1)$ maka g menjadi *generator* untuk $(\mathbf{Z}/p^2\mathbf{Z})^*$ dan kita selesai, jadi kita lanjutkan dengan $d = p-1$. Kita buat $h = g+p$, jadi $h \equiv g \pmod{p}$ yang berarti h adalah *generator* untuk $(\mathbf{Z}/p\mathbf{Z})^*$, jadi seperti halnya dengan g, h mempunyai *order* $p-1$ atau $p(p-1)$ dalam $(\mathbf{Z}/p^2\mathbf{Z})^*$. Karena $g^{p-1} \equiv 1 \pmod{p^2}$, kita dapatkan

$$\begin{aligned} h^{p-1} &= (g+p)^{p-1} \\ &= g^{p-1} + (p-1)g^{p-2}p + \ldots \\ &= g^{p-1} + p^2g^{p-2} - pg^{p-2} + \ldots \\ &\equiv 1 - pg^{p-2} \pmod{p^2} \end{aligned}$$

karena suku-suku yang direpresentasikan oleh ... semua dapat dibagi oleh p^2. Karena $\gcd(g,p) = 1$, maka $pg^{p-2} \not\equiv 0 \pmod{p^2}$, jadi $h^{p-1} \not\equiv 1 \pmod{p^2}$ yang berarti *order* dari h bukan $p-1$. Jadi *order* dari h dalam $(\mathbf{Z}/p^2\mathbf{Z})^*$ adalah $p(p-1)$ yang berarti kita mempunyai *generator* h untuk $(\mathbf{Z}/p^2\mathbf{Z})^*$. Selesailah pembuktian untuk $e = 2$. Untuk $e \geq 2$ kita akan buktikan dengan induksi bahwa *generator* h untuk $(\mathbf{Z}/p^2\mathbf{Z})^*$ juga merupakan *generator* untuk $(\mathbf{Z}/p^e\mathbf{Z})^*$. Apabila h adalah suatu *generator* untuk $(\mathbf{Z}/p^e\mathbf{Z})^*$, karena $\gcd(h,p^e) = 1$, maka $\gcd(h,p^{e+1}) = 1$. Jadi h merupakan elemen (walaupun belum tentu *generator* untuk $(\mathbf{Z}/p^{e+1}\mathbf{Z})^*$. Jika d adalah *order* dari h dalam $(\mathbf{Z}/p^{e+1}\mathbf{Z})^*$, maka teorema 34 mengatakan bahwa $d|\phi(p^{e+1})$, jadi d membagi $p^e(p-1)$. Karena $h^d \equiv 1 \pmod{p^{e+1}}$, maka $h^d \equiv 1 \pmod{p^e}$. Tetapi h mempunyai *order* $p^{e-1}(p-1)$ dalam $(\mathbf{Z}/p^e\mathbf{Z})^*$, jadi $p^{e-1}(p-1)$ membagi d. Karena $d|p^e(p-1)$, $p^{e-1}(p-1)|d$ dan p merupakan bilangan prima, maka hanya ada dua kemungkinan:

$$\begin{aligned} d &= p^e(p-1) \text{ atau} \\ d &= p^{e-1}(p-1). \end{aligned}$$

Jika $d = p^e(p-1)$, maka h merupakan *generator* untuk $(\mathbf{Z}/p^{e+1}\mathbf{Z})^*$. Kita ingin tunjukkan bahwa $d = p^{e-1}(p-1)$ adalah sesuatu yang tidak mungkin, jadi kita ingin tunjukkan bahwa $h^{p^{e-1}(p-1)} \not\equiv 1 \pmod{p^{e+1}}$. Karena h merupakan *generator* untuk $(\mathbf{Z}/p^e\mathbf{Z})^*$, maka h mempunyai *order* $\phi(p^e) = p^{e-1}(p-1)$ dalam

$(\mathbf{Z}/p^e\mathbf{Z})^*$, jadi $h^{p^{e-2}(p-1)} \not\equiv 1 \pmod{p^e}$. Akan tetapi $p^{e-2}(p-1) = \phi(p^{e-1})$, jadi $h^{p^{e-2}(p-1)} \equiv 1 \pmod{p^{e-1}}$. Jadi kita dapatkan $h^{p^{e-2}(p-1)} = 1 + kp^{e-1}$. Kita pangkatkan kedua sisi persamaan dengan p:

$$\begin{aligned} h^{p^{e-1}(p-1)} &= (1 + kp^{e-1})^p \\ &= 1 + pkp^{e-1} + \frac{p(p-1)}{2}(kp^{e-1})^2 + \dots \\ &= 1 + kp^e + \frac{1}{2}k^2p^{2e-1}(p-1) + \dots \end{aligned}$$

dimana suku-suku yang direpresentasikan oleh $\dots$ dapat dibagi oleh $(p^{e-1})^3$, jadi dapat dibagi oleh p^{e+1} karena $3(e-1) \geq e+1$ untuk $e \geq 2$, jadi

$$h^{p^{e-1}(p-1)} \equiv 1 + kp^e + \frac{1}{2}k^2p^{2e-1}(p-1) \pmod{p^{e+1}}.$$

Karena p ganjil, maka $\frac{1}{2}k^2p^{2e-1}(p-1)$ dapat dibagi oleh p^{e+1} karena $2e-1 \geq e+1$ untuk $e \geq 2$, jadi

$$h^{p^{e-1}(p-1)} \equiv 1 + kp^e \pmod{p^{e+1}}.$$

Karena p tidak membagi k, maka $kp^e \not\equiv 0 \pmod{p^{e+1}}$, jadi

$$h^{p^{e-1}(p-1)} \not\equiv 1 \pmod{p^{e+1}}.$$

Selesailah pembuktian induksi dan selesailah pembuktian teorema 38.

10.5 Homomorphism Theorem

Di bagian 5.2 kita telah membahas konsep *homomorphism* dan *ideal* untuk struktur *ring*. Disini akan kita lanjutkan pembahasan *homomorphism*, dimulai dengan pembahasan *homomorphism theorem* sampai dengan teorema *isomorphism*, yang akan digunakan dalam pembahasan *field extension* di bagian 10.6.

Untuk suatu *homomorphism* $\varphi : R \to S$ antar dua *ring*, dan setiap $a, b \in R$,

$$\varphi(a) = \varphi(b) \iff \varphi(a-b) = 0 \iff (a-b) \in \ker(\varphi).$$

Jadi φ mengumpulkan elemen-elemen R yang perbedaannya berada dalam *ideal* $\ker(\varphi)$. Jika *ideal* $I \subseteq \ker(\varphi)$, maka kita dapat uraikan efek dari φ menjadi dua langkah:

1. Kita kumpulkan elemen-elemen yang perbedaannya berada dalam *ideal* I melalui R/I.

2. Karena $I \subseteq \ker(\varphi)$ maka elemen-elemen yang perbedaannya berada dalam *ideal* I juga telah dikumpulkan oleh φ, jadi kita dapat melanjutkan dari R/I ke S sehingga komposisi langkah 1 dan 2 menghasilkan φ.

Inilah isi dari teorema berikut.

Teorema 39 (Homomorphism Theorem) *Jika $\varphi : R \longrightarrow S$ merupakan homomorphism antar ring, I merupakan ideal dari R dengan $I \subseteq ker(\varphi)$ dan kita beri label χ untuk canonical homomorphism dari R ke R/I, maka pemetaan ψ:*

$$\begin{array}{ccc} R/I & \longrightarrow & S \\ (a+I) & \mapsto & \varphi(a) \end{array}$$

telah didefinisikan dengan baik (well-defined). ψ adalah homomorphism antar ring yang mematuhi $\psi \circ \chi = \varphi$.

$$\begin{array}{ccc} R & \overset{\varphi}{\longrightarrow} & S \\ \chi \downarrow & \nearrow \psi & \\ R/I & & \end{array}$$

ψ surjective jika dan hanya jika φ surjective. ψ injective jika dan hanya jika $I = ker(\varphi)$.

Untuk menunjukkan bahwa ψ telah didefinisikan dengan baik, kita harus tunjukkan bahwa nilai $\varphi(a)$ tidak tergantung pada representasi $a + I$ yang dipilih. Jika $a, a' \in R$ dengan $a + I = a' + I$, maka

$$a - a' \in I \subseteq \ker(\varphi),$$

yang berarti

$$0 = \varphi(a - a') = \varphi(a) - \varphi(a').$$

Jadi $\varphi(a) = \varphi(a')$, yang berarti nilai $\varphi(a)$ independen dari representasi $a + I$ yang dipilih. Untuk menunjukkan bahwa ψ adalah suatu *homomorphism*, kita dapatkan

$$\begin{aligned} \psi((a+I)+(b+I)) &= \psi((a+b)+I) \\ &= \varphi(a+b) \\ &= \varphi(a)+\varphi(b) \\ &= \psi(a+I)+\psi(b+I). \end{aligned}$$

$$\begin{aligned}\psi((a+I)(b+I)) &= \psi((ab)+I)\\ &= \varphi(ab)\\ &= \varphi(a)\varphi(b)\\ &= \psi(a+I)\psi(b+I).\end{aligned}$$

$$\psi(1+I) = \varphi(1_R) = 1_S.$$

Untuk menunjukkan bahwa $\psi \circ \chi = \varphi$, jika $a \in R$,

$$\psi(\chi(a)) = \psi(a+I) = \varphi(a).$$

Jika φ *surjective*, maka $\psi \circ \chi$ *surjective*, jadi ψ *surjective*. Sebaliknya, jika ψ *surjective*, maka φ yang merupakan komposisi dua pemetaan *surjective* $(\psi \circ \chi)$ juga surjective. Jika ψ *injective*, untuk $a \in \ker(\varphi)$,

$$\psi(a+I) = \varphi(a) = 0.$$

Karena ψ *injective*, maka $a + I = 0_{R/I} = I$, jadi $a \in I$. Jadi $\ker(\varphi) \subseteq I$, dan karena $I \subseteq \ker(\varphi)$, maka $I = \ker(\varphi)$. Sebaliknya, dengan $I = \ker(\varphi)$, untuk menunjukkan bahwa ψ *injective*, kita harus tunjukkan bahwa $\ker(\psi) = \{0\} = \{I\}$. Jika $a + I \in \ker(\psi)$, maka $\varphi(a) = \psi(a + I) = 0$. Jadi $a \in \ker(\varphi)$, yang berarti $a \in I$. Jadi $a + I = I$, yang berarti $\ker(\psi)$ hanya berisi I. Selesailah pembuktian teorema 39.

Tiga teorema berikut adalah teorema mengenai *isomorphism* yang sudah merupakan standard dalam aljabar abstrak.

Teorema 40 (First Isomorphism Theorem) *Jika $\varphi : R \longrightarrow S$ merupakan homomorphism antar ring, maka*

$$R/ker(\varphi) \simeq \varphi(R).$$

Untuk membuktikan teorema ini, kita gunakan teorema 39 dengan $I = \ker(\varphi)$, dengan hasil *injective homomorphism* ψ:

$$\begin{aligned}R/\ker(\varphi) &\longrightarrow S\\ \psi(a+\ker(\varphi)) &\mapsto \varphi(a).\end{aligned}$$

Karena $\psi \circ \chi = \varphi$, dimana χ adalah *canonical homomorphism* dari R ke $R/\ker(\varphi)$, kita dapatkan $\psi(R/\ker(\varphi)) = \varphi(R)$. Jadi sebagai pemetaan dari $R/\ker(\varphi)$ ke $\psi(R/\ker(\varphi))$, ψ bersifat *surjective*. Karena bersifat *injective* dan *surjective*, ψ dari $R/\ker(\varphi)$ ke $\psi(R/\ker(\varphi))$ bersifat *bijective*, jadi (karena $\psi(R/\ker(\varphi)) = \varphi(R)$)

$$R/\ker(\varphi) \simeq \varphi(R).$$

Selesailah pembuktian teorema 40.

Sebelum membahas teorema kedua mengenai *isomorphism*, kita perlu definisikan terlebih dahulu konsep *subring* dan *subfield*.

Definisi 18 (Subring) *Untuk ring R dan $S \subseteq R$, jika*

- $1 \in S$,
- $a - b \in S$ *untuk setiap* $a, b \in S$ *dan*
- $ab \in S$ *untuk setiap* $a, b \in S$,

maka S adalah subring dari R.

Tidak terlalu sulit untuk menunjukkan bahwa $0 \in S$ karena $a - a \in S$, jadi S juga mempunyai struktur *ring*. Jika S merupakan suatu *field*, maka S disebut *subfield* dari R. Jika S dan R merupakan *field*, maka R adalah *field extension* dari S. Dalam teori mengenai *field*, biasanya kita hanya ingin menunjukkan bahwa K *isomorphic* dengan *subfield* dari F, dimana K dan F keduanya merupakan *field*. Untuk itu kita hanya perlu menunjukkan bahwa terdapat *injective field homomorphism* dari K ke F. Ini dapat ditunjukkan menggunakan teorema 40 (*First Isomorphism Theorem*) dengan $R = K, S = F,$ dan $\ker(\varphi) = \{0\}$.

Teorema 41 (Second Isomorphism Theorem) *Jika R adalah suatu ring, S merupakan subring dari R, I merupakan proper ideal dari R, maka*

$$S/(S \cap I) \simeq (S + I)/I$$

dimana $S + I = \{s + a | s \in S, a \in I\}$.

Mari kita buktikan teorema 41. Jika $\iota : S \longrightarrow (S + I)$ dan $\chi : (S + I) \longrightarrow (S + I)/I$ adalah *homomorphism* dengan

$$\begin{aligned} \iota : s &\mapsto s \\ \chi : a &\mapsto a + I \end{aligned}$$

maka $\varphi = \chi \circ \iota$ juga merupakan *homomorphism* dari S ke $(S + I)/I$, jadi

$$S/\ker(\varphi) \simeq \varphi(S).$$

Jadi jika kita dapat tunjukkan bahwa $\ker(\varphi) = S \cap I$ dan $\varphi(S) = (S + I)/I$ (φ *surjective*) maka selesailah pembuktian kita. Jika $s \in \ker(\varphi)$ maka $s \in I$, dan karena $s \in S$, maka $s \in S \cap I$. Sebaliknya jika $s \in S \cap I$ maka $\varphi(s) = I$, jadi $s \in \ker(\varphi)$. Jadi kita sudah tunjukkan bahwa $\ker(\varphi) = S \cap I$. Untuk menunjukkan bahwa φ *surjective*, kita harus tunjukkan bahwa untuk sembarang $b + I \in (S + I)/I$ terdapat $s \in S$ dimana $\varphi(s) = b + I$. Jika kita pilih $s \in S$ dan $a \in I$ dimana $b = s + a$ maka

$$\begin{aligned} \varphi(s) &= s + I \\ &= (s + a) + I \\ &= b + I. \end{aligned}$$

Selesailah pembuktian teorema 41.

Sebelum membahas teorema ketiga mengenai *isomorphism*, kita perlu definisikan dahulu notasi J/I untuk I dan J berupa *ideal*, dan buat teorema mengenai *bijection* antara himpunan *ideal*.

Definisi 19 *Jika I merupakan proper ideal dalam R dan J merupakan ideal dalam R dengan $I \subseteq J$, maka*

$$J/I = \{a + I | a \in J\}.$$

Jadi J/I merupakan *image* dari J berdasarkan *canonical homomorphism* dari R ke R/I dan tidak terlalu sulit untuk melihat bahwa J/I merupakan *ideal* dalam R dan dalam R/I.

Teorema 42 *Jika I merupakan proper ideal dalam ring R, maka terdapat bijection dari himpunan ideal yang mencakup I dalam R ke himpunan ideal dalam R/I. Bijection tersebut mempunyai pemetaan sebagai berikut:*

$$J \mapsto J/I.$$

Kita mulai pembuktian teorema 42 dengan membuat

$$\varphi : R \longrightarrow R/I$$

sebagai *canonical homomorphism*, jadi $\ker(\varphi) = I$. Tidak terlalu sulit untuk melihat bahwa φ memetakan suatu *homomorphism* J ke J/I sesuai teorema 42. Jika $\mathcal{I}_\varphi$ merupakan himpunan semua *ideal* dalam R yang mencakup I dan $\mathcal{I}_S$ merupakan himpunan semua *ideal* dalam R/I, kita ingin tunjukkan bahwa pemetaan

$$\begin{array}{rcl} \chi : \mathcal{I}_\varphi & \longrightarrow & \mathcal{I}_S \\ J_\varphi & \mapsto & \varphi(J_\varphi) \end{array}$$

merupakan pemetaan yang *bijective*, dimana $\varphi(J_\varphi)$ merupakan *image* dari J_φ menurut φ. Jadi kita harus tunjukkan bahwa dengan pemetaan

$$\begin{array}{rcl} \kappa : \mathcal{I}_S & \longrightarrow & \mathcal{I}_\varphi \\ J_S & \mapsto & \varphi^{-1}(J_S) \end{array}$$

maka

$$\kappa \circ \chi = \mathrm{id}_{\mathcal{I}_\varphi}$$

dan

$$\chi \circ \kappa = \mathrm{id}_{\mathcal{I}_S},$$

dimana $\varphi^{-1}(J_S)$ adalah *inverse image* dari J_S menurut φ. Untuk menunjukkan persamaan pertama, kita harus tunjukkan bahwa

$$\varphi^{-1}(\varphi(J_S)) = J_S$$

untuk setiap $J_S \in \mathcal{I}_S$. Jelas bahwa $J_S \subseteq \varphi^{-1}(\varphi(J_S))$. Untuk menunjukkan bahwa $\varphi^{-1}(\varphi(J_S)) \subseteq J_s$, mari kita lihat apa konsekuensi dari $a \in \varphi^{-1}(\varphi(J_S))$. Karena $\varphi(a) \in \varphi(J_S)$ maka terdapat $a' \in J_S$ dimana $\varphi(a) = \varphi(a')$. Jadi $a - a' \in \ker(\varphi) \subseteq J_S$, jadi

$$a = a' + (a - a') \in J_S.$$

Untuk menunjukkan persamaan kedua, kita harus tunjukkan bahwa

$$\varphi(\varphi^{-1}(J_\varphi)) = J_\varphi$$

untuk setiap $J_\varphi \in \mathcal{I}_\varphi$, yang jelas terpenuhi karena φ *surjective*. Selesailah pembuktian teorema 42.

Teorema 43 (Third Isomorphism Theorem) *Jika R adalah suatu ring, I dan J merupakan proper ideal dari R dengan $I \subseteq J$ maka*

$$R/J \simeq (R/I)/(J/I).$$

Jika

$$\chi_1 : R \longrightarrow R/I \text{ dan } \chi_2 : R/I \longrightarrow (R/I)/(J/I)$$

keduanya merupakan *canonical homomorphism*, maka

$$\varphi = \chi_2 \circ \chi_1$$

yang merupakan komposisi dari dua *surjective homomorphism* merupakan suatu *surjective homomorphism*. Berarti

$$R/\ker(\varphi) \simeq \varphi(R) = (R/I)/(J/I).$$

Jadi kita tinggal menunjukkan bahwa $\ker(\varphi) = J$. Jika $a \in \ker(\varphi)$ maka

$$\varphi(a) = (a + I) + J/I = J/I$$

karena J/I merupakan 0 dalam $(R/I)/(J/I)$. Jadi $a+I \in J/I$ dan berdasarkan teorema 42, $a \in J$. Sebaliknya jika $a \in J$, maka berdasarkan definisi dari J/I, $a + I \in J$, jadi

$$\varphi(a) = (a + I) + J/I = J/I$$

yang berarti $a \in \ker(\varphi)$. Selesailah pembuktian teorema 43.

10.6 Field Extension

Kita akan bahas konsep *field extension*, tetapi sebelumnya perlu dijelaskan konsep *restriction* yang berlaku pada relasi (termasuk fungsi dan *morphism*). Suatu relasi dapat dipandang sebagai himpunan dari pasangan berurut (*ordered pair*), dimana elemen pertama dalam suatu pasangan berasal dari *domain* relasi dan elemen kedua berasal dari *range* relasi. Untuk suatu relasi R, R *restricted* pada D (diberi notasi $R \restriction D$) adalah subrelasi dari R yang terdiri dari semua pasangan dalam R yang elemen pertamanya berada dalam D. Kita definisikan *restriction* secara formal:

Definisi 20 (Restriction)

$$R \restriction D = \{x | x \in R \text{ dan } fst(x) \in D\}$$

dimana fst adalah fungsi yang memberi elemen pertama dalam pasangan berurut.

Jika F dan K keduanya merupakan *field*, K merupakan *subfield* dari F, dan $A = \{a_1, a_2, \ldots, a_n\} \subseteq F$, maka dengan "menambahkan" (*adjoining*) A pada K (yang diberi notasi $K(A)$) kita dapatkan *field extension* yang merupakan *intersection* dari semua *subfield* F yang mencakup K dan A. $K(A)$ terdiri dari semua elemen F yang dapat ditulis sebagai

$$f(a_1, a_2, \ldots, a_n) \cdot (g(a_1, a_2, \ldots, a_n))^{-1},$$

dimana $f, g \in K[x_1, x_2, \ldots, x_n]$, $a_1, a_2, \ldots, a_n \in A$ dan $g(a_1, a_2, \ldots, a_n) \neq 0$. ($K(A)$ merupakan himpunan semua ekspresi rasional yang melibatkan elemen dari K dan A.) Jika $A = \{a\}$, notasi $K(a)$ sering digunakan untuk $K(\{a\})$, dan *extension* disebut *simple extension*. Kita akan fokus pada *simple extension* karena *extension* dengan beberapa elemen dapat dipandang sebagai runtunan dari beberapa *simple extension*.

Definisi 21 (Algebraic) *Elemen $a \in F$ disebut algebraic atas K jika terdapat polynomial $0 \neq f \in K[x]$ dengan $f(a) = 0$ (a merupakan akar dari polynomial f). Extension dengan elemen algebraic disebut algebraic extension.*

Jika elemen $a \in F$ tidak *algebraic* atas K maka a disebut *transcendental* atas K dan *extension* dengan a disebut *transcendental extension*. Untuk elemen a yang *algebraic*, karena terdapat *polynomial* $0 \neq f \in K[x]$ dengan akar a, maka terdapat *polynomial* dengan *degree* minimal dengan akar a. Lebih dari itu, terdapat *monic polynomial* f dengan akar a yang mempunyai *degree* minimal karena setiap koefisien berasal dari *field* K (jadi dapat dikalikan dengan elemen *inverse* dari K juga untuk menghasilkan 1). Kita ingin tunjukkan bahwa *monic polynomial* dengan *degree* minimal itu unik. Jadi, jika $0 \neq f, g \in K[x]$ keduanya merupakan *monic polynomial* dengan *degree* minimal yang mempunyai akar

a dan $\deg(f) = \deg(g)$), kita ingin tunjukkan bahwa $f = g$. Karena $K[x]$ merupakan *Euclidean domain* (lihat bagian 5.6), maka terdapat $q, r \in K[x]$ dengan

$$f = qg + r \text{ dan } \deg(r) < \deg(g) \text{ atau } r = 0.$$

Kita dapatkan

$$r(a) = f(a) - q(a)g(a) = 0,$$

dan karena f dan g mempunyai *degree* minimal untuk *polynomial* dengan akar a, maka tidak mungkin $\deg(r) < \deg(g)$. Jadi $r = 0$ dan

$$f = qg.$$

Karena f dan g merupakan *monic polynomial* dengan *degree* yang sama, maka $q = 1$ dan $f = g$ seperti yang diharapkan. *Polynomial* seperti diatas disebut *minimal polynomial* dari a atas K dengan notasi $\min_K^a$. Jadi kita baru saja membuktikan teorema berikut:

Teorema 44 *Jika $a \in F$ algebraic atas K, maka terdapat suatu monic polynomial unik dengan degree minimal (polynomial diberi notasi $\min_K^a$) dimana $\min_K^a(a) = 0$.*

Sekarang kita ingin buktikan teorema berikut:

Teorema 45 *Jika $a \in F$ algebraic atas K dan $f \in K[x]$, maka $f(a) = 0$ jika dan hanya jika $\min_K^a$ membagi f dalam struktur ring $K[x]$.*

Jika $\min_K^a$ membagi f dalam $K[x]$ maka terdapat $q \in K[x]$ dimana $f = q \min_K^a$, jadi

$$f(a) = q(a)\min_K^a(a) = q(a) \cdot 0 = 0.$$

Sebaliknya, jika $f(a) = 0$, maka terdapat $q, r \in K[x]$ dimana

$$f = q \min_K^a + r$$

dan

$$\deg(r) < \deg(\min_K^a) \text{ atau } r = 0.$$

Dengan argumentasi yang sama seperti dalam pembuktian teorema 44 kita dapatkan $r = 0$, jadi $f = q \min_K^a$.

Teorema 46 *Jika $a \in F$ algebraic atas K, maka $\min_K^a$ merupakan monic irreducible polynomial unik dalam $K[x]$ dengan akar a.*

Untuk membuktikan teorema ini, kita harus tunjukkan bahwa $\min_K^a$ *irreducible* dan satu-satunya *monic irreducible polynomial* dalam $K[x]$ dengan akar a. Jika $\min_K^a$ dapat diuraikan dalam $K[x]$, misalnya

$$\min_K^a = fg$$

maka

$$\deg(f) < \deg(\min\nolimits_K^a) \text{ dan } \deg(g) < \deg(\min\nolimits_K^a),$$

dan $f(a) = 0$ atau $g(a) = 0$ karena $f(a)g(a) = \min_K^a(a) = 0$. Tetapi ini merupakan kontradiksi, karena $\min_K^a$ minimal (tidak mungkin $f(a) = 0$ atau $g(a) = 0$). Jadi $\min_K^a$ *irreducible*. Untuk menunjukkan bahwa *monic irreducible polynomial* dalam $K[x]$ dengan akar a harus sama dengan $\min_K^a$, teorema 45 mengatakan *polynomial* tersebut harus merupakan kelipatan dari $\min_K^a$, dan karena *irreducible*, maka harus sama dengan $\min_K^a$.

Mari kita lihat apa yang terjadi jika kita lakukan *simple extension* yang bersifat *transcendental* terhadap suatu *field*. Jika K merupakan suatu *field* dan F merupakan *rational function field* atas K (F terdiri dari semua pecahan dengan *numerator* dan *denominator* dari $K[x]$, tentunya dengan pengecualian *denominator* tidak boleh 0), maka F merupakan *field extension* dari K. Karena dalam F, $g = g(x) = 0$ jika dan hanya jika g merupakan *zero polynomial*, maka x *transcendental* atas K. Jika kita lakukan *simple extension* terhadap K menggunakan x, maka kita dapatkan F. Jadi *simple extension* terhadap K menggunakan x yang *transcendental* terhadap K menghasilkan *rational function field* $K(x)$. Karena *rational function field* tidak *finite* (banyaknya *polynomial* adalah *infinite*), maka *field extension* yang bersifat *transcendental* tidak mungkin menghasilkan *finite field*.

Sekarang kita lihat apa yang terjadi jika kita lakukan *simple extension* yang bersifat *algebraic* terhadap suatu *field*. Jika g merupakan *monic irreducible polynomial* dalam $K[x]$, maka teorema 26 (lihat bagian 5.7) mengatakan bahwa $K[x]/gK[x]$ merupakan suatu *field*. Mari kita beri notasi $\overline{f}$ untuk *residue class* $f + gK[x]$. Kita teliti bagian dari *canonical homomorphism*

$$\begin{array}{rcl} \varphi : K[x] & \longrightarrow & K[x]/gK[x] \\ f & \mapsto & \overline{f} \end{array}$$

yang berlaku pada K (yaitu $\varphi \restriction K$). Karena semua elemen dari K merupakan konstan dalam $K[x]$, maka $\varphi(K) = K$, jadi K merupakan *subfield* dari $K[x]/gK[x]$.

Jika $f = \sum_{i=1}^m a_i x^i \in K[x]$, sifat *homomorphism* dari φ menghasilkan

$$\overline{f} = \overline{\sum_{i=1}^m a_i x^i} = \sum_{i=1}^m a_i \overline{x}^i = f(\overline{x}).$$

Jadi $g(\overline{x}) = \overline{g} = 0$, dan $f(\overline{x}) = \overline{f} \neq 0$ jika $\deg(f) < \deg(g)$. Jika kita buat $F = K[x]/gK[x]$, maka $\overline{x} \in F$ bersifat *algebraic* atas K dengan *minimal polynomial* $g \in K[x]$. Lebih dari itu, elemen-elemen dari F mempunyai format $\overline{f} = f(\overline{x})$ dengan $f \in K[x]$ (karena F merupakan *quotient ring* dari $K[x]$). Karena F juga merupakan suatu field, maka *elemen* F yang bukan 0 mempunyai *inverse*

yang juga *elemen* F yang bukan 0. Jadi sebenarnya elemen-elemen dari F meliputi semua ekspresi yang mempunyai format

$$f(\overline{x}) \cdot (h(\overline{x}))^{-1},$$

dimana $f, h \in K[x]$, dan $h(\overline{x}) \neq 0$. (F merupakan himpunan semua ekspresi rasional yang melibatkan elemen dari K dan $\{\overline{x}\}$.) Jadi $K(\overline{x}) = F$.

Kita telah membuktikan teorema berikut mengenai *simple field extension*:

Teorema 47 *Untuk suatu field K:*

1. *Jika F merupakan rational function field atas K dengan variabel x, maka F adalah simple extension dari K dengan elemen transcendental x.*

2. *Jika g merupakan monic irreducible polynomial dalam $K[x]$, maka $F = K[x]/gK[x]$ adalah simple extension dari K dengan elemen algebraic $\overline{x}$ yang mempunyai minimal polynomial g.*

Karena *field extension* menggunakan elemen *transcendental* akan menghasilkan *field* yang tidak *finite*, maka untuk mendapatkan *finite field*, *field extension* harus menggunakan elemen *algebraic* (dan *extension* harus dilakukan pada *finite field*).

Berikut kita ingin tunjukkan bahwa *simple field extension* selalu menghasilkan *extension field* yang unik (*up to isomorphism*[1]). *Isomorphism* antara *field extension* yang sama adalah *isomorphism* yang khusus: jika K' dan K'' keduanya merupakan *field extension* dari K, maka K' disebut *K-isomorphic* dengan K'' jika terdapat *isomorphism*

$$\varphi : K' \longrightarrow K''$$

dengan $\varphi \restriction K = \mathrm{id}_K$. Jadi *isomorphism* diantara kedua *extension field* mempertahankan struktur *base field*.

Teorema 48 *Untuk $a \in K'$ dimana K' merupakan field extension dari K, jika a transcendental atas K maka $K(a)$ K-isomorphic dengan $K(x)$ dimana x dipetakan ke a. Jika a algebraic atas K maka $K(a)$ K-isomorphic dengan $K[x]/\min_K^a K[x]$ dimana $\overline{x} = x + \min_K^a K[x]$ dipetakan ke a.*

Untuk pembuktian teorema 48 kita gunakan *homomorphism*:

$$\begin{array}{rcl} \varphi : K[x] & \longrightarrow & K' \\ f & \mapsto & f(a). \end{array}$$

[1]Dalam aljabar abstrak, unik selalu berarti *up to isomorphism* karena dua struktur yang *isomorphic* dari sudut pandang abstrak dianggap sama.

Jika a *transcendental* atas K maka $\ker(\varphi) = \{0\}$, jadi φ bersifat *injective* dan dapat diperluas menjadi *homomorphism*:

$$\psi : K(x) \quad \longrightarrow \quad K'$$

dengan $\psi \restriction K[x] = \varphi$ dan $\psi(f/g) = \varphi(f) \cdot (\varphi(g))^{-1}$. Tidak terlalu sukar untuk melihat bahwa ψ bersifat *injective* dan $\psi(K(x)) = K(a)$. Jika a *algebraic* atas K maka dengan menggunakan teorema 45 kita dapatkan $\ker(\varphi) = \min_K^a K[x]$. Berdasarkan teorema 40, $K[x]/\min_K^a K[x]$ *isomorphic* dengan $\varphi(K[x])$ menggunakan:

$$\begin{array}{rcl} \psi : K[x]/\min_K^a K[x] & \longrightarrow & \varphi(K[x]) \\ \overline{f} & \mapsto & f(a). \end{array}$$

Jelas bahwa $\psi \restriction K = \mathrm{id}_K$ jadi ψ merupakan suatu K-*isomorphism*. Jadi kita dapatkan $\psi(K[x]/\min_K^a K[x])$ yang merupakan *subfield* dari K' yang mencakup seluruh K dan mempunyai a sebagai elemen, dan setiap elemennya mempunyai bentuk $f(a)$ dimana $f \in K[x]$. Jadi $\psi(K[x]/\min_K^a K[x])$ harus sama dengan $K(a)$. Selesailah pembuktian teorema 48.

Sekarang mari kita tunjukkan bahwa *field extension* dengan elemen *algebraic* dapat dipandang sebagai ruang vektor, dimana *field* semula menjadi *scalar*. Kita beri label K untuk field semula dan label a untuk elemen *algebraic* yang digunakan oleh *field extension*. Jadi terdapat *minimal polynomial* $\min_K^a$ yang kita beri label g dan mempunyai format:

$$g = x^n + b_{n-1}x^{n-1} + \ldots + b_0.$$

Kita ingin tunjukkan bahwa $\{1, a, a^2, \ldots, a^{n-1}\}$ merupakan basis untuk $K(a)$ sebagai ruang vektor atas K. Untuk menunjukkan bahwa pangkat-pangkat a tersebut independen secara linear sangat mudah karena jika ada dependensi linear maka terdapat *polynomial* dengan *degree* kurang dari n yang mempunyai akar a, sesuatu yang kontradiksi dengan minimalitas dari g. Untuk menunjukkan bahwa pangkat-pangkat a tersebut merupakan *spanning set* untuk ruang vektor, kita mengetahui bahwa $K(a)$ terdiri dari semua elemen yang dapat diekspresikan sebagai *polynomial* dengan a menggantikan variabel x. Untuk pemangkatan a dengan $r > n-1$, kita gunakan fakta bahwa $g(a) = 0$. Jadi

$$\begin{array}{rcl} a^{r-n}g(a) & = & 0, \\ a^{r-n}(a^n + b_{n-1}a^{n-1} + \ldots + b_0) & = & 0, \\ a^r & = & -b_{n-1}a^{r-1} - \ldots - b_0a^{r-n}. \end{array}$$

Dengan menggunakan prinsip induksi kita dapatkan bahwa setiap pemangkatan a dengan $r > n-1$ dapat digantikan dengan kombinasi linear pemangkatan a yang merupakan elemen-elemen dari $\{1, a, a^2, \ldots, a^{n-1}\}$. Jadi selesai sudah

pembuktian bahwa $\{1, a, a^2, \ldots, a^{n-1}\}$ merupakan basis untuk $K(a)$ sebagai ruang vektor.

Berikutnya kita akan bahas konsep *algebraic closure*.

Definisi 22 (Algebraically Closed) *Field F disebut algebraically closed jika setiap polynomial f dalam $F[x]$ dengan $deg(f) > 0$ mempunyai akar (solusi x untuk $f(x) = 0$) dalam F.*

Sebagai contoh, berdasarkan *Fundamental Theorem of Algebra*, $\mathbf{C}$ (*field* untuk bilangan kompleks) merupakan *field* yang *algebraically closed*.

Definisi 23 (Algebraic Closure) *Algebraic closure dari suatu field K adalah suatu algebraic field extension F atas K dimana F merupakan field yang algebraically closed.*

Tentunya jika K adalah suatu *field* yang *algebraically closed*, maka *algebraic closure* dari K adalah K. Setiap *field* memiliki *algebraic closure*, akan tetapi kita tidak akan membuktikan itu disini karena pembuktiannya cukup rumit dan memerlukan penggunaan *Axiom of Choice*.

10.7 Finite Field

Konsep *finite field* telah diperkenalkan pada bab-bab sebelum ini, dengan contoh aplikasi *polynomial field* yang digunakan oleh enkripsi AES. Sebenarnya *polynomial field* adalah cara pandang atau implementasi dari *finite field*, dengan kata lain setiap *finite field* dapat diimplementasi sebagai *polynomial field*. Di bagian ini kita akan bahas esensi dari *finite field* terlepas dari implementasi. Kita akan mendalami lebih lanjut teori mengenai *finite field*, termasuk pembahasan konsep *characteristic* dan *generator*.

Untuk suatu *field*, jika kelipatan dari 1 tidak akan dapat menghasilkan 0, maka *characteristic* dari *field* tersebut adalah 0. Jika dapat menghasilkan 0, maka *characteristic* adalah kelipatan terkecil p dari 1 yang menghasilkan 0:

$$\underbrace{1 + 1 + \ldots + 1}_{p} = 0.$$

Characteristic dari suatu *field* harus berupa bilangan prima (kecuali jika *characteristic* $= 0$), karena jika *characteristic* tidak prima dan dapat diuraikan, misalnya $p = mn$ dimana $1 < m, n < p$, maka

$$0 = \underbrace{1 + 1 + \ldots + 1}_{p} = (\underbrace{1 + 1 + \ldots + 1}_{m})(\underbrace{1 + 1 + \ldots + 1}_{n}),$$

yang berarti $\underbrace{1 + 1 + \ldots + 1}_{m} = 0$ atau $\underbrace{1 + 1 + \ldots + 1}_{n} = 0$, sesuatu yang mustahil karena p adalah kelipatan terkecil dari 1 dengan hasil 0. *Characteristic*

dari suatu *finite field* $\mathbf{F}$ harus berupa bilangan prima karena jika 0, maka himpunan dari semua elemen $\mathbf{F}$ harus mencakup $\mathbf{N}$ dan juga $\mathbf{Q}$ ($\mathbf{Q}$ merupakan *subfield* dari $\mathbf{F}$), jadi $\mathbf{F}$ tidak *finite* — suatu kontradiksi. Suatu *finite field* dengan *characteristic* p mempunyai $\mathbf{F}_p$ (atau $\mathbf{GF}(p)$, *Galois field* dengan p elemen) sebagai *subfield*. Sebetulnya lebih tepat jika dikatakan bahwa $\mathbf{F}_p$ *isomorphic* dengan *subfield* yang bersangkutan, tetapi dalam teori *finite field*, dua *field* yang *isomorphic* dianggap sama, hanya implementasinya mungkin berbeda. Mari kita coba buktikan bahwa $\mathbf{F}_p$ *isomorphic* dengan *subfield* dari *field* dengan *characteristic* p. Jadi harus kita tunjukkan bahwa terdapat *injective homomorphism* φ dari $\mathbf{F}_p$ ke $\mathbf{F}$, dimana $\mathbf{F}$ merupakan *field* dengan *characteristic* p. Kita definisikan

$$\begin{aligned} \varphi : \mathbf{F}_p &\longrightarrow \mathbf{F} \\ m &\mapsto m \cdot 1_{\mathbf{F}}. \end{aligned}$$

Definisi diatas merupakan definisi yang baik (*well-defined*) karena jika $m = n$ dalam $\mathbf{F}_p$, maka $p|(m-n)$, yang berarti

$$\begin{aligned} (m-n)1_{\mathbf{F}} &= 0, \\ m \cdot 1_{\mathbf{F}} &= n \cdot 1_{\mathbf{F}}. \end{aligned}$$

Definisi juga menghasilkan φ yang *injective* karena jika $m, n \in \mathbf{F}_p$ keduanya dipetakan ke elemen $\mathbf{F}$ yang sama, maka

$$\begin{aligned} m \cdot 1_{\mathbf{F}} &= n \cdot 1_{\mathbf{F}}, \\ (m-n)1_{\mathbf{F}} &= 0, \end{aligned}$$

jadi karena $p|(m-n)$ dan $m, n \in \mathbf{F}_p$ maka $m = n$, yang berarti φ adalah *injective*. Sekarang tinggal kita tunjukkan bahwa φ merupakan *field homomorphism*. Dari definisi φ kita dapatkan $\varphi(1) = 1_{\mathbf{F}}$ dan $\varphi(0) = 0_{\mathbf{F}}$. Untuk $m + n$ kita dapatkan

$$\begin{aligned} \varphi(m+n) &= (m+n) \cdot 1_{\mathbf{F}} \\ &= m \cdot 1_{\mathbf{F}} + n \cdot 1_{\mathbf{F}} \\ &= \varphi(m) + \varphi(n). \end{aligned}$$

Untuk mn kita dapatkan

$$\begin{aligned} \varphi(mn) &= mn \cdot 1_{\mathbf{F}} \\ &= mn \cdot 1_{\mathbf{F}} \cdot 1_{\mathbf{F}} \\ &= (m \cdot 1_{\mathbf{F}})(n \cdot 1_{\mathbf{F}}) \\ &= \varphi(m)\varphi(n). \end{aligned}$$

Jadi φ merupakan *field homomorphism*.

Untuk suatu *finite field* $\mathbf{F}_q$, terdapat $q-1$ elemen non 0. Kumpulan dari elemen non 0 membentuk suatu *multiplicative group* $\mathbf{F}_q^*$, dan setiap elemen a dalam *group* tersebut mempunyai *order*, yaitu pangkat positif terkecil dari a yang menghasilkan 1. Untuk melihat bahwa suatu elemen non 0 a dalam $\mathbf{F}_q^*$ mempunyai *order* positif, jika semua pangkat a berbeda, maka *field* tidak *finite*, suatu kontradiksi. Jadi terdapat $m > n, a^m = a^n$, atau $a^{m-n} = 1$, yang berarti ada pangkat positif dari a yang menghasilkan 1. Karena pangkat positif merupakan subset dari $\mathbf{N}$, prinsip *well-ordering* mengatakan terdapat pangkat positif terkecil dari a yang menghasilkan 1, pangkat tersebut merupakan *order* dari a dalam $\mathbf{F}_q^*$.

Teorema 49 *Jika a adalah elemen non* 0 *dari finite field* $\mathbf{F}_q$*, order dari a (yang kita beri label d) membagi* $q-1$.

Pertama kita tunjukkan dahulu bahwa $a^{q-1} = 1$. Untuk itu, kita deretkan semua elemen $\mathbf{F}_q^*$ sebanyak $q-1$ elemen:

$$a_1, a_2, \ldots, a_{q-1}$$

dan ambil produknya:

$$p_1 = a_1 a_2 \ldots a_{q-1}.$$

Jika kita kalikan setiap elemen dalam deretan dengan a, maka akan kita dapatkan deretan dengan $q-1$ elemen juga:

$$b_1, b_2, \ldots, b_{q-1}$$

dimana $b_i = a_i a$. Kita ambil produk deretan kedua:

$$p_2 = b_1 b_2 \ldots b_{q-1}.$$

Karena dua elemen $\mathbf{F}_q^*$ yang berbeda jika masing-masing dikalikan dengan a menghasilkan dua elemen $\mathbf{F}_q^*$ yang berbeda juga, maka semua elemen dalam deretan kedua berbeda, jadi deretan kedua terdiri dari semua elemen $\mathbf{F}_q^*$. Perbedaan antara deretan pertama dengan deretan kedua hanya terletak pada urutan elemen. Jadi

$$p_1 = p_2 = p_1 a^{q-1}$$

yang berarti $a^{q-1} = 1$. Kembali ke pembuktian semula yaitu menunjukkan bahwa d membagi $q-1$, jika d tidak membagi $q-1$, maka terdapat $0 < r < d$ dimana

$$q - 1 = bd + r$$

untuk suatu b, dan

$$a^r = a^{q-1-bd} = 1.$$

Ini adalah suatu kontradiksi karena d merupakan pangkat positif a terkecil yang menghasilkan 1. Jadi d membagi $q-1$, dan selesailah pembuktian teorema 49.

Definisi 24 (Generator) *Elemen dari* $\mathbf{F}_q^*$ *yang mempunyai order* $q-1$ *disebut generator dari* $\mathbf{F}_q^*$. *Setiap elemen* $\mathbf{F}_q^*$ *merupakan hasil pemangkatan generator, jadi hasil-hasil pemangkatan generator "mengunjungi" semua elemen* $\mathbf{F}_q^*$.

Teorema 50 *Setiap finite field* $\mathbf{F}_q$ *mempunyai generator untuk multiplicative group* $\mathbf{F}_q^*$. *Jika* g *adalah generator untuk* $\mathbf{F}_q^*$, *maka* g^j *juga merupakan generator untuk* $\mathbf{F}_q^*$ *jika dan hanya jika* $\gcd(j, q-1) = 1$. *Jadi terdapat* $\phi(q-1)$ *generator untuk* $\mathbf{F}_q^*$.

Untuk pembuktian teorema 50, mari kita fokus pada suatu elemen $a \in \mathbf{F}_q^*$ yang mempunyai *order* d (jadi $a^d = 1$ dan tidak ada pemangkatan a dengan sesuatu yang lebih kecil dari d yang menghasilkan 1). Teorema 49 mengatakan bahwa $d|q-1$. Juga, karena a^d merupakan pemangkatan terkecil yang menghasilkan 1, maka $a, a^2, \ldots, a^d$ semua merupakan elemen yang berbeda. Kita ingin tunjukkan bahwa elemen-elemen yang mempunyai *order* d adalah yang bernilai a^j dimana $\gcd(j, d) = 1$, jadi ada $\phi(d)$ elemen yang mempunyai order d. Kita ketahui bahwa setiap pemangkatan a diatas merupakan solusi dari persamaan $x^d = 1$, jadi semua merupakan akar dari $x^d - 1$. Kita juga mengetahui bahwa $x^d - 1$ tidak mempunyai akar ganda karena $x^d - 1$ dan derivatifnya (dx^{d-1}) tidak mempunyai pembagi persekutuan. Alhasil hanya pemangkatan a yang dapat menjadi akar dari $x^d - 1$, jadi elemen dengan *order* d harus merupakan pemangkatan dari a. Namun tidak semua pemangkatan a merupakan elemen dengan *order* d, karena jika $\gcd(j, d) = d' > 1$, maka a^j mempunyai order lebih kecil dari d yaitu d/d'. (d/d' dan j/d' keduanya merupakan bilangan bulat, dan $(a^j)^{(d/d')} = (a^d)^{j/d'} = 1$.) Kita juga harus tunjukkan bahwa jika $\gcd(j, d) = 1$ maka a^j mempunyai *order* d. Untuk itu kita tunjukkan bahwa jika *order* dari a^j lebih kecil dari d, sebut saja d'', maka kita akan dapatkan suatu kontradiksi. Karena d'' merupakan *order* dari a^j maka

$$(a^j)^{d''} = 1 = (a^{d''})^j.$$

Kita juga mengetahui bahwa

$$(a^{d''})^d = 1.$$

Jadi $a^{d''}$ dipangkatkan dengan $\gcd(j, d) = 1$ akan menghasilkan 1, dengan kata lain $a^{d''} = 1$. Tetapi ini merupakan suatu kontradiksi karena tidak ada pemangkatan a dengan sesuatu yang lebih kecil dari d yang dapat menghasilkan 1. Jadi kita sudah buktikan bahwa jika suatu elemen a mempunyai *order* d, maka terdapat $\phi(d)$ elemen dengan *order* d. Sekarang tinggal menunjukkan bahwa untuk setiap $d|q-1$ terdapat elemen dengan *order* d. Kita gunakan teorema 33 yang mengatakan bahwa

$$\sum_{d|q-1} \phi(d) = q - 1.$$

Jika terdapat $d|q-1$ dimana tidak ada elemen dengan *order* d, karena teorema 49 mengatakan *order* dari setiap elemen dari $\mathbf{F}_q^*$ membagi $q-1$, maka jumlah elemen dari $\mathbf{F}_q^*$ akan lebih kecil dari $q-1$, suatu kontradiksi. Jadi kita sudah buktikan bahwa untuk setiap $d|q-1$, terdapat $\phi(d)$ elemen dengan *order* d, jadi terdapat $\phi(q-1)$ elemen dengan *order* $q-1$, dengan kata lain terdapat $\phi(q-1)$ *generator* untuk $\mathbf{F}_q^*$. Selesailah pembuktian teorema 50.

Satu lagi konsep yang perlu dijelaskan sebelum penjelasan hasil terpenting mengenai *finite field* adalah konsep *splitting field*. Untuk suatu *field* K dan *polynomial* non-konstan f dalam $K[x]$, jika f dapat diuraikan menjadi produk dari *polynomial* linear dalam $K[x]$, dengan kata lain

$$f = f_1 f_2 \dots f_n$$

dimana setiap f_i untuk $1 \leq i \leq n$ adalah *poynomial* linear dalam $K[x]$, maka f disebut *splits* dalam K. Untuk suatu *field* F dan *polynomial* f dalam $F[x]$, *field* K adalah *splitting field* untuk f jika K adalah *field extension* terkecil dari F dimana f *splits* dalam K.

Teorema 51 *Untuk suatu field F dan polynomial non-konstan f dalam $F[x]$, terdapat splitting field untuk f yang unik (up to isomorphism).*

Splitting field untuk f unik *up to isomorphism* yang berarti jika K dan K' keduanya merupakan *splitting field* untuk f, maka K *isomorphic* dengan K'. Untuk membuktikan teorema 51, kita gunakan induksi pada *degree* dari f. Jika *degree* dari f adalah 1, maka $f = ax + b$ dimana $a, b \in F$, jadi $K = F(-b/a) = F$ karena $-b/a \in F$. Sekarang kita harus buktikan bahwa jika teorema 51 berlaku untuk *degree* n maka teorema juga berlaku untuk *degree* $n+1$. Jika f mempunyai *degree* $n+1$, maka terdapat faktor *irreducible* f_0 dari f dan teorema 26 mengatakan bahwa $K = F[x]/f_0F[x]$ adalah suatu *field*. Jadi terdapat $\theta_0 \in K$ yang merupakan akar dari f_0, oleh karena itu kita dapat uraikan $f = g \cdot h$ dimana $g = x - \theta_0$. *Degree* dari h adalah n, jadi dengan menggunakan hipotesis induksi terdapat *splitting field* unik L untuk h yang merupakan *field extension* dari K. Karena h dapat diuraikan dalam $L[x]$ sebagai berikut:

$$h = a_{n+1} \prod_{i=1}^{k} (x - \theta_i)^{e_i},$$

maka

$$L = K(\theta_1, \theta_2, \dots, \theta_k)$$

dan f dapat diuraikan sebagai berikut:

$$f = a_{n+1}(x - \theta_0) \prod_{i=1}^{k} (x - \theta_i)^{e_i},$$

jadi $f \in L[x]$. Karena

$$K = F(\theta_0, \theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_m})$$

dimana $\{\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_m}\} \subseteq \{\theta_1, \theta_2, \ldots, \theta_k\}$, maka

$$L = F(\theta_0, \theta_1, \theta_2, \ldots, \theta_k)$$

merupakan *splitting field* untuk f. Karena *field extension* unik (lihat teorema 48) maka *splitting field* juga unik. Selesailah pembuktian teorema 51.

Sekarang kita tiba pada hasil terpenting mengenai *finite field* yaitu teorema mengenai struktur dari *finite field*.

Teorema 52

- *Banyaknya elemen dalam suatu finite field adalah p^n dimana p adalah characteristic dari field berupa suatu bilangan prima dan n adalah bilangan bulat positif.*
- *Untuk setiap bilangan prima p dan bilangan bulat positif n terdapat suatu finite field dengan p^n elemen.*
- *Setiap finite field isomorphic dengan finite field yang mempunyai jumlah elemen yang sama.*

Untuk menunjukkan bahwa banyaknya elemen dalam suatu *finite field* F mempunyai bentuk p^n dimana p adalah bilangan prima dan n adalah bilangan bulat positif, kita mengetahui bahwa *characteristic* dari suatu *finite field* adalah suatu bilangan prima. Jadi p merupakan *characteristic* dari F dan elemen-elemen

$$0, 1, 2, \ldots, p-1$$

dimana $2 = 1+1$, $3 = 1+1+1$ dan seterusnya, membuat suatu *subfield* dari F yang *isomorphic* dengan Z/pZ. F merupakan ruang vektor atas *subfield* diatas dan dimensi dari ruang vektor adalah n. Karena ada p kemungkinan untuk setiap koordinat, maka terdapat p^n elemen secara keseluruhan.

Untuk menunjukkan bahwa terdapat *finite field* dengan p^n elemen untuk setiap bilangan prima p dan bilangan positif n, kita gunakan *polynomial*

$$f = x^q - x$$

dimana $q = p^n$. Kita dapat membuat suatu F yang mempunyai Z/pZ sebagai subfield dan merupakan *splitting field* untuk f atas Z/pZ. Jadi f dapat diuraikan sebagai berikut:

$$f = (x - r_1)(x - r_2)\ldots(x - r_q)$$

dimana $r_i \in F$ untuk $i = 1, 2, \ldots, q$. Setiap akar r_i berbeda ($r_i \neq r_j$ jika $i \neq j$) karena derivatif dari f:

$$qx^{q-1} - 1 \equiv -1 \pmod{p}$$

tidak mempunyai akar yang sama dengan akar dari f. Kita kumpulkan akar-akar tersebut dalam suatu himpunan R:

$$R = \{r_1, r_2, \ldots, r_q\}.$$

Kita ingin tunjukkan bahwa R adalah suatu *field*. 0 dan 1 merupakan elemen dari R karena 0 dan 1 keduanya merupakan akar dari f. Untuk $+$, kita dapatkan:

$$\begin{aligned}(a+b)^q &= a^q + \ldots + b^q \\ &\equiv a + b \pmod{p}\end{aligned}$$

karena semua suku kecuali a^q dan b^q mempunyai koefisien yang dapat dibagi oleh p jadi $\equiv 0 \pmod{p}$ (koefisien binomial $\begin{pmatrix} q \\ i \end{pmatrix}$ dapat dibagi oleh p untuk $0 < i < q$ dan $= 1$ untuk $i = 0$ dan $i = q$). Jadi jika $a, b \in R$, karena $(a+b)$ merupakan akar dari f, maka $(a+b) \in R$. Untuk perkalian kita dapatkan:

$$\begin{aligned}(ab)^q &= a^q b^q \\ &= ab,\end{aligned}$$

jadi jika $a, b \in R$, karena ab merupakan akar dari f, maka $ab \in R$. Untuk *inverse* kita dapatkan:

$$\begin{aligned}(a^{-1})^q &= (a^q)^{-1} \\ &= a^{-1},\end{aligned}$$

jadi jika $a \in R$, karena a^{-1} merupakan akar dari f, maka $a^{-1} \in R$. Jadi R adalah *field* dengan p^n elemen.

Untuk menunjukkan bahwa semua *finite field* dengan jumlah elemen yang sama *isomorphic*, kita umpamakan K_1 dan K_2 keduanya adalah *finite field* dengan p^n elemen. Jadi K_1 dan K_2 keduanya merupakan *field extension* dari Z/pZ dan masing-masing memiliki p^n elemen. Setiap elemen dalam K_1 dan K_2 harus mematuhi persamaan $x^q = x$ dimana $q = p^n$, jadi karena keduanya merupakan *splitting field* dari $x^q - x$ maka K_1 dan K_2 *isomorphic*. Selesailah pembuktian teorema 52.

Teorema 52 menunjukkan bahwa dua *finite field* dengan jumlah elemen yang sama adalah *isomorphic*. Dari segi teori abstrak mengenai *finite field*, dua *finite field* yang *isomorphic* dianggap sama, hanya implementasinya saja yang dapat berbeda. *Finite field* disebut juga *Galois field* (**GF**) atas nama Évariste Galois yang sangat berjasa dalam pengembangan teori *finite field*. Suatu *finite field* dengan p^n elemen diberi notasi $\mathbf{GF}(p^n)$. Notasi $\mathbf{F}_p$ juga digunakan untuk $\mathbf{GF}(p)$ dan $\mathbf{F}_q$ untuk $\mathbf{GF}(q)$ dimana $q = p^n$.

10.8 Ringkasan

Dalam bab ini kita telah kembangkan lebih lanjut matematika yang diperlukan untuk *public key cryptography*, antara lain *Fermat's Little Theorem*, *Chinese Remainder Theorem*, Fungsi Euler, dan teori mengenai *finite field* (setelah terlebih dahulu menjelaskan *group of units*, *homomorphism theorem* dan *field extension*).

Fermat's Little Theorem merupakan dasar dari banyak konsep dalam *public key cryptography*. Fungsi Euler memungkinkan *Fermat's Little Theorem* digeneralisasi untuk modulus non-prima, dengan menggunakan *Chinese Remainder Theorem* untuk menjelaskannya. *Chinese Remainder Theorem* juga akan digunakan dalam test bilangan prima. Teori mengenai *finite field* tentunya sangat penting karena *public key cryptography* mengandalkan struktur aljabar *finite field*.

Bab 11

Matematika IV - Kuadrat

Di bab ini kita akan bahas konsep *quadratic residue* dan akar kuadrat modulo bilangan ganjil.

11.1 Quadratic Residue

Kita akan bahas *quadratic residue* yang kerap digunakan dalam test bilangan prima dan dalam beberapa teknik penguraian. Akan tetapi, sebelum kita bahas *quadratic residue*, kita perkenalkan dahulu konsep akar dari 1 (*root of unity*), yaitu solusi persamaan $x^n = 1$ dalam suatu *finite field*, dan konsep akar primitif (*primitive root*) , yaitu akar yang jika dipangkatkan mengunjungi semua akar-akar persamaan yang sama dalam suatu *finite field*, dengan kata lain, semua akar-akar persamaan merupakan pemangkatan dari *primitive root*.

Definisi 25 (Root of Unity) *Untuk suatu finite field* $\mathbf{F}$*:*

- *elemen* a *adalah* n*-th root of unity jika* $a^n = 1$ *dalam* $\mathbf{F}$*;*
- *elemen* a *adalah primitive* n*-th root of unity jika* $a^n = 1$ *dalam* $\mathbf{F}$ *dan untuk setiap elemen* b *dengan* $b^n = 1$ *terdapat suatu* j *dimana* $b = a^j$ *dalam* $\mathbf{F}$*.*

Contoh dari *primitive root* adalah *generator* g untuk $\mathbf{F}_q^*$ dimana g merupakan akar dari persamaan $x^{\phi(q)} = 1$ dalam $\mathbf{F}_q$ dan setiap akar dari persamaan (jadi setiap elemen dari $\mathbf{F}_q^*$) merupakan pemangkatan dari g. Teorema berikut menjawab pertanyaan ada berapa solusi persamaan $x^n = 1$ dalam suatu *finite field* $\mathbf{F}_q$ (banyaknya *n-th root of unity*).

Teorema 53 *Jika* g *adalah generator untuk* $\mathbf{F}_q^*$*, maka* g^j *merupakan* n*-th root of unity (solusi persamaan* $x^n = 1$*) jika dan hanya jika* $nj \equiv 0 \pmod{\phi(q)}$*.*

Banyaknya n-th root of unity adalah $\gcd(n, \phi(q))$ *dan* $\mathbf{F}_q$ *mempunyai primitive n-th root of unity jika dan hanya jika* $n|\phi(q)$. *Jika* ξ *merupakan primitive n-th root of unity maka* ξ^j *juga merupakan primitive n-th root of unity jika dan hanya jika* $\gcd(j, n) = 1$.

Mari kita buktikan teorema 53. Setiap elemen dari $\mathbf{F}_q^*$ merupakan pemangkatan g^j dari *generator* g, dan pemangkatan g^{nj} menghasilkan 1 jika dan hanya jika $\phi(q)$ membagi nj. Jadi elemen g^j merupakan *n-th root of unity* jika dan hanya jika $nj \equiv 0 \pmod{\phi(q)}$. Untuk menunjukkan bahwa banyaknya *n-th root of unity* adalah $d = \gcd(n, \phi(q))$, kita fokus pada persamaan $nj \equiv 0 \pmod{\phi(q)}$ yang mempunyai bentuk dasar

$$ax \equiv b \pmod{n}. \tag{11.1}$$

Jika $d = \gcd(a, n)$, maka persamaan 11.1 mempunyai solusi untuk x jika dan hanya jika $d|b$, dan solusi persamaan 11.1 sama dengan solusi untuk

$$\frac{a}{d}x \equiv \frac{b}{d} \pmod{\frac{n}{d}}. \tag{11.2}$$

Karena $\gcd(\frac{a}{d}, \frac{n}{d}) = 1$, maka $\frac{a}{d}$ mempunyai *inverse* dalam $\mathbf{Z}/\frac{n}{d}\mathbf{Z}$, dan solusi untuk x didapat dengan mengalikan bagian kiri dan kanan persamaan 11.2 dengan *inverse* tersebut. Kembali pada banyaknya *n-th root of unity*, karena $d|0$, maka kita dapat fokus pada persamaan

$$\frac{n}{d}j \equiv 0 \pmod{\frac{\phi(q)}{d}}$$

yang, karena $\gcd(\frac{n}{d}, \frac{\phi(q)}{d}) = 1$, ekuivalen dengan

$$j \equiv 0 \pmod{\frac{\phi(q)}{d}}$$

yang berarti j harus merupakan kelipatan dari $\frac{\phi(q)}{d}$. Ada d kelipatan $\frac{\phi(q)}{d}$ (mod $\phi(q)$) yaitu

$$\begin{aligned}
j &\equiv \frac{0\phi(q)}{d} \pmod{\phi(q)} \\
j &\equiv \frac{1(\phi(q))}{d} \pmod{\phi(q)} \\
j &\equiv \frac{2(\phi(q))}{d} \pmod{\phi(q)} \\
&\cdots \\
j &\equiv \frac{(d-1)(\phi(q))}{d} \pmod{\phi(q)}.
\end{aligned}$$

$\mathbf{F}_q$ mempunyai *primitive n-th root of unity* jika dan hanya jika banyaknya *n-th root of unity* adalah n, jadi $d = n$ yang berarti $n|\phi(q)$. Sekarang kita buktikan bagian terahir teorema 53. Jika $n|\phi(q)$ maka $\mathbf{F}_q$ mempunyai *primitive n-th root of unity*, satu diantaranya adalah $\xi = g^{\phi(q)/n}$. Pemangkatan $\xi^i = 1$ jika dan hanya jika $n|i$. Pemangkatan $(\xi^j)^k = 1$ jika dan hanya jika

$$kj \equiv 0 \pmod{n}. \tag{11.3}$$

Jadi ξ^j mempunyai *order* n (persamaan 11.3 tidak berlaku untuk $0 < k < n$) jika dan hanya jika $\gcd(j, n) = 1$, yang juga berarti terdapat $\phi(n)$ *primitive n-th root of unity* untuk $\mathbf{F}_q$. Selesailah pembuktian teorema 53. *Generator* g untuk suatu *cyclic group* G merupakan contoh dari *primitive root*: g merupakan solusi untuk $x^n = 1$ dimana n merupakan banyaknya elemen dalam G, dan untuk setiap elemen a dalam G (a juga merupakan solusi untuk $x^n = 1$), terdapat i dengan $0 \le i < n$ dimana $a = g^i$.

Sekarang mari kita bahas konsep *quadratic residues*. Jika p merupakan suatu bilangan prima ganjil ($p > 2$), kita kerap ingin mengetahui elemen-elemen mana dari $\{1, 2, 3, \ldots, p-1\}$ ($\mathbf{F}_p^*$) yang merupakan kuadrat. Jika $a \in \mathbf{F}_p^*$ merupakan suatu kuadrat (misalnya $b^2 = a$), maka a memiliki tidak lebih dan tidak kurang dari dua akar pangkat dua yaitu $\pm b$ (karena persamaan $x^2 - a$ mempunyai paling banyak 2 solusi dalam suatu *field*, dan jika p ganjil maka b dan $-b$ merupakan dua solusi untuk x yang berbeda). Jadi semua kuadrat dalam $\mathbf{F}_p^*$ dapat dicari dengan mengkalkulasi b^2 untuk

$$b = 1, 2, 3, \ldots, (p-1)/2$$

karena setiap dari sisa bilangan sampai dengan $p - 1$ merupakan $-b$ untuk suatu b diatas. Jadi setengah dari bilangan dalam $\mathbf{F}_p^*$ merupakan kuadrat. Sebagai contoh, untuk $\mathbf{F}_{11}$ mereka adalah $1^2 = (-1)^2 = 1$, $2^2 = (-2)^2 = 4$, $3^2 = (-3)^2 = 9$, $4^2 = (-4)^2 = 5$ dan $5^2 = (-5)^2 = 3$. Kuadrat dalam $\mathbf{F}_p^*$ dinamakan *quadratic residues* sedangkan elemen-elemen yang bukan kuadrat disebut *non-residues*. Untuk $\mathbf{F}_{11}$ *non-residues* adalah 2, 6, 7, 8 dan 10.

Jika g merupakan *generator* untuk $\mathbf{F}_p^*$, setiap kuadrat merupakan g^j untuk suatu bilangan genap j. Sebaliknya setiap g^j, dengan j suatu bilangan genap, merupakan suatu kuadrat yaitu kuadrat dari $\pm g^{j/2}$.

Sekarang kita definisikan simbol Legendre (*Legendre symbol* dengan notasi $(\frac{a}{p})$) sebagai berikut:

Definisi 26 (Legendre Symbol)

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \textit{jika } p|a; \\ 1 & \textit{jika } a \textit{ merupakan quadratic residue} \pmod{p}; \\ -1 & \textit{jika } a \textit{ merupakan nonresidue} \pmod{p}. \end{cases}$$

Jadi $(\frac{a}{p})$ dapat digunakan untuk memberi indikasi apakah suatu bilangan bulat merupakan suatu *quadratic residue* $\pmod p$. Berikut adalah teorema penting mengenai simbol Legendre.

Teorema 54

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod p.$$

Mari kita buktikan teorema 54. Jika $p|a$ maka kedua sisi dari persamaan akan sama dengan 0. Jika $p \not| a$ maka berdasarkan *Fermat's little theorem* (teorema 30) kita dapatkan

$$\begin{aligned}(a^{(p-1)/2})^2 &= a^{p-1} \\ &\equiv 1 \pmod p,\end{aligned}$$

jadi $a^{(p-1)/2} = \pm 1$. Jika g adalah suatu *generator* untuk $\mathbf{F}_p^*$ maka terdapat bilangan j dimana $a = g^j$. Kita ketahui bahwa a merupakan *quadratic residue* jika dan hanya jika j genap. Juga $a^{(p-1)/2} = g^{j(p-1)/2} = 1$ jika dan hanya jika $j(p-1)/2$ dapat dibagi oleh $(p-1)$ (karena *generator* menghasilkan 1 jika dan hanya jika dipangkatkan oleh kelipatan $(p-1)$). Jadi $j(p-1)/2$ dapat dibagi oleh $(p-1)$ jika dan hanya jika j dapat dapat dibagi 2 (j genap). Alhasil kedua sisi dari persamaan dalam teorema sama dengan 1 jika dan hanya jika j genap. Karena untuk $p \not| a$ kedua sisi persamaan menghasilkan ± 1 dan kedua sisi menghasilkan 1 jika dan hanya jika j genap, berarti kedua sisi menghasilkan -1 jika dan hanya jika j tidak genap (ganjil), jadi kedua sisi selalu sama, jadi selesailah pembuktian teorema 54. Berikut kita buktikan dahulu beberapa persamaan mengenai simbol Legendre sebelum kita bahas teorema mengenai *quadratic reciprocity*.

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right); \tag{11.4}$$

$$\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right) \text{ jika } a \equiv b \pmod p; \tag{11.5}$$

$$\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right) \text{ jika } \gcd(b,p) = 1; \tag{11.6}$$

$$\left(\frac{1}{p}\right) = 1; \tag{11.7}$$

$$\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2} \tag{11.8}$$

$$\left(\frac{2}{n}\right) = (-1)^{(p^2-1)/8}. \tag{11.9}$$

Untuk membuktikan persamaan 11.4, kita gunakan teorema 54:

$$\begin{aligned}\left(\frac{ab}{p}\right) &\equiv (ab)^{(p-1)/2} \pmod{p} \\ &= a^{(p-1)/2}b^{(p-1)/2} \\ &\equiv \left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \pmod{p}.\end{aligned}$$

Untuk membuktikan persamaan 11.5, kita gunakan definisi *quadratic residue*: jika $a \equiv b \pmod{p}$, maka a merupakan *quadratic residue* $\pmod{p}$ jika dan hanya jika b merupakan *quadratic residue* $\pmod{p}$. Untuk membuktikan persamaan 11.6, kita gunakan persamaan 11.4:

$$\begin{aligned}\left(\frac{ab^2}{p}\right) &= \left(\frac{a}{p}\right)\left(\frac{b^2}{p}\right) \\ &= \left(\frac{a}{p}\right), \text{ jika } \gcd(b,p) = 1,\end{aligned}$$

karena $\left(\frac{b^2}{p}\right) = 1$ jika $\gcd(b,p) = 1$. Persamaan 11.7 didapat karena $1^2 = 1$ dan persamaan 11.8 didapat dari teorema 54 dengan $a = -1$. Sebelum membuktikan persamaan 11.9, kita buktikan dahulu teorema yang kita beri nama *Gauss's Lemma 1.* Untuk itu kita jelaskan notasi yang digunakan. Jika p adalah bilangan prima, maka kita dapat mempartisi $(\mathbf{Z}/p\mathbf{Z})^*$ menjadi dua subset:

$$\begin{aligned}P &= \{1, 2, 3, \ldots, (p-1)/2\} \subset (\mathbf{Z}/p\mathbf{Z})^*, \\ N &= \{-1, -2, -3, \ldots, -(p-1)/2\} \subset (\mathbf{Z}/p\mathbf{Z})^*.\end{aligned}$$

Untuk setiap $a \in (\mathbf{Z}/p\mathbf{Z})^*$, kita definisikan:

$$\begin{aligned}aP &= \{ax | x \in P\} \\ &= \{a, 2a, 3a, \ldots, (p-1)a/2\}.\end{aligned}$$

Sebagai contoh, $-1P = N$.

Teorema 55 (Gauss's Lemma 1) *Jika p adalah bilangan prima ganjil, dan $a \in (\mathbf{Z}/p\mathbf{Z})^*$, maka*

$$\left(\frac{a}{p}\right) = (-1)^{\mu}$$

dimana $\mu = |aP \cap N|$.

Pangkat μ sama dengan banyaknya elemen dari aP yang juga berada dalam N ("negatif"). Mari kita buktikan teorema 55. Jika $x, y \in P$ dan $x \neq y$, maka

$ax \neq \pm ay$ dalam $(\mathbf{Z}/p\mathbf{Z})^*$ (karena jika $ax \equiv \pm ay \pmod{p}$, maka $p|a(x \mp y)$, jadi $p|(x \mp y)$, sesuatu yang tidak mungkin karena x dan y adalah elemen yang berbeda dalam $\{1, 2, 3, \ldots, (p-1)/2\}$). Jadi elemen-elemen dari aP tersebar di himpunan-himpunan berikut:

$$\{\pm 1\}, \{\pm 2\}, \ldots \{\pm (p-1)/2\}$$

dimana dua elemen dari aP tidak mungkin berada dalam satu himpunan. Karena terdapat $(p-1)/2$ himpunan dan terdapat $(p-1)/2$ elemen dalam aP, maka setiap himpunan mempunyai satu elemen dari aP, tidak lebih dan tidak kurang. Jadi

$$aP = \{\varepsilon_i i | i = 1, 2, \ldots, (p-1)/2\}$$

dimana $\varepsilon_i = 1$ jika $\varepsilon_i i \in P$ dan $\varepsilon_i = -1$ jika $\varepsilon_i i \in N$. Kita kalikan semua elemen aP dalam $(\mathbf{Z}/p\mathbf{Z})^*$ menggunakan definisi pertama aP mendapatkan:

$$a^{(p-1)/2} \cdot ((p-1)/2)!.$$

Kita juga dapat menggunakan definisi alternatif untuk mendapatkan

$$(\prod_{i=1}^{(p-1)/2} \varepsilon_i) \cdot ((p-1)/2)!.$$

Jadi

$$\begin{aligned} a^{(p-1)/2} \cdot ((p-1)/2)! &= (\prod_{i=1}^{(p-1)/2} \varepsilon_i) \cdot ((p-1)/2)! \\ &= (-1)^{\mu} \cdot ((p-1)/2)! \end{aligned}$$

dalam $(\mathbf{Z}/p\mathbf{Z})^*$, dimana $\mu = |aP \cap N|$. Jadi

$$a^{(p-1)/2} \equiv (-1)^{\mu} \pmod{p}.$$

Dengan menggunakan teorema 54 kita dapatkan

$$\left(\frac{a}{p}\right) \equiv (-1)^{\mu} \pmod{p}$$

membuktikan teorema 55. Sekarang kita gunakan teorema 55 untuk membuktikan persamaan 11.9. Dengan $a = 2$, kita dapatkan

$$\begin{aligned} aP &= 2P \\ &= \{2, 4, 6, \ldots, p-1\}. \end{aligned}$$

Untuk $p \equiv 1 \pmod 4$,

$$2P = \{2, 4, \ldots, (p-1)/2, (p+3)/2, \ldots, p-1\}$$

dimana $(p-1)/4$ elemen pertama $\{2, 4, \ldots, (p-1)/2$ berada dalam P, dan sisanya $(p-1)/4$ elemen $\{(p+3)/2, \ldots, (p-1)\}$ berada dalam N. Jadi $\mu = |2P \cap N| = (p-1)/4$, dan menurut teorema 55:

$$\begin{aligned}\left(\frac{2}{p}\right) &= (-1)^{(p-1)/4} \\ &= ((-1)^{(p-1)/4})^{(p+1)/2} \\ &= (-1)^{(p^2-1)/8}.\end{aligned}$$

Perhatikan bahwa kita telah menggunakan fakta bahwa $(p+1)/2$ adalah bilangan ganjil, jadi $(\pm 1)^{(p+1)/2} = (\pm 1)$. Untuk $p \equiv -1 \pmod 4$,

$$2P = \{2, 4, \ldots, (p-3)/2, (p+1)/2, \ldots, p-1\}$$

dimana $(p-3)/4$ elemen pertama $\{2, 4, \ldots, (p-3)/2$ berada dalam P, dan sisanya $(p+1)/4$ elemen $\{(p+1)/2, \ldots, (p-1)\}$ berada dalam N. Jadi $\mu = |2P \cap N| = (p+1)/4$, dan menurut teorema 55:

$$\begin{aligned}\left(\frac{2}{p}\right) &= (-1)^{(p+1)/4} \\ &= ((-1)^{(p+1)/4})^{(p-1)/2} \\ &= (-1)^{(p^2-1)/8}.\end{aligned}$$

Perhatikan bahwa kita telah menggunakan fakta bahwa $(p-1)/2$ adalah bilangan ganjil, jadi $(\pm 1)^{(p-1)/2} = (\pm 1)$. Kita telah membuktikan persamaan 11.9 untuk $p \equiv 1 \pmod 4$ dan $p \equiv -1 \pmod 4$. Karena untuk bilangan prima ganjil p, $p \equiv \pm 1 \pmod 4$, maka selesailah pembuktian persamaan 11.9.

Teorema 56 (Quadratic Reciprocity) *Untuk dua bilangan prima ganjil p dan q:*

$$\left(\frac{p}{q}\right) = (-1)^{(p-1)(q-1)/4} \left(\frac{q}{p}\right) = \begin{cases} -\left(\frac{q}{p}\right) & \textit{jika } p \equiv q \equiv 3 \pmod 4; \\ \left(\frac{q}{p}\right) & \textit{jika tidak.} \end{cases}$$

Untuk membuktikan teorema 56, kita bentuk *finite field* $\mathbf{F}_{p^{q-1}}$ (perhatikan bahwa $p^{q-1} \equiv 1 \pmod q$). Karena $q | p^{q-1} - 1$, maka menurut teorema 53, terdapat *primitive qth-root of unity* dalam $\mathbf{F}_{p^{q-1}}$ yang kita beri notasi ξ. Kita definisikan *Gauss sum* G:

$$G = \sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \xi^j.$$

Kita ingin tunjukkan bahwa

$$G^2 = (-1)^{(q-1)/2} q. \tag{11.10}$$

Perhatikan bahwa

$$\sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \xi^j \quad = \quad \sum_{j=1}^{q-1} \left(\frac{j}{q}\right) \xi^j,$$

karena $\left(\frac{0}{q}\right) = 0$, dan

$$\sum_{k=1}^{q-1} \left(\frac{k}{q}\right) \xi^k \quad = \quad \sum_{k=1}^{q-1} \left(\frac{-k}{q}\right) \xi^{-k},$$

karena mengganti k dengan $-k$ dalam penjumlahan tetap menjumlahkan semua suku yang sama (setiap elemen $\neq 0$ dari *finite field* dikunjungi), jadi

$$\begin{aligned}
G^2 &= \sum_{j,k=1}^{q-1} \left(\frac{j}{q}\right) \xi^j \left(\frac{-k}{q}\right) \xi^{-k} \\
&= \left(\frac{-1}{q}\right) \sum_{j=1}^{q-1} \sum_{k=1}^{q-1} \left(\frac{jk}{q}\right) \xi^{j-k} \\
&= (-1)^{(q-1)/2} \sum_{j=1}^{q-1} \sum_{k=1}^{q-1} \left(\frac{jk}{q}\right) \xi^{j-k} \text{ (menggunakan 11.8)} \\
&= (-1)^{(q-1)/2} \sum_{j=1}^{q-1} \sum_{k=1}^{q-1} \left(\frac{j^2 k}{q}\right) \xi^{j-kj} \text{ (tukar } k \text{ dengan } kj) \\
&= (-1)^{(q-1)/2} \sum_{j=1}^{q-1} \sum_{k=1}^{q-1} \left(\frac{j^2 k}{q}\right) \xi^{j(1-k)} \\
&= (-1)^{(q-1)/2} \sum_{j=1}^{q-1} \sum_{k=1}^{q-1} \left(\frac{k}{q}\right) \xi^{j(1-k)} \text{ (menggunakan 11.6)} \\
&= (-1)^{(q-1)/2} \sum_{k=1}^{q-1} \left(\frac{k}{q}\right) \sum_{j=1}^{q-1} \xi^{j(1-k)} \\
&= (-1)^{(q-1)/2} \sum_{k=1}^{q-1} \left(\frac{k}{q}\right) \sum_{j=0}^{q-1} \xi^{j(1-k)} \\
&= (-1)^{(q-1)/2} \left(\frac{1}{q}\right) \sum_{j=0}^{q-1} \xi^{j(1-1)} \text{ (hanya } k = 1 \text{ yang memberi kontribusi)}
\end{aligned}$$

$$= \quad (-1)^{(q-1)/2}q.$$

Pada langkah ketiga dari ahir, penjumlahan dengan indeks j dapat dimulai dari 0 karena hanya menambahkan

$$\sum_{k=1}^{q-1}\left(\frac{k}{q}\right) = 0$$

(banyaknya *residue* dan *non-residue* sama $\pmod q$). Pada langkah kedua dari ahir, hanya penjumlahan dengan $k = 1$ yang dihitung, karena untuk $k \neq 1$ penjumlahan terdalam ($\sum_j$) menghasilkan 0. Ini karena jika $k \neq 1$ maka ξ^{1-k} merupakan *non-trivial qth root of unity*, jadi jika setiap elemen dalam urutan

$$(\xi^{1-k})^0, (\xi^{1-k})^1, (\xi^{1-k})^2, \ldots, (\xi^{1-k})^{q-1}$$

dikalikan dengan ξ^{1-k} maka kita dapatkan elemen-elemen yang sama dengan urutan yang berbeda. Jadi

$$\xi^{1-k}\sum_{j=0}^{q-1}\xi^{(1-k)j} = \sum_{j=0}^{q-1}\xi^{(1-k)j}$$

dan

$$(\xi^{1-k} - 1)\sum_{j=0}^{q-1}\xi^{(1-k)j} = 0.$$

Karena $(\xi^{1-k} - 1) \neq 0$, maka

$$\sum_{j=0}^{q-1}\xi^{(1-k)j} = 0.$$

Kembali ke pembuktian teorema 56, kita dapatkan

$$\begin{aligned} G^p &= (G^2)^{(p-1)/2}G \\ &= ((-1)^{(q-1)/2}q)^{(p-1)/2}G \\ &= (-1)^{(p-1)(q-1)/4}q^{(p-1)/2}G \\ &= (-1)^{(p-1)(q-1)/4}\left(\frac{q}{p}\right)G \end{aligned}$$

menggunakan teorema 54 dengan $a = q$. Dengan menggunakan definisi dari G, kita juga dapatkan

$$G^p \quad = \quad (\sum_{j=0}^{q-1}\left(\frac{j}{q}\right)\xi^j)^p$$

$$
\begin{aligned}
&= \sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \xi^{jp} \\
&= \sum_{j=0}^{q-1} \left(\frac{p}{q}\right)\left(\frac{jp}{q}\right) \xi^{jp} \\
&= \left(\frac{p}{q}\right) \sum_{j=0}^{q-1} (jpq)\, \xi^{jp} \\
&= \left(\frac{p}{q}\right) G.
\end{aligned}
$$

Menggunakan hasil sebelumnya untuk G^p, kita dapatkan

$$\left(\frac{p}{q}\right) G = (-1)^{(p-1)(q-1)/4} \left(\frac{q}{p}\right) G$$

dan dengan membagi kedua sisi dengan G kita dapatkan

$$\left(\frac{p}{q}\right) = (-1)^{(p-1)(q-1)/4} \left(\frac{q}{p}\right).$$

Selesailah pembuktian teorema 56.

Simbol Legendre dapat digunakan hanya jika modulus adalah bilangan prima. Jika modulus belum tentu bilangan prima, kita harus menggunakan simbol Jacobi (*Jacobi symbol*).

Definisi 27 (Jacobi Symbol) *Jika*

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_m^{\alpha_m}$$

adalah prime factorization (unique factorization) dari n, maka simbol Jacobi didefinisikan sebagai berikut:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \dots \left(\frac{a}{p_m}\right)^{\alpha_m}.$$

Berikut adalah beberapa persamaan mengenai simbol Jacobi, dimana m, n adalah bilangan ganjil positif, dan a, b adalah bilangan bulat.

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right); \tag{11.11}$$

$$\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right); \tag{11.12}$$

$$\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right) \text{ jika } a \equiv b \pmod{n}; \tag{11.13}$$

$$\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}; \quad (11.14)$$

$$\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}; \quad (11.15)$$

$$\left(\frac{a}{n}\right)\left(\frac{n}{a}\right) = (-1)^{\frac{a-1}{2}\frac{n-1}{2}} \text{ jika } \gcd(a,n) = 1, a > 0, a \text{ ganjil}. \quad (11.16)$$

Persamaan 11.11 dapat dijelaskan menggunakan definisi simbol Jacobi dan persamaan 11.4 mengenai simbol Legendre:

$$\begin{aligned}\left(\frac{ab}{n}\right) &= \left(\frac{ab}{p_1}\right)^{\alpha_1} \cdots \left(\frac{ab}{p_m}\right)^{\alpha_m} \\ &= \left(\frac{a}{p_1}\right)^{\alpha_1}\left(\frac{b}{p_1}\right)^{\alpha_1} \cdots \left(\frac{a}{p_m}\right)^{\alpha_m}\left(\frac{b}{p_m}\right)^{\alpha_m} \\ &= \left(\frac{a}{n}\right)\left(\frac{b}{n}\right).\end{aligned}$$

Persamaan 11.12 didapat dari definisi simbol Jacobi:

$$\begin{aligned}\left(\frac{a}{mn}\right) &= \left(\frac{a}{p_1}\right)^{\alpha_1} \cdots \left(\frac{a}{p_j}\right)^{\alpha_j}\left(\frac{a}{q_1}\right)^{\beta_1} \cdots \left(\frac{a}{q_k}\right)^{\beta_k} \\ &= \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)\end{aligned}$$

dimana

$$\begin{aligned}m &= p_1^{\alpha_1} \ldots p_j^{\alpha_j} \text{ dan} \\ n &= q_1^{\beta_1} \ldots q_k^{\beta_k}.\end{aligned}$$

Persamaan 11.13 dapat dijelaskan menggunakan definisi simbol Jacobi dan persamaan 11.5 mengenai simbol Legendre:

$$\begin{aligned}\left(\frac{a}{n}\right) &= \left(\frac{a}{p_1}\right)^{\alpha_1} \cdots \left(\frac{a}{p_m}\right)^{\alpha_m} \\ &= \left(\frac{b}{p_1}\right)^{\alpha_1} \cdots \left(\frac{b}{p_m}\right)^{\alpha_m} \\ &= \left(\frac{b}{n}\right)\end{aligned}$$

dimana

$$n = p_1^{\alpha_1} \ldots p_m^{\alpha_m}.$$

Untuk membuktikan persamaan 11.14, kita tunjukkan dahulu untuk m, n bilangan ganjil,

$$\frac{m-1}{2} + \frac{n-1}{2} \equiv \frac{mn-1}{2} \pmod 2.$$

Karena m, n ganjil, maka terdapat m', n' dimana $m = 2m' + 1, n = 2n' + 1$, jadi

$$\begin{aligned}\frac{m-1}{2} + \frac{n-1}{2} &= \frac{2m'+1-1}{2} + \frac{2n'+1-1}{2} \\ &= m' + n'.\end{aligned}$$

Kita juga dapatkan

$$\begin{aligned}\frac{mn-1}{2} &= \frac{(2m'+1)(2n'+1)-1}{2} \\ &= \frac{4m'n' + 2m' + 2n' + 1 - 1}{2} \\ &= 2m'n' + m' + n'.\end{aligned}$$

Karena $m' + n' \equiv 2m'n' + m' + n' \pmod 2$, maka kita dapatkan

$$\frac{m-1}{2} + \frac{n-1}{2} \equiv \frac{mn-1}{2} \pmod 2.$$

Karena produk bilangan ganjil juga ganjil, maka kita dapatkan

$$\frac{n_1 - 1}{2} + \ldots + \frac{n_m - 1}{2} \equiv \frac{n_1 \ldots n_m - 1}{2} \pmod 2.$$

Kembali ke persamaan 11.14:

$$\begin{aligned}\left(\frac{-1}{n}\right) &= \left(\frac{-1}{p_1}\right)^{\alpha_1} \cdots \left(\frac{-1}{p_m}\right)^{\alpha_m} \\ &= ((-1)^{(p_1-1)/2})^{\alpha_1} \ldots ((-1)^{(p_m-1)/2})^{\alpha_m} \\ &= ((-1)^{\alpha_1(p_1-1)/2}) \ldots ((-1)^{\alpha_m(p_m-1)/2}) \\ &= (-1)^k\end{aligned}$$

dengan $k = \sum_{i=1}^{k} \alpha_i (p_i - 1)/2$, jadi

$$\begin{aligned}k &= \underbrace{(p_1-1)/2 + \ldots + (p_1-1)/2}_{\alpha_1 \times} + \ldots + \underbrace{(p_m-1)/2 + \ldots + (p_m-1)/2}_{\alpha_m \times} \\ &\equiv (\underbrace{p_1 \ldots p_1}_{\alpha_1 \times} \ldots \underbrace{p_m \ldots p_m}_{\alpha_m \times} - 1)/2 \pmod 2 \\ &\equiv (p_1^{\alpha_1} \ldots p_m^{\alpha_m} - 1)/2 \pmod 2 \\ &\equiv (n-1)/2 \pmod 2.\end{aligned}$$

Jadi karena $(-1)^k = (-1)^{(n-1)/2}$, kita dapatkan

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

Untuk membuktikan persamaan 11.15, kita tunjukkan dahulu untuk m, n bilangan ganjil,

$$\frac{m^2-1}{8} + \frac{n^2-1}{8} \equiv \frac{m^2n^2-1}{8} \pmod 2.$$

Karena m, n ganjil, maka terdapat m', n' dimana $m = 2m'+1, n = 2n'+1$, jadi

$$\begin{aligned}
\frac{m^2-1}{8} + \frac{n^2-1}{8} &= \frac{(2m'+1)^2-1}{8} + \frac{(2n'+1)^2-1}{8} \\
&= \frac{(4m'^2+4m'+1)-1}{8} + \frac{(4n'^2+4n'+1)-1}{8} \\
&= \frac{4m'^2+4m'}{8} + \frac{4n'^2+4n'}{8} \\
&= \frac{4m'^2+4m'+4n'^2+4n'}{8}.
\end{aligned}$$

Kita juga dapatkan

$$\begin{aligned}
\frac{m^2n^2-1}{8} &= \frac{(2m'+1)^2(2n'+1)^2-1}{8} \\
&= \frac{(4m'^2+4m'+1)(4n'^2+4n'+1)-1}{8} \\
&= 2(m'^2n'^2+m'^2n'+m'n'^2+m'n') + \frac{4m'^2+4m'+4n'^2+4n'}{8} \\
&\equiv \frac{4m'^2+4m'+4n'^2+4n'}{8} \pmod 2.
\end{aligned}$$

Jadi

$$\frac{m^2-1}{8} + \frac{n^2-1}{8} \equiv \frac{m^2n^2-1}{8} \pmod 2.$$

Karena produk bilangan ganjil juga ganjil, maka kita dapatkan

$$\frac{n_1^2-1}{8} + \ldots + \frac{n_m^2-1}{8} \equiv \frac{n_1^2 \ldots n_m^2-1}{8} \pmod 2.$$

Kembali ke persamaan 11.15:

$$\begin{aligned}
\left(\frac{2}{n}\right) &= \left(\frac{2}{p_1}\right)^{\alpha_1} \cdots \left(\frac{2}{p_m}\right)^{\alpha_m} \\
&= ((-1)^{(p_1^2-1)/8})^{\alpha_1} \ldots ((-1)^{(p_m^2-1)/8})^{\alpha_m} \\
&= ((-1)^{\alpha_1(p_1^2-1)/8}) \ldots ((-1)^{\alpha_m(p_m^2-1)/8}) \\
&= (-1)^k
\end{aligned}$$

dengan $k = \sum_{i=1}^{k} \alpha_i (p_i^2 - 1)/8$, jadi

$$\begin{aligned} k &= \underbrace{(p_1^2-1)/8 + \ldots + (p_1^2-1)/8}_{\alpha_1 \times} + \ldots + \underbrace{(p_m^2-1)/8 + \ldots + (p_m^2-1)/8}_{\alpha_m \times} \\ &\equiv (\underbrace{p_1^2 \ldots p_1^2}_{\alpha_1 \times} \ldots \underbrace{p_m^2 \ldots p_m^2}_{\alpha_m \times} - 1)/8 \pmod 2 \\ &\equiv ((p_1^{\alpha_1})^2 \ldots (p_m^{\alpha_m})^2 - 1)/8 \pmod 2 \\ &\equiv (n^2-1)/8 \pmod 2. \end{aligned}$$

Jadi karena $(-1)^k = (-1)^{(n^2-1)/8}$, kita dapatkan

$$\left(\frac{-1}{n}\right) = (-1)^{(n^2-1)/8}.$$

Selesailah pembuktian persamaan 11.15. Untuk pembuktian persamaan 11.16, kita uraikan a dan n:

$$\begin{aligned} a &= p_1 p_2 \ldots p_k \\ n &= q_1 q_2 \ldots q_l \end{aligned}$$

dimana setiap p_i dengan $1 \le i \le k$ dan q_j dengan $1 \le j \le l$ merupakan bilangan prima, jadi pangkat bilangan prima telah diuraikan. Maka

$$\begin{aligned} \left(\frac{a}{n}\right)\left(\frac{n}{a}\right) &= \prod_{i=1}^{k}\prod_{j=1}^{l}\left(\frac{p_i}{q_j}\right)\left(\frac{q_j}{p_i}\right) \\ &= (-1)^{k_1 k_2} \end{aligned}$$

dimana

$$\begin{aligned} k_1 &= \sum_{i=1}^{k} \frac{p_i - 1}{2} \\ &\equiv (a-1)/2 \pmod 2, \text{ dan} \\ k_2 &= \sum_{j=1}^{l} \frac{q_j - 1}{2} \\ &\equiv (n-1)/2 \pmod 2. \end{aligned}$$

Jadi

$$\left(\frac{a}{n}\right)\left(\frac{n}{a}\right) = (-1)^{\frac{a-1}{2}\frac{n-1}{2}}$$

dan selesailah pembuktian persamaan 11.16.

Persamaan 11.16 dapat digunakan untuk membuktikan generalisasi dari *quadratic reciprocity*:

Teorema 57 *Untuk dua bilangan positif ganjil m dan n:*

$$\left(\frac{m}{n}\right) = (-1)^{(m-1)(n-1)/4} \left(\frac{n}{m}\right).$$

Jika $\gcd(m,n) \neq 1$ maka kedua sisi dari persamaan menghasilkan 0. Jika $\gcd(m,n) = 1$ kita gunakan persamaan 11.16, dan karena $\left(\frac{m}{n}\right)$ dan $\left(\frac{n}{m}\right)$ mempunyai nilai ± 1 maka $\left(\frac{m}{n}\right) = (-1)^{(m-1)(n-1)/4} \left(\frac{n}{m}\right)$.

Kita dapat menggunakan *quadratic reciprocity* untuk menentukan dengan cepat apakah suatu bilangan bulat a merupakan kuadrat modulo suatu bilangan prima p. Sebagai contoh, kita periksa apakah 7411 merupakan suatu kuadrat modulo bilangan prima 9283. Karena $7411 \equiv 9283 \equiv 3 \pmod 4$ maka

$$\begin{aligned}
\left(\frac{7411}{9283}\right) &= -\left(\frac{9283}{7411}\right) \\
&= -\left(\frac{1872}{7411}\right) \\
&= -\left(\frac{16}{7411}\right)\left(\frac{117}{7411}\right) \\
&= -\left(\frac{117}{7411}\right) \\
&= -\left(\frac{7411}{117}\right) \\
&= -\left(\frac{40}{117}\right) \\
&= -\left(\frac{4}{117}\right)\left(\frac{2}{117}\right)\left(\frac{5}{117}\right) \\
&= -\left(\frac{2}{117}\right)\left(\frac{5}{117}\right) \\
&= \left(\frac{5}{117}\right) \\
&= \left(\frac{117}{5}\right) \\
&= \left(\frac{2}{5}\right) \\
&= -1.
\end{aligned}$$

Jadi 7411 bukan merupakan kuadrat modulo 9283.

11.2 Akar Kuadrat Modulo Bilangan Ganjil

Dengan menggunakan *quadratic reciprocity* kita dapat dengan cepat menentukan apakah suatu bilangan merupakan kuadrat modulo bilangan prima tertentu. Akan tetapi, untuk mencari akar dari kuadrat tersebut, kita tidak dapat menggunakan *quadratic reciprocity*. Kita akan bahas metode yang dapat digunakan untuk mencari akar tersebut. Jika p merupakan bilangan prima ganjil dan a merupakan suatu kuadrat modulo p, jadi

$$\left(\frac{a}{p}\right) = 1,$$

maka kita ingin dapatkan x dimana $x^2 \equiv a \pmod{p}$. Pertama, kita tulis $p-1$ dalam bentuk

$$p - 1 = 2^\alpha \cdot s,$$

dimana s adalah bilangan ganjil, jadi s didapat dengan membagi $p-1$ dengan 2 berulang kali hingga tidak dapat dibagi 2 lagi. Maka

$$r = a^{(s+1)/2} \pmod{p}$$

sudah mendekati akar dari a. Persisnya

$$\begin{aligned}(a^{-1}r^2)^{2^{\alpha-1}} &\equiv a^{s2^{\alpha-1}} \pmod{p} \\ &\equiv a^{(p-1)/2} \pmod{p} \\ &\equiv \left(\frac{a}{p}\right) \pmod{p} \\ &= 1.\end{aligned}$$

Jadi rasio r^2/a jika dipangkatkan $2^{\alpha-1}$ menghasilkan 1. Yang kita inginkan adalah rasio x^2/a sama dengan 1. Seberapa dekat nilai r dari x? Ini tergantung dari nilai p, jika $p \equiv 3 \pmod{4}$ maka $\alpha = 1$, jadi nilai r dan x sama. Jika tidak, maka langkah-langkah berikut dapat digunakan untuk mendapatkan nilai x dari nilai r.

Secara garis besar, kita harus kalikan r dengan suatu akar pangkat 2^α dari 1 untuk mendapatkan x sehingga $(x^2/a) = 1$. Kita cari akar pangkat 2^α pengali ini menggunakan akar primitif pangkat 2^α sebagai patokan. Pertama, kita cari bilangan n yang merupakan *quadratic non-residue* modulo p, jadi

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

Jika kita buat

$$b \equiv n^s \pmod{p}$$

maka b merupakan akar pangkat 2^α dari 1 yang primitif (setiap akar pangkat 2^α dari 1, termasuk juga setiap akar pangkat 2^i dari 1 dimana $0 \leq i \leq \alpha$, dapat ditulis dalam bentuk pemangkatan b). Mari kita buktikan ini. Karena

$$\begin{aligned} b^{2^\alpha} &\equiv n^{2^\alpha s} \pmod{p} \\ &\equiv n^{p-1} \pmod{p} \\ &\equiv 1 \pmod{p} \end{aligned}$$

maka jelas b merupakan akar pangkat 2^α dari 1. Untuk menunjukkan bahwa b merupakan akar primitif, kita periksa apa konsekuensinya jika b bukan akar primitif: ada pemangkatan $b^i \equiv 1 \pmod{p}$ dimana $1 < i < 2^\alpha$, jadi $i|2^\alpha$ dan i genap, dan b sendiri adalah pemangkatan genap $(2^\alpha/i)$ dari akar primitif. Tetapi ini adalah kontradiksi karena jika b adalah hasil pemangkatan genap, maka b merupakan suatu kuadrat, sedangkan

$$\left(\frac{b}{p}\right) = \left(\frac{n}{p}\right)^s = -1$$

karena s adalah bilangan ganjil dan n adalah *non-residue*. Jadi b harus merupakan akar primitif.

Jadi kita gunakan b, yang merupakan akar primitif pangkat 2^α dari 1, sebagai patokan. Pengali r untuk mendapatkan x harus merupakan pemangkatan b, kita sebut saja b^j. Kita dapat umpamakan bahwa $j < 2^{\alpha-1}$ karena $b^{2^{\alpha-1}} = -1$, jadi j dapat ditambah dengan $2^{\alpha-1}$ untuk mendapatkan akar kuadrat yang satu lagi. Berikut cara mendapatkan j dengan satu persatu mencari bit $j_0, j_1, \ldots, j_{\alpha-2}$ secara induktif.

1. Kita pangkatkan (r^2/a) dengan $2^{\alpha-2}$ modulo p. Karena kita telah buktikan bahwa kuadrat bilangan ini adalah 1, maka bilangan ini adalah ± 1. Jika bilangan ini adalah 1 maka nilai j_0 adalah 0, sedangkan jika bilangan ini adalah -1 maka nilai j_o adalah 1.

2. Jika bit $j_0, j_1, \ldots, j_{k-1}$ telah didapat, maka $(b^{j_0+j_1+\ldots+j_{k-1}}r)^2/a$ merupakan akar pangkat $2^{\alpha-k-1}$ dari 1, jadi jika kita pangkatkan bilangan ini dengan $2^{\alpha-k-2}$ kita akan dapatkan ± 1. Jika kita dapatkan 1 maka nilai j_k adalah 0, sedangkan jika kita dapatkan -1 maka nilai j_k adalah 1.

Setiap kali kita selesai dengan langkah 2 untuk suatu k, maka

$$(b^{j_0+2j_1+\ldots+2^k j_k}r)^2/a$$

adalah akar pangkat $2^{\alpha-k-2}$ dari 1, jadi kita semakin dekat dengan solusi untuk akar kuadrat, dan saat kita selesai dengan $k = \alpha - 2$, maka

$$(b^{j_0+2j_1+\ldots+2^{\alpha-2}j_{\alpha-2}}r)^2/a = 1,$$

jadi $b^j r$ merupakan akar kuadrat dari a modulo p, dimana $j = j_0 + 2j_1 + \ldots + 2^{\alpha-2} j_{\alpha-2}$.

Mari kita coba gunakan metode diatas untuk mencari akar kuadrat dari 186 modulo 401, jadi $a = 186$, $p = 401$ dan $a^{-1} \equiv 235 \pmod{401}$. Kita temukan $n = 3$ merupakan *non-residue*, dan $p - 1 = 2^4 \cdot 25$, jadi $\alpha = 4$, $s = 25$,

$$b \equiv 3^{25} \equiv 268 \pmod{401}$$

dan

$$r \equiv 186^{13} \equiv 103 \pmod{401}.$$

Jadi $r^2/a \equiv 98 \pmod{401}$ yang merupakan akar pangkat $2^{\alpha-1} = 2^3 = 8$ dari 1. Kita lakukan langkah 1: $98^4 \equiv -1 \pmod{401}$, jadi $j_0 = 1$. Menggunakan langkah 2 kita dapatkan $j_1 = 0$ dan $j_2 = 1$, jadi $j = 1 + 2 \cdot 0 + 2^2 \cdot 1 = 5$. Jadi akar kuadrat dari 186 modulo 401 adalah

$$b^5 r \equiv 268^5 \cdot 103 \equiv 304 \pmod{401}.$$

Metode diatas adalah untuk mencari akar kuadrat modulo bilangan prima. Kita kembangkan metode diatas untuk mencari akar kuadrat modulo bilangan ganjil m yang telah diuraikan sebagai berikut:

$$m = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_r^{\alpha_r}$$

dimana setiap p_i merupakan bilangan prima ganjil. Mari kita lihat bagaimana mencari solusi x untuk persamaan

$$x^2 \equiv a \pmod{m}.$$

Metode diatas dapat digunakan untuk mencari solusi x_0 untuk persamaan

$$x_0^2 \equiv a \pmod{p_i}$$

untuk setiap p_i. Selanjutnya, kita harus cari

$$x = x_0 + x_1 p_i + \cdots + x_{\alpha_i - 1} p_i^{\alpha_i - 1}$$

sehingga $x^2 \equiv a \pmod{p_i^{\alpha_i}}$. Kita gunakan induksi pada pangkat dari p_i. Untuk *base case* kita sudah dapatkan x_0. Untuk *step case*, jika kita sudah dapatkan bilangan berbasis p dengan $\alpha - 1$ *digit* $\hat{x}$ dimana $\hat{x}^2 \equiv a \pmod{p^{\alpha-1}}$, maka *digit* ke α dari

$$x = \hat{x} + x_{\alpha-1} p_i^{\alpha-1}$$

yaitu $x_{\alpha-1}$ dapat dicari, dimulai dengan menuliskan

$$\hat{x}^2 = a + b p_i^{\alpha-1}$$

untuk mendapatkan b. Jadi

$$\begin{aligned} x^2 &= (\hat{x} + x_{\alpha-1}p_i^{\alpha-1})^2 \\ &= \hat{x}^2 + 2\hat{x}x_{\alpha-1}p_i^{\alpha-1} + x_{\alpha-1}^2 p_i^{2\alpha-2} \\ &\equiv \hat{x}^2 + 2\hat{x}x_{\alpha-1}p_i^{\alpha-1} \pmod{p_i^{\alpha}} \\ &\equiv a + p_i^{\alpha-1}(b + 2x_0x_{\alpha-1}) \pmod{p_i^{\alpha}}. \end{aligned}$$

Jadi kita dapatkan $x_{\alpha-1} \equiv -(2x_0)^{-1}b \pmod{p}$. Untuk menggabungkan hasil dari setiap $p_i^{\alpha_i}$ kita dapat gunakan *Chinese Remainder Theorem*. Metode yang telah dikembangkan ini tentunya hanya dapat digunakan jika m telah diuraikan.

11.3 Ringkasan

Di bab ini kita telah bahas konsep *quadratic residue* dan metode untuk mencari akar kuadrat modulo bilangan ganjil. Konsep *quadratic residue* yaitu kuadrat modulo bilangan prima, digunakan dalam beberapa metode untuk test bilangan prima dan penguraian bilangan besar, sedangkan metode mencari akar kuadrat modulo bilangan ganjil digunakan dalam metode penguraian *quadratic sieve*.

Bab 12

Matematika V - Algebraic Number

Di bab ini kita akan bahas konsep *algebraic number*. Kita mulai dengan penjelasan konsep ruang vektor dan *module*, diikuti oleh *separable field extension*, kemudian konsep *norm* dan *trace*, dan dikulminasi dengan *algebraic number theory*.

12.1 Ruang Vektor dan Module

Konsep ruang vektor banyak dipergunakan dalam ilmu pengetahuan dan teknologi, meskipun banyak orang yang menggunakannya tanpa menyadari struktur aljabar yang terdapat didalamnya. Kita akan rumuskan struktur aljabar untuk ruang vektor dan bahas konsep *module*. Jika K adalah suatu *field*, maka suatu K-*vector space* V adalah suatu *Abelian group* dengan operasi $+$ ditambah dengan *scalar multiplication*

$$\circ : K \times V \longrightarrow V$$

dimana untuk setiap $\alpha, \beta \in K$ dan $a, b \in V$:

- $\alpha \circ (a + b) = \alpha \circ a + \alpha \circ b$,
- $(\alpha + \beta) \circ a = \alpha \circ a + \beta \circ a$,
- $(\alpha \cdot \beta) \circ a = \alpha \circ (\beta \circ a)$, dan
- $1 \circ a = a$.

Sebagai contoh, jika K merupakan suatu *field* dan kita definisikan

$$\begin{aligned}(v_1, v_2) + (w_1, w_2) &= (v_1 + v_2, w_1 + w_2),\\ \alpha \circ (v_1, v_2) &= (\alpha v_1, \alpha v_2),\end{aligned}$$

maka K^2 merupakan suatu ruang vektor (*K-vector space*). Berbagai konsep aljabar linear didefinisikan untuk ruang vektor sebagai berikut.

Definisi 28 *Jika V adalah suatu K-vector space dan B adalah subset dari V, maka*

1. *B linearly independent jika untuk setiap $n \in \mathbf{N}^+$, $v_1, v_2, \ldots, v_n \in B$ dimana setiap v_i berbeda dan $\lambda_1, \lambda_2, \ldots, \lambda_n \in K$,*

$$\sum_{i=1}^{n} \lambda_i \cdot v_i = 0 \Longrightarrow \lambda_1 = \lambda_2 = \ldots = \lambda_n = 0.$$

2. *B adalah generator untuk V jika untuk setiap $v \in V$ terdapat $n \in \mathbf{N}^+$, $v_1, v_2, \ldots, v_n \in B$ dan $\lambda_1, \lambda_2, \ldots, \lambda_n \in K$, dimana*

$$v = \sum_{i=1}^{n} \lambda_i.$$

3. *B adalah basis untuk V jika B adalah generator untuk V yang linearly independent.*

Konsep *subspace* untuk ruang vektor dapat didefinisikan sebagai berikut. Jika V adalah suatu *K-vector space* dan U adalah subset non-kosong dari V yang *closed* untuk pertambahan (jadi U adalah *subgroup* dari V) dan U *closed* untuk *scalar multiplication* (jika $\alpha \in K$ dan $a \in U$, $\alpha \circ a \in U$), maka U merupakan *subspace* dari V.

Berikutnya kita bahas konsep *module*. Konsep *module* atas suatu *ring* sangat mirip dengan konsep ruang vektor atas suatu *field* (untuk *module*, struktur aljabar *scalar* adalah *ring* sedangkan untuk ruang vektor, struktur aljabar *scalar* adalah *field*). Jika R adalah suatu *ring*, maka suatu *R-module M* adalah suatu *Abelian group* dengan operasi + ditambah dengan *scalar multiplication*

$$\circ : R \times M \longrightarrow M$$

dimana untuk setiap $\alpha, \beta \in R$ dan $a, b \in M$:

- $\alpha \circ (a + b) = \alpha \circ a + \alpha \circ b$,
- $(\alpha + \beta) \circ a = \alpha \circ a + \beta \circ a$,

- $(\alpha \cdot \beta) \circ a = \alpha \circ (\beta \circ a)$, dan
- $1 \circ a = a$.

Berikut adalah beberapa contoh dari *module*:

- Jika R adalah suatu *ring* dan I adalah suatu *ideal* dalam R maka tidak terlalu sulit untuk melihat bahwa I adalah suatu R-*module*.
- Untuk $M = R^n$ berupa produk *finite* (*finite direct product*) dari *ring* R, jika kita abaikan perkalian dalam M dan definisikan $\alpha \circ (\beta_1, \ldots, \beta_n) = (\alpha\beta_1, \ldots, \alpha\beta_n)$, maka M merupakan R-*module* yang dinamakan *free* R-*module* dengan *rank* n.

Jika M merupakan suatu R-*module*, N merupakan *additive subgroup* dari M, dan N *closed* untuk *scalar multiplication* (jika $\alpha \in R$ dan $a \in N$, $\alpha \circ a \in N$) maka N adalah *submodule* dari M. Jika B merupakan subset dari M maka terdapat *submodule* terkecil N yang mencakup B. N terdiri dari kombinasi linear

$$\sum_{i=1}^{n} \alpha_i \circ a_i$$

dimana $\alpha_i \in R$ dan $a_i \in B$. N adalah *submodule* dengan *generator* B dalam M. Seperti halnya dengan ruang vektor, konsep *linear independence* juga berlaku untuk *module*. Jika B *linearly independent* maka B merupakan basis untuk N.

12.2 Separable Field Extension

Konsep *separable extension* kita bahas karena akan diperlukan dalam pembahasan *norm* dan *trace* pada bagian 12.3.

Definisi 29 *Suatu field extension L/K disebut separable jika untuk setiap $a \in L$, $\min_K^a$ tidak memiliki akar ganda dalam L.*

Field extension yang bukan berupa *separable extension* boleh dikatakan merupakan kekecualian atau anomali. Dalam buku ini, kita hanya peduli dengan *field extension* yang *separable*. Kita akan tunjukkan bahwa setiap *algebraic field extension* terhadap *field* dengan *characteristic* 0 dan setiap *algebraic field extension* terhadap *finite field* merupakan *separable field extension*. Untuk itu, kita akan gunakan konsep *perfect field*.

Definisi 30 *Suatu field K disebut perfect field jika setiap algebraic field extension L/K merupakan suatu separable field extension.*

Kita juga akan gunakan konsep *square-free* (bebas dari kuadrat) dan derivatif dari *polynomial*.

Definisi 31 *Jika f merupakan polynomial sebagai berikut:*

$$f = \sum_{i=0}^{n} a_i x^i,$$

maka derivatif dari f adalah f' sebagai berikut:

$$f' = \sum_{i=1}^{n} i a_i x^{i-1}.$$

Definisi 32 *Suatu polynomial f disebut square-free jika tidak terdapat suatu polynomial non-konstan g dimana $g^2|f$.*

Teorema 58 *Jika K merupakan suatu field, $f \in K[x]$ dan tidak terdapat suatu polynomial non-konstan g dimana $g|f$ dan $g|f'$, maka f square-free.*

Untuk membuktikan teorema 58 mari kita lihat apa konsekuensinya jika f tidak *square-free*, jadi terdapat *polynomial* $g, h \in K[x]$ dimana g non-konstan dan $f = g^2 h$. Karena

$$f' = 2gg'h + g^2 h',$$

maka $g|f$ dan $g|f'$, suatu kontradiksi. Jadi jika tidak terdapat *polynomial* non-konstan g dimana $g|f$ dan $g|f'$ maka f *square-free*, membuktikan teorema 58. Kebalikannya (jika $f \in F[x]$ *square-free* maka tidak terdapat *polynomial* non-konstan g dimana $g|f$ dan $g|f'$) tidak selalu benar, tetapi berlaku jika *characteristic* dari F adalah 0 atau F merupakan *finite field*. Ini akan kita tunjukkan, tetapi sebelumnya kita perlu beberapa teorema mengenai f'.

Teorema 59 *Jika F adalah suatu field dengan characteristic 0 dan $f \in F[x]$ maka $f' = 0$ jika dan hanya jika f merupakan konstan ($f \in F$).*

Mari kita buktikan teorema 59. Jika f dituliskan sebagai $f = \sum_{i=0}^{n} a_i x^i$ maka f' dapat dituliskan sebagai

$$f' = \sum_{i=1}^{n} i a_i x^{i-1}$$

dan $f' = 0$ jika dan hanya jika setiap $a_i = 0$ untuk $1 \leq i \leq n$ atau dengan kata lain f merupakan konstan.

Teorema 60 *Jika F adalah suatu field dengan characteristic $p \neq 0$ dan $f \in F[x]$ maka $f' = 0$ jika dan hanya jika terdapat $g \in F[x]$ dimana $f = g(x^p)$.*

Untuk *field* dengan *characteristic* $p \neq 0$, $f' = 0$ jika dan hanya jika setiap $a_i = 0$ untuk semua i dimana $p \not| i$, jadi

$$f = \sum_{i=0}^{m'} a_{ip} x^{ip} = \sum_{i=0}^{m'} a_{ip} (x^p)^i,$$

atau, dengan $g = \sum_{i=0}^{m'} a_{ip} x^i$,

$$f = g(x^p),$$

membuktikan teorema 60.

Teorema 61 *Jika F merupakan finite field dengan characteristic $p \neq 0$ dan $f \in F[x]$ maka $f' = 0$ jika dan hanya jika terdapat $g \in F[x]$ dimana $f = g^p$.*

Mari kita buktikan teorema 61. Jika $f = g^p$, maka

$$\begin{aligned} f' &= p \cdot g' g^{p-1} \\ &= 0. \end{aligned}$$

Sebaliknya, jika $f' = 0$ maka menurut teorema 60 terdapat $h \in F[x]$ dimana $f = h(x^p)$. Jika $h = \sum_{i=0}^{m} a_i x^i$, kita dapat tuliskan f sebagai

$$f = \sum_{i=0}^{m} a_i (x^p)^i.$$

Karena setiap elemen dari F mempunyai akar pangkat p, maka setiap a_i dapat ditulis sebagai $a_i = b_i^p$. Jadi

$$\begin{aligned} f &= \sum_{i=0}^{m} b_i^p (x^i)^p \\ &= \left(\sum_{i=0}^{m} b_i x^i \right)^p. \end{aligned}$$

Jadi dengan $g = \sum_{i=0}^{m} b_i x^i$ kita dapatkan $f = g^p$. Selesailah pembuktian teorema 61.

Teorema 62 *Jika F merupakan suatu field dengan characteristic 0 atau F merupakan suatu finite field dengan characteristic $p \neq 0$, dan $f \in F[x]$ square-free, maka tidak terdapat suatu polynomial non-konstan $g \in F[x]$ dimana $g|f$ dan $g|f'$.*

Untuk membuktikan teorema 62 mari kita lihat apa konsekuensinya jika terdapat suatu *polynomial* non-konstan $g \in F[x]$ dimana $g|f$ dan $g|f'$. Berarti terdapat suatu *polynomial* non-konstan yang *irreducible* $h \in F[x]$ dimana $h|f$ dan $h|f'$. Jadi terdapat *polynomial* $e \in F[x]$ dimana $f = he$ dan

$$f' = he' + h'e.$$

Agar $h|f'$ maka h harus membagi $h'e$. Ini bisa saja terjadi jika $h' = 0$. Tetapi jika *characteristic* dari F adalah 0 ini hanya bisa terjadi jika h adalah suatu konstan (lihat teorema 59), suatu kontradiksi. Jika F merupakan suatu *finite field* dengan *characteristic* $p \neq 0$ maka ini hanya bisa terjadi jika terdapat suatu $g \in F[x]$ dimana $f = g^p$ (lihat teorema 61), yang berarti f tidak *square-free*, lagi suatu kontradiksi. Kita tinggal periksa kemungkinan lain yang dapat membuat $h|h'e$. Karena h *irreducible* yang berarti h adalah prima, $h|h'e$ jika $h|h'$ atau $h|e$. Tidak mungkin h membagi h' karena h' mempunyai *degree* yang lebih kecil dari h. Jika $h|e$ maka terdapat suatu $d \in F[x]$ dimana $e = hd$ yang membuat $f = h^2d$, jadi f tidak *square-free*, lagi suatu kontradiksi. Karena semua kemungkinan yang membuat $h|f'$ menimbulkan kontradiksi, maka konklusinya tidak terdapat suatu *polynomial* non-konstan $g \in F[x]$ dimana $g|f$ dan $g|f'$. Selesailah pembuktian teorema 62.

Teorema 63 *Jika F merupakan suatu field dengan characteristic 0 atau F merupakan suatu finite field dengan characteristic $p \neq 0$, maka $f \in F[x]$ square-free, jika dan hanya jika tidak terdapat suatu polynomial non-konstan $g \in F[x]$ dimana $g|f$ dan $g|f'$.*

Teorema 63 adalah konsekuensi teorema 58 dan teorema 62.

Teorema 64 *Jika F merupakan suatu field dan $f \in F[x]$, maka tiga proposisi berikut equivalen:*

1. *Tidak terdapat non-konstan $g \in F[x]$ dimana $g|f$ dan $g|f'$.*
2. *Untuk setiap extension field L/F, f square-free dalam $L[x]$.*
3. *Untuk setiap extension field L/F, f tidak memiliki akar ganda dalam L.*

Untuk membuktikan bahwa proposisi 2 adalah konsekuensi proposisi 1, kita gunakan fakta bahwa jika tidak terdapat non-konstan $g \in F[x]$ dimana $g|f$ dan $g|f'$ maka tidak terdapat non-konstan $h \in L[x]$ dimana $h|f$ dan $h|f'$. Ini karena jika terdapat non-konstan $h \in L[x]$ dimana $h|f$ dan $h|f'$ maka algoritma Euclid dapat digunakan untuk mendapatkan $h \in F[x]$ dimana $h|f$ dan $h|f'$ (algoritma Euclid hanya menggunakan pertambahan, perkalian dan pembagian koefisien dari f dan f' jadi h tidak tergantung apakah sebagai *polynomial* dalam $F[x]$ atau dalam $L[x]$). Ini tentunya adalah suatu kontradiksi, jadi tidak terdapat $h \in L[x]$ dimana $h|f$ dan $h|f'$. Menggunakan teorema 58 kita dapatkan f

square-free dalam $L[x]$. Untuk membuktikan bahwa proposisi 3 adalah konsekuensi dari proposisi 2, jika f memiliki akar ganda dalam L maka f tidak *square-free* dalam $L[x]$, suatu kontradiksi. Untuk menunjukkan bahwa proposisi 1 adalah konsekuensi dari proposisi 3, mari kita lihat apa konsekuensinya jika terdapat suatu *polynomial* non-konstan $g \in F[x]$ dimana $g|f$ dan $g|f'$. Berarti terdapat suatu *polynomial* non-konstan yang *irreducible* $h \in F[x]$ dimana $h|f$ dan $h|f'$. Jadi terdapat *polynomial* $e \in F[x]$ dimana $f = he$ dan

$$f' = he' + h'e.$$

Jadi $h|h'e$. Karena h *irreducible* yang berarti h prima, $h|h'$ atau $h|e$. Jika $h|e$ maka $h^2|f$ yang berarti f mempunyai akar ganda dalam suatu L dimana L/F merupakan *field extension*, suatu kontradiksi. Jika $h|h'$ maka $h' = 0$ karena jika h' mempunyai *degree* lebih kecil dari h maka tidak mungkin $h|h'$, jadi $h' = 0$. Karena h bukan konstan, maka ini hanya bisa terjadi jika *characteristic* dari F adalah $p \neq 0$, dan h dapat ditulis sebagai

$$h = \sum_{i=0}^{m} a_i x^{ip}.$$

Karena setiap elemen dari F mempunyai akar pangkat p, maka setiap a_i dapat ditulis sebagai $a_i = b_i^p$. Jadi

$$\begin{aligned} h &= \sum_{i=0}^{m} b_i^p (x^i)^p \\ &= \left(\sum_{i=0}^{m} b_i x^i \right)^p. \end{aligned}$$

Jadi f mempunyai akar ganda (ada p akar dari f yang mempunyai nilai yang sama), suatu kontradiksi. Selesailah pembuktian teorema 64.

Teorema 65 *Jika F merupakan suatu field dan untuk setiap $f \in F[x]$, f square-free jika dan hanya jika tidak terdapat suatu $g \in F[x]$ dimana $g|f$ dan $g|f'$, maka F merupakan suatu perfect field.*

Mari kita buktikan teorema 65. Jika F bukan merupakan *perfect field* maka terdapat suatu *algebraic field extension* L/F yang bukan merupakan *separable field extension*. Jadi terdapat suatu *irreducible polynomial* (berarti *square-free*) $f \in F[x]$ yang mempunyai akar ganda dalam L. Berdasarkan teorema 64, terdapat suatu non-konstan $g \in F[x]$ dimana $g|f$ dan $g|f'$. Tetapi ini bertentangan dengan asumsi bahwa f *square-free*. Selesailah pembuktian kita.

Teorema 66 *Setiap field dengan characteristic* 0 *dan setiap finite field merupakan perfect field.*

Teorema ini adalah hasil kombinasi teorema 65 dengan teorema 63.

12.3 Norm, Trace

Kita akan bahas konsep *norm* dan *trace* untuk *separable field extension*. Kita asumsikan di bagian ini bahwa setiap *field extension* merupakan *separable field extension* terhadap suatu *perfect field*. Kita mulai dengan teorema berikut.

Teorema 67 *Jika K adalah suatu perfect field, F merupakan algebraic closure dari K, $f(x)$ merupakan monic irreducible polynomial dalam $K[x]$ dengan degree d, dan $a \in F$ merupakan akar dari $f(x)$, maka terdapat tidak lebih dan tidak kurang dari d ring homomorphism yang injective dari field $K(a)$ ke field F dengan rumus*

- $\sigma_i(r) = r$ *untuk* $r \in K$, *dan*
- $\sigma_i(a) = a_i$

dimana $1 \leq i \leq d$ *dan* $f(x)$ *dapat diuraikan dalam* $F[x]$ *sebagai berikut:*

$$f(x) = (x - a_1)(x - a_2) \cdots (x - a_d).$$

Mari kita buktikan teorema 67. Setiap pemetaan $\sigma_i : K(a) \longrightarrow K(a_i)$ merupakan *field isomorphism*, jadi setiap σ_i menentukan *isomorphic copy* dari $K(a)$ yang berbeda dalam F (karena K merupakan *perfect field*, jadi f tidak memiliki akar ganda dalam F). Jadi sedikitnya terdapat d *injective ring homomorphisms* (atau *embeddings*) dari $K(a)$ ke F. Untuk menunjukkan bahwa hanya terdapat d *embeddings* yang telah disebutkan diatas, jika $\sigma : K(a) \longrightarrow F$ merupakan *injective ring homomorphism*, maka $\sigma(r) = r$ untuk $r \in K$ dan $\sigma(a) = \theta \in F$. Karena

$$f(x) = x^d + c_{d-1}x^{d-1} + \ldots + c_1 x + c_0,$$

maka

$$\begin{aligned} f(\theta) &= \theta^d + c_{d-1}\theta^{d-1} + \ldots + c_1\theta + c_0 \\ &= \sigma(a)^d + c_{d-1}\sigma(a)^{d-1} + \ldots + c_1\sigma(a) + c_0 \\ &= \sigma(a^d + c_{d-1}a^{d-1} + \ldots + c_1 a + c_0) \\ &= \sigma(0) \\ &= 0. \end{aligned}$$

Jadi $\theta = a_i$ dan $\sigma = \sigma_i$ untuk suatu i dengan $1 \leq i \leq d$. Jadi *embedding* harus salah satu dari σ_i dan ada tidak lebih dan tidak kurang dari d *embeddings*. Selesailah pembuktian teorema 67.

Teorema 68 (Dedekind) $\sigma_1, \sigma_2, \ldots, \sigma_d$ *diatas linearly independent, dengan kata lain jika terdapat* $c_1, c_2, \ldots, c_d \in K$ *dimana*

$$c_1\sigma_1(b) + c_2\sigma_2(b) + \ldots + c_d\sigma_d(b) = 0$$

untuk setiap $b \in K(a)$, *maka setiap* $c_i = 0$ *untuk* $1 \leq i \leq d$.

Kita buktikan teorema 68 menggunakan induksi pada d. Untuk $d = 1$ maka sangat jelas bahwa jika

$$c_1\sigma_1(b) = 0$$

untuk setiap $b \in K(a)$, maka $c_1 = 0$ karena ada $b_0 \in K(a)$ dimana $\sigma_1(b_0) \neq 0$. Untuk $d \geq 2$, kita dapat asumsikan setiap $d-1$ *homomorphism* σ_i yang berbeda adalah *linearly independent*. Kita lihat apa konsekuensinya jika terdapat $c_1, c_2, \ldots, c_d$ dimana untuk setiap $b \in K(a)$:

$$c_1\sigma_1(b) + c_2\sigma_2(b) + \ldots + c_d\sigma_d(b) = 0. \tag{12.1}$$

Karena $\sigma_1 \neq \sigma_d$ maka terdapat $b_0 \in K(a)$ dimana $\sigma_1(b_0) \neq \sigma_d(b_0)$. Karena persamaan 12.1 berlaku untuk setiap $b \in K(a)$ maka persamaan juga berlaku untuk $b_0 b$. Jadi kita dapatkan

$$c_1\sigma_1(b_0)\sigma_1(b) + c_2\sigma_2(b_0)\sigma_2(b) + \ldots + c_d\sigma_d(b_0)\sigma_d(b) = 0. \tag{12.2}$$

Jika kita kalikan persamaan 12.1 dengan $\sigma_d(b_0)$ maka kita dapatkan

$$c_1\sigma_d(b_0)\sigma_1(b) + c_2\sigma_d(b_0)\sigma_2(b) + \ldots + c_d\sigma_d(b_0)\sigma_d(b) = 0. \tag{12.3}$$

Jika kita kurangkan persamaan 12.3 dari persamaan 12.2 maka kita dapatkan

$$c_1(\sigma_1(b_0) - \sigma_d(b_0))\sigma_1(b) + \ldots + c_{d-1}(\sigma_{d-1}(b_0) - \sigma_{d-1}(b_0))\sigma_{d-1}(b) = 0.$$

Dengan $e_i = c_i(\sigma_i(b_0) - \sigma_d(b_0))$ untuk $1 \leq i \leq d-1$, kita dapatkan

$$e_1\sigma_1(b) + e_2\sigma_2(b) + \ldots + e_{d-1}\sigma_{d-1}(b) = 0.$$

Berdasarkan hipotesis induksi, $\sigma_1, \sigma_2, \ldots, \sigma_{d-1}$ *linearly independent*, jadi setiap $e_i = 0$. Jadi $c_1(\sigma_1(b_0) - \sigma_d(b_0)) = 0$, dan karena $\sigma_1(b_0) \neq \sigma_d(b_0)$ maka $c_1 = 0$. Menggunakan cara yang sama kita akan dapatkan $c_2 = 0, \ldots, c_{d-1} = 0$. Persamaan 12.1 menjadi $c_d\sigma_d(b) = 0$ untuk setiap $b \in K(a)$, dan karena terdapat $e_0 \in K(a)$ dimana $\sigma_d(e_0) \neq 0$, maka $c_d = 0$. Selesailah pembuktian teorema 68.

Sekarang kita definisikan konsep *norm*:

Definisi 33 *Jika $f(x)$ merupakan monic irreducible polynomial dalam $K[x]$ dengan degree d, $a \in F$ merupakan akar dari $f(x)$, dan $\theta \in K(a)$, maka norm dari elemen θ untuk field extension $K(a)/K$, yang diberi notasi $N_K^{K(a)}(\theta)$, didefinisikan sebagai berikut:*

$$N_K^{K(a)}(\theta) = \sigma_1(\theta)\sigma_2(\theta)\cdots\sigma_d(\theta)$$

dimana setiap σ_i merupakan embedding yang berbeda seperti yang berada dalam teorema 67.

Tidak terlalu sulit untuk menunjukkan bahwa $N_K^{K(a)}$ bersifat *multiplicative*:

$$N_K^{K(a)}(\theta_1\theta_2) = N_K^{K(a)}(\theta_1)N_K^{K(a)}(\theta_2).$$

Berikutnya kita definisikan konsep *trace*:

Definisi 34 *Jika $f(x)$ merupakan monic irreducible polynomial dalam $K[x]$ dengan degree d, $a \in F$ merupakan akar dari $f(x)$, dan $\theta \in K(a)$, maka trace dari elemen θ untuk field extension $K(a)/K$, yang diberi notasi $T_K^{K(a)}(\theta)$, didefinisikan sebagai berikut:*

$$T_K^{K(a)}(\theta) = \sigma_1(\theta) + \sigma_2(\theta) + \ldots + \sigma_d(\theta)$$

dimana setiap σ_i merupakan embedding yang berbeda seperti yang berada dalam teorema 67.

Tidak terlalu sulit untuk menunjukkan bahwa $T_K^{K(a)}$ bersifat *additive*, jika $a, b \in K$ dan $x, y \in K(a)$, maka:

$$T_K^{K(a)}(ax + by) = aT_K^{K(a)}(x) + bT_K^{K(a)}(y).$$

Selanjutnya kita bahas efek komposisi *field extension* terhadap *norm*. Jika $x = N_K^L(u)$ dan E/L adalah *field extension* dengan dimensi n, maka

$$N_K^E(u) = x^n.$$

Ini karena *field extension* menghasilkan n pemetaan $\sigma LE_1, \sigma LE_2, \ldots \sigma LE_n$ yang *injective* dan setiap pemetaan menghasilkan

$$\sigma LE_i(x) = x.$$

Jika *field extension L/K* mempunyai dimensi m make terdapat mn pemetaan *injective* dari K ke E yang merupakan komposisi pemetaan

$$K \overset{\sigma KL_j}{\longrightarrow} L \overset{\sigma LE_i}{\longrightarrow} E$$

dimana $1 \leq i \leq n$ dan $1 \leq j \leq m$. Jadi

$$\begin{aligned} N_K^E(u) &= \prod_{i=1}^{n} \sigma LE_i(\prod_{j=1}^{m} \sigma KL_j(u)) \\ &= \prod_{i=1}^{n} \sigma LE_i(x) \\ &= x^n. \end{aligned}$$

Untuk *trace*, jika $x = T_K^L(u)$, rumusnya adalah:

$$T_K^E(u) = nx.$$

Berikutnya, kita akan lihat bahwa *norm* juga bisa didapat menggunakan determinan. Kita gunakan matrik pengali untuk elemen dalam $K(a)$. Dengan basis sebagai berikut

$$\begin{bmatrix} 1 \\ a \\ a^2 \\ \vdots \\ a^{d-1} \end{bmatrix},$$

jika v adalah vektor untuk suatu elemen $x \in K(a)$, matrik pengali A untuk suatu elemen $e \in K(a)$ adalah matrik yang jika dikalikan dengan vektor v:

$$Av = v'$$

menghasilkan vektor v' yang merepresentasikan ex. Jika basis yang digunakan adalah $b_1, b_2, \ldots, b_n$, maka setiap kolom i merepresentasikan eb_i sebagai kombinasi linear $b_1, b_2, \ldots, b_n$. Tidak terlalu sulit untuk melihat bahwa matrik pengali untuk a adalah *companion matrix* untuk $f(x) = x^d + c_{d-1}x^{d-1} + \ldots + c_1x + c_0$ sebagai berikut:

$$C(f) = \begin{bmatrix} 0 & 0 & \ldots & 0 & -c_0 \\ 1 & 0 & \ldots & 0 & -c_1 \\ 0 & 1 & \ldots & 0 & -c_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & -c_{d-1} \end{bmatrix}.$$

Kolom pertama dalam matrik merepresentasikan a, kolom kedua merepresentasikan a^2, dan seterusnya sampai dengan kolom terahir yang merepresentasikan a^d. Perhatikan bahwa kolom untuk a^d didapat dari

$$0 = a^d + c_{d-1}a^{d-1} + \ldots + c_1a + c_0,$$

jadi

$$a^d = -c_{d-1}a^{d-1} \ldots - c_1a - c_0.$$

Matrik menghasilkan determinan

$$\det(C(f)) = (-1)^{d-1}(-c_0) = (-1)^d c_0.$$

Menggunakan determinan kita dapatkan *norm*

$$N_K^{K(a)}(a) = \det(C(f)) = (-1)^d c_0.$$

Mari kita periksa apakah ini sesuai dengan *norm* yang didapatkan menggunakan definisi 33.

$$\begin{aligned} f(x) &= (x-a_1)(x-a_2)\cdots(x-a_d) \\ &= x^d + \ldots + (-1)^d(a_1a_2\cdots a_d). \end{aligned}$$

Jadi karena $(-1)^d(a_1a_2\cdots a_d) = c_0$ maka menggunakan definisi 33:

$$\begin{aligned} N_K^{K(a)}(a) &= \sigma_1(a)\sigma_2(a)\cdots\sigma_d(a) \\ &= a_1a_2\cdots a_d \\ &= (-1)^d(-1)^d(a_1a_2\cdots a_d) \\ &= (-1)^d c_0. \end{aligned}$$

Jadi *norm* yang didapatkan menggunakan determinan matrik sesuai dengan *norm* yang didapatkan menggunakan definisi 33. Demikian juga *trace* bisa didapatkan dari matrik pengali, yaitu dari penjumlahan elemen-elemen diagonal. Jadi menggunakan *companion matrix* kita dapatkan

$$T_K^{K(a)}(a) = -c_{d-1}.$$

Mari kita periksa apakah ini sesuai dengan *trace* yang didapatkan menggunakan definisi 34.

$$\begin{aligned} f(x) &= (x-a_1)(x-a_2)\cdots(x-a_d) \\ &= x^d - (a_1+a_2+\ldots+a_d)x^{d-1} + \ldots. \end{aligned}$$

Jadi karena $-(a_1+a_2+\ldots a_d) = c_{d-1}$ maka menggunakan definisi 34:

$$\begin{aligned} T_K^{K(a)}(a) &= \sigma_1(a) + \sigma_2(a) + \ldots + \sigma_d(a) \\ &= a_1 + a_2 + \ldots + a_d \\ &= -c_{d-1}. \end{aligned}$$

Jadi *trace* yang didapatkan menggunakan matrik sesuai dengan *trace* yang didapatkan menggunakan definisi 34.

Norm dan *trace* tidak tergantung pada basis yang digunakan. Jika basis lain digunakan (bukan 1, a, a^2, ...), maka terdapat matrik *change of basis* Q, dan karena

$$Q^{-1}QC(f)Q^{-1}Q = C(f)$$

maka $QC(f)Q^{-1}$ *similar* dengan $C(f)$ yang berarti $QC(f)Q^{-1}$ dan $C(f)$ mempunyai determinan yang sama dan *trace* yang sama. Jadi *norm* dan *trace* adalah *invariant* dari basis.

Mari kita periksa apakah penggunaan determinan berlaku untuk sembarang elemen $u \in K(a)$. Jika $u \in K$ maka matrik pengali adalah

$$\begin{bmatrix} u & 0 & \dots & 0 & 0 \\ 0 & u & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & u & 0 \\ 0 & 0 & \dots & 0 & u \end{bmatrix}$$

yang menghasilkan determinan u^d. Menggunakan definisi 33 kita dapatkan:

$$\begin{aligned} N_K^{K(a)}(u) &= \sigma_1(u)\sigma_2(u)\cdots\sigma_d(u) \\ &= \underbrace{uu\cdots u}_{d\times} \\ &= u^d \end{aligned}$$

jadi sesuai dengan hasil yang didapat menggunakan deteminan. Jika $u \notin K$ maka terdapat *irreducible polynomial* $g(x)$ dengan *degree* $n|d$ dimana u merupakan akar dari $g(x)$. Jika

$$g(x) = x^n + b_{n-1}x^{n-1} + \ldots + b_1 x + b_0,$$

maka matrik pengali u untuk $K(u)$ adalah

$$C(g) = \begin{bmatrix} 0 & 0 & \dots & 0 & -b_0 \\ 1 & 0 & \dots & 0 & -b_1 \\ 0 & 1 & \dots & 0 & -b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -b_{n-1} \end{bmatrix}$$

dan $\det(C(g)) = (-1)^n b_0$. Jika $K(u) = K(a)$ maka kita selesai karena

$$g = \min_K^u = \min_K^a = f,$$

jadi $n = d$, $b_0 = c_0$ dan u sama dengan a atau merupakan suatu *conjugate* dari a atas f. Jika $K(u) \subset K(a)$ maka $N_K^{K(u)}(u) = \det(C(g)) = (-1)^n b_0$ dan $K(a)/K(u)$ adalah *field extension* dengan dimensi $m = \frac{d}{n}$, jadi

$$\begin{aligned} N_K^{K(a)}(u) &= (N_K^{K(u)}(u))^m \\ &= ((-1)^n b_0)^m. \end{aligned}$$

Bagaimana dengan determinan matrik pengali u untuk $K(a)/K$? Sebagai basis kita dapat gunakan *cross product* basis $K(u)/K$ dengan basis $K(a)/K(u)$:

$$1, u, u^2, \ldots, u^d, v, uv, u^2v, \ldots, u^dv, \ldots, v^m, uv^m, u^2v^m, \ldots, u^dv^m$$

dimana $1, v, \ldots, v^m$ merupakan basis untuk $K(a)/K(u)$. Matrik pengali menjadi

$$U = \begin{bmatrix} C(g) & 0 & \ldots & 0 & 0 \\ 0 & C(g) & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & C(g) & 0 \\ 0 & 0 & \ldots & 0 & C(g) \end{bmatrix}$$

dimana terdapat m salinan dari submatrik $C(g)$ dan setiap 0 merupakan submatrik 0 yang dimensinya sama dengan $C(g)$. Kita dapatkan

$$\begin{aligned} \det(U) &= (\det(C(g)))^m \\ &= ((-1)^n b_0)^m \end{aligned}$$

sesuai dengan *norm* diatas. Jadi untuk sembarang $u \in K(a)$, $N_K^{K(a)}(u)$ bisa didapat menggunakan determinan matrik pengali untuk u.

Sekarang mari kita periksa apakah rumus untuk *trace* berlaku untuk sembarang elemen $u \in K(a)$. Jika $u \in K$ maka matrik pengali adalah

$$\begin{bmatrix} u & 0 & \ldots & 0 & 0 \\ 0 & u & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & u & 0 \\ 0 & 0 & \ldots & 0 & u \end{bmatrix}$$

yang menghasilkan *trace du*. Menggunakan definisi 34 kita dapatkan:

$$\begin{aligned} T_K^{K(a)}(u) &= \sigma_1(u) + \sigma_2(u) + \ldots + \sigma_d(u) \\ &= \underbrace{u + u + \ldots u}_{d\times} \\ &= du \end{aligned}$$

jadi sesuai dengan hasil yang didapat menggunakan *trace* matrik. Jika $u \notin K$ maka terdapat *irreducible polynomial* $g(x)$ dengan *degree* $n|d$ dimana u merupakan akar dari $g(x)$. Jika

$$g(x) = x^n + b_{n-1}x^{n-1} + \ldots + b_1 x + b_0,$$

maka matrik pengali u untuk $K(u)$ adalah

$$C(g) = \begin{bmatrix} 0 & 0 & \ldots & 0 & -b_0 \\ 1 & 0 & \ldots & 0 & -b_1 \\ 0 & 1 & \ldots & 0 & -b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & -b_{n-1} \end{bmatrix}$$

dan *trace* matrik adalah $-b_{n-1}$. Jika $K(u) = K(a)$ maka kita selesai karena

$$g = \min_K^u = \min_K^a = f,$$

jadi $n = d$, $b_{n-1} = c_{n-1}$ dan u sama dengan a atau merupakan suatu *conjugate* dari a atas f. Jika $K(u) \subset K(a)$ maka $T_K^{K(u)}(u) = trace(C(g)) = -b_{n-1}$ dan $K(a)/K(u)$ adalah *field extension* dengan dimensi $m = \frac{d}{n}$, jadi

$$\begin{aligned} T_K^{K(a)}(u) &= m(T_K^{K(u)}(u)) \\ &= -mb_{n-1}. \end{aligned}$$

Bagaimana dengan *trace* matrik pengali u untuk $K(a)/K$? Sebagai basis kita dapat gunakan *cross product* basis $K(u)/K$ dengan basis $K(a)/K(u)$:

$$1, u, u^2, \ldots, u^d, v, uv, u^2v, \ldots, u^dv, \ldots, v^m, uv^m, u^2v^m, \ldots, u^dv^m$$

dimana $1, v, \ldots, v^m$ merupakan basis untuk $K(a)/K(u)$. Matrik pengali menjadi

$$U = \begin{bmatrix} C(g) & 0 & \ldots & 0 & 0 \\ 0 & C(g) & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & C(g) & 0 \\ 0 & 0 & \ldots & 0 & C(g) \end{bmatrix}$$

dimana terdapat m salinan dari submatrik $C(g)$ dan setiap 0 merupakan submatrik 0 yang dimensinya sama dengan $C(g)$. Kita dapatkan

$$\begin{aligned} trace(U) &= m(trace(C(g))) \\ &= -mb_{n-1} \end{aligned}$$

sesuai dengan *trace* diatas. Jadi untuk sembarang $u \in K(a)$, $T_K^{K(a)}(u)$ bisa didapat menggunakan *trace* matrik pengali untuk u.

12.4 Algebraic Number Theory

Teori mengenai *algebraic numbers* diperlukan dalam pembahasan metode *number field sieve*, yaitu metode tercepat hingga saat ini untuk menguraikan bilangan sangat besar (lebih dari 100 digit). Pembahasan *algebraic number theory* biasanya melibatkan 4 komponen:

- suatu *Dedekind domain* yaitu **Z** (bilangan bulat),
- suatu *fraction field*[1] untuk **Z** yaitu **Q** (bilangan rasional),

[1] *Fraction field* untuk suatu *ring* terdiri dari semua pecahan dimana *numerator* dan *denominator* (kecuali 0 tidak dapat menjadi *denominator*) berasal dari *ring*.

- suatu *number field* $\mathbf{Q}(\alpha)$ yang merupakan *algebraic field extension* dari $\mathbf{Q}$, dan
- suatu *Dedekind domain* $\mathfrak{D}$ terdiri dari semua *algebraic integers* dalam $\mathbf{Q}(\alpha)$ (semua elemen dalam $\mathbf{Q}(\alpha)$ yang *integral* atas $\mathbf{Z}$).

Pertama kita akan bahas konsep *Dedekind domain* yaitu struktur *ring* dimana setiap *proper ideal* dapat diuraikan secara unik (*unique factorization*). Untuk itu kita perlu definisikan terlebih dahulu beberapa konsep dimulai dengan *integral closure*. Konsep *integral closure* untuk *ring* adalah generalisasi dari konsep *algebraic closure* untuk *field*

Definisi 35 *Jika A dan B keduanya merupakan ring dengan A subring dari B dan $b \in B$, maka b disebut integral atas A jika terdapat monic polynomial f dengan koefisien dalam A dimana $f(b) = 0$.*

Jadi *integral* untuk *ring* serupa dengan konsep *algebraic* untuk *field*.

Definisi 36 (Integral Closure) *Jika A dan B keduanya merupakan ring dengan $A \subseteq B$, maka subset C dari B yang berisi semua elemen B yang integral atas A merupakan subring dari B yang mencakup A dan disebut integral closure dari A dalam B. Jika $C = A$ maka A disebut integrally closed dalam B. Jika A disebut integrally closed tanpa menyebut dalam ring apa, maka yang dimaksud adalah integrally closed dalam fraction field untuk A.*

Contoh dari suatu *ring* yang *integrally closed* adalah $\mathbf{Z}$:

Teorema 69 $\mathbf{Z}$ *(himpunan bilangan bulat) adalah suatu ring yang integrally closed.*

Mari kita buktikan teorema 69. Kita tunjukkan bahwa setiap elemen dalam *fraction field* $\mathbf{Q}$ yang *integral* atas $\mathbf{Z}$ berada dalam $\mathbf{Z}$. Jika x adalah elemen sebagaimana diatas, maka x dapat dituliskan sebagai $x = \frac{a}{b}$ dimana $a, b \in \mathbf{Z}$ dan a koprima dengan b. Karena x *integral* atas $\mathbf{Z}$ maka terdapat persamaan sebagai berikut

$$(\frac{a}{b})^n + a_{n-1}(\frac{a}{b})^{n-1} + \ldots + a_1(\frac{a}{b}) + a_0 = 0$$

dimana setiap $a_i \in \mathbf{Z}$. Jika persamaan kita kalikan dengan b^n kita dapatkan

$$a^n + bc = 0$$

untuk suatu $c \in \mathbf{Z}$. Jadi b membagi a^n yang, karena a koprima dengan b, hanya bisa terjadi jika b merupakan suatu *unit*. Jika b merupakan *unit*, maka

$$x = ab^{-1} \in \mathbf{Z}.$$

Jadi $\mathbf{Z}$ *integrally closed*.

Teorema 70 *Jika $x \in \mathfrak{D}$, maka setiap*

$$\sigma_i(x) \in \mathbf{Z}$$

untuk $0 \leq i \leq d$, dimana setiap σ_i adalah homomorphism sesuai teorema 67 dengan $f = \min_{\mathbf{Q}}^{\alpha}$ dan d adalah degree dari $\min_{\mathbf{Q}}^{\alpha}$.

Mari kita buktikan teorema 70. Pertama kita ingin tunjukkan bahwa setiap

$$x_i = \sigma_i(x)$$

integral atas $\mathbf{Z}$. Karena $x \in \mathfrak{D}$ maka x *integral* atas $\mathbf{Z}$, jadi terdapat *polynomial* f dengan koefisien dalam $\mathbf{Z}$ dimana $f(x) = 0$. Kita dapatkan

$$\sigma_i(f(x)) = f(\sigma_i(x)) = f(x_i)$$

jadi setiap x_i *integral* atas $\mathbf{Z}$. Karena menurut teorema 69, $\mathbf{Z}$ *integrally closed*, maka $x_i \in \mathbf{Z}$ membuktikan teorema 70.

Konsep berikutnya yang diperlukan untuk *Dedekind domain* adalah konsep *Noetherian ring*.

Definisi 37 (Noetherian Ring) *Suatu ring R adalah Noetherian jika tidak terdapat deretan yang infinite dari ideal $I_0, I_1, I_2, \ldots$ dalam R:*

$$I_0 \subset I_1 \subset I_2 \subset \ldots$$

Jadi setiap himpunan non-kosong berisi *ideals* dari suatu *Noetherian ring* mempunyai elemen maksimal. Karena $\subset$ untuk *ideal* bersifat *partial order*, elemen maksimal tidak unik. Himpunan dapat memiliki lebih dari satu elemen maksimal. Suatu *Noetherian ring* juga mempunyai sifat bahwa setiap *ideal* dalam *ring* mempunyai *generator* yang *finite* (*finitely generated*). Artinya setiap *ideal* I dalam *Noetherian ring* R mempunyai *generator* dengan bentuk

$$A = \{a_0, a_1, \ldots, a_n\},$$

jadi setiap elemen dalam *ideal* I dapat ditulis sebagai

$$\sum_{i=0}^{n} a_i r_i$$

dimana setiap $r_i \in R$. Notasi $\mathrm{Id}(a_0, a_1 \ldots, a_n)$ kerap digunakan untuk *ideal* dengan *generator* A. Untuk menunjukkan bahwa setiap *ideal* I dalam suatu *Noetherian ring* R mempunyai *generator* yang *finite*, diperlukan penggunaan *axiom of choice*. Pembuktian dilakukan dengan menunjukkan bahwa jika *generator* tidak *finite* maka kita akan dapatkan kontradiksi. Dengan φ berupa fungsi *choice* yang jika diaplikasikan pada $0 \neq A \in \mathcal{P}(R)$ (A adalah subset

non-kosong dari R) menghasilkan suatu elemen dalam A, dan dengan $a_0 \in I$ sembarang elemen dalam I, kita definisikan

$$a_{i+1} = \varphi(I \setminus \mathrm{Id}(a_0, a_1, \ldots, a_i))$$

dan

$$I_i = \mathrm{Id}(a_0, a_1, \ldots, a_i)$$

untuk setiap $i \in \mathbf{N}$. Maka terdapat deretan *infinite*

$$I_o \subset I_1 \subset I_2 \subset \ldots$$

yang kontradiksi dengan definisi *Noetherian ring* untuk R. Sekarang kita buktikan sebaliknya, yaitu jika setiap *ideal* dalam *ring* R adalah *finitely generated*, maka R adalah *Noetherian ring*. Pertama, kita buktikan terlebih dahulu bahwa jika setiap *ideal* dalam *ring* R adalah *finitely generated*, maka untuk setiap $B \subseteq R$ terdapat *finite subset* $C \subseteq B$ dimana $\mathrm{Id}(C) = \mathrm{Id}(B)$. Karena $\mathrm{Id}(B)$ *finitely generated*, berarti terdapat *subset* $D = \{d_1, \ldots, d_n\} \subseteq \mathrm{Id}(B)$ yang *finite* dimana $\mathrm{Id}(D) = \mathrm{Id}(B)$. Kita dapat tuliskan setiap d_i sebagai:

$$d_i = \sum_{j=1}^{k_i} r_{ij} b_{ij} \text{ dengan } r_{ij} \in R, b_{ij} \in B,$$

untuk $1 \leq i \leq n$. Jika kita buat

$$C = \{b_{ij} | 1 \leq i \leq n, 1 \leq j \leq k_i\}$$

maka C adalah *finite subset* B yang menjadi *generator* untuk $\mathrm{Id}(B)$ karena untuk setiap $b \in \mathrm{Id}(B)$ terdapat $s_1, \ldots, s_n$ dimana setiap $s_i \in R$ dan

$$\begin{aligned} b &= \sum_{i=1}^{n} s_i d_i \\ &= \sum_{i=1}^{n} s_i \sum_{j=1}^{k_i} r_{ij} b_{ij} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{k_i} s_i r_{ij} b_{ij}. \end{aligned}$$

Jadi C merupakan *finite subset* B dengan $\mathrm{Id}(C) = \mathrm{Id}(B)$. Berikutnya kita akan tunjukkan bahwa jika setiap *ideal* dalam R *finitely generated*, maka untuk setiap deretan elemen dalam R

$$a_0, a_1, a_2, \ldots$$

$(\{a_i\}_{i\in\mathbf{N}})$ terdapat $n \in \mathbf{N}$ dimana $a_{n+1} \in \mathrm{Id}(a_o, a_1, \ldots, a_n)$. Dengan

$$I = \mathrm{Id}(\{a_i | i \in \mathbf{N}\})$$

hasil sebelumnya mengatakan bahwa terdapat *finite subset* $B \subset \{a_i | i \in \mathbf{N}\}$ dimana $\mathrm{Id}(B) = I$. Jadi terdapat $n \in \mathbf{N}$ dimana $B \subseteq \{a_0, \ldots, a_n\}$. Alhasil $\mathrm{Id}(a_0, \ldots, a_n) = I$, jadi

$$a_{n+1} \in \mathrm{Id}(a_0, \ldots, a_n).$$

Sekarang kita tunjukkan bahwa jika setiap *ideal* dalam R *finitely generated* maka R adalah *Noetherian ring*. Kita lihat apa konsekuensi jika R bukan *Noetherian ring*, jadi terdapat deretan *infinite*

$$I_0 \subset I_1 \subset I_2 \subset \ldots$$

Kita gunakan *axiom of choice* dengan fungsi *choice* φ untuk mendefinisikan dereten

$$a_0, a_1, a_2, \ldots$$

dengan $a_0 = \varphi(I_0)$ dan $a_{i+1} = \varphi(I_{i+1} \setminus I_i)$. Menggunakan hasil sebelumnya, terdapat $n \in \mathbf{N}$ dimana $a_{n+1} \in \mathrm{Id}(a_0, \ldots, a_n)$, suatu kontradiksi. Jadi kita telah membuktikan teorema berikut.

Teorema 71 *Suatu ring R adalah Noetherian ring jika dan hanya jika setiap ideal dalam R finitely generated.*

Sekarang kita definisikan konsep *Dedekind domain*.

Definisi 38 (Dedekind Domain) *Suatu Dedekind domain adalah suatu integral domain A dimana*

- *A integrally closed.*
- *A merupakan suatu Noetherian ring.*
- *Setiap ideal prima yang bukan 0 merupakan ideal maksimal.*

Tidak terlalu sulit untuk menunjukkan bahwa $\mathbf{Z}$ merupakan suatu *Dedekind domain*:

- Berdasarkan teorema 69, $\mathbf{Z}$ *integrally closed*.
- Berikutnya, karena $\mathbf{Z}$ merupakan *principal ideal domain* dimana setiap *ideal* I mempunyai bentuk $n\mathbf{Z}$ dengan $n \in \mathbf{Z}$, jadi I *finitely generated* oleh

  $$\{n\},$$

  maka $\mathbf{Z}$ merupakan *Noetherian ring*.

- Yang terahir, karena **Z** merupakan suatu *principal ideal domain*, maka menurut teorema 18 setiap *non-trivial ideal* prima dalam **Z** merupakan *ideal* maksimal.

Sebelum menunjukkan bahwa $\mathfrak{D}$ juga merupakan *Dedekind domain*, kita definisikan terlebih dahulu konsep *Noetherian module*.

Definisi 39 (Noetherian Module) *Suatu module M adalah Noetherian jika tidak terdapat deretan infinite submodule $M_0, M_1, M_2, \ldots$ dari M:*

$$M_0 \subset M_1 \subset M_2 \subset \ldots$$

Jika suatu *ring* R adalah *Noetherian* sebagai *module*, jelas bahwa R merupakan *Noetherian ring* karena setiap *ideal* dalam R adalah *submodule* dari R. Juga sangat jelas bahwa jika *module* M *Noetherian*, maka *submodule* G dari M juga *Noetherian*. Konsep *quotient module* M/G didefinisikan mirip dengan *quotient ring*, hanya saja *ideal* diganti oleh *submodule* sebagai modulo.

Teorema 72 *Jika M adalah suatu module dan G adalah submodule dari M, maka M Noetherian jika dan hanya jika G dan M/G Noetherian.*

Jika M *Noetherian*, sangat jelas bahwa G juga *Noetherian*, dan *submodule* dari M/G dapat ditulis sebagai

$$G_0/G,\ G_1/G,\ G_2/G,\ \ldots$$

dimana setiap G_i adalah *submodule* dari M yang mencakup G (G merupakan subset dari G_i). Karena tidak terdapat deretan *infinite*

$$G_0 \subset G_1 \subset G_2 \subset \ldots$$

maka tidak terdapat deretan *infinite*

$$G_0/G \subset G_1/G \subset G_2/G \subset \ldots$$

jadi M/G *Noetherian*. Jika G dan M/G *Noetherian*, mari kita tunjukkan bahwa M juga *Noetherian*. Jika

$$M_0 \subseteq M_1 \subseteq M_2 \subseteq \ldots$$

merupakan sembarang deretan *infinite* dimana setiap M_i merupakan *submodule* dari M, maka terdapat k_1 dimana

$$M_0 \cap G \ \subset\ \ldots\ \subset\ M_{k_1} \cap G \ =\ M_{k_1+1} \cap G \ =\ \ldots$$

dan k_2 dimana

$$(G + M_0)/G \ \subset\ \ldots\ \subset\ (G + M_{k_2})/G \ =\ (G + M_{k_2+1})/G \ =\ \ldots$$

Jika $k = \max(k_1, k_2)$, maka

$$M_k \cap G = M_{k+i} \cap G \text{ dan } G + M_k = G + M_{k+i}$$

untuk setiap $i \in \mathbf{N}$. Kita ketahui bahwa $M_k \subseteq M_{k+i}$. Jika $g \in M_{k+i}$, maka $g \in G + M_{k+i} = G + M_k$. Jadi terdapat $a \in G$ dan $b \in M_k$ dimana $g = a + b$, dan kita dapatkan

$$a = g - b \in M_{k+i} \cap G = M_k \cap G.$$

Ini menghasilkan $a, b \in M_k$, yang berarti $g = a + b \in M_k$, jadi $M_{k+i} \subseteq M_k$, dan bersama dengan $M_k \subseteq M_{k+i}$ menghasilkan $M_k = M_{k+i}$. Jadi M *Noetherian*, dan selesailah pembuktian teorema 72. Satu konsekuensi dari teorema 72 adalah *direct product* dari dua *Noetherian module* juga *Noetherian* (komponen pertama adalah G, komponen kedua adalah M/G, dan produk adalah M). *Direct product* P dari dua R-*module* M dan N didefinisikan sebagai berikut:

- $P = \{(x_1, x_2) | x_1 \in M, x_2 \in N\}$ (jadi elemen-elemen P membentuk himpunan yang merupakan *Cartesian product* dari M dan N).
- $(x_1, x_2) + (y_1, y_2) = (x_1 + x_2, y_1 + y_2)$.
- $\alpha \circ (x_1, y_1) = (\alpha \circ x_1, \alpha \circ x_2)$.

Dengan menggunakan induksi kita dapatkan teorema berikut:

Teorema 73 *Finite direct product dari Noetherian modules juga Noetherian.*

Kembali ke $\mathfrak{D}$, sifat pertama yang harus dipenuhi $\mathfrak{D}$ agar menjadi suatu *Dedekind domain* adalah bahwa $\mathfrak{D}$ *integrally closed*. Untuk itu kita perlukan dua teorema.

Teorema 74 *Terdapat basis untuk field extension* $\mathbf{Q}(\alpha)/\mathbf{Q}$ *yang seluruhnya terdiri dari elemen-elemen* $\mathfrak{D}$.

Mari kita buktikan teorema 74. Jika $x_1, x_2, \ldots, x_n$ merupakan basis untuk $\mathbf{Q}(\alpha)/\mathbf{Q}$, maka setiap x_i *algebraic* atas $\mathbf{Q}$ (karena *extension* bersifat *algebraic*), jadi terdapat *polynomial* sebagai berikut:

$$a_m x_i^m + \ldots + a_1 x_i + a_0$$

dimana $a_m \neq 0$ dan $a_j \in \mathbf{Z}$ (*polynomial* dengan koefisien dalam $\mathbf{Z}$ didapat dari *polynomial* dengan koefisien dalam $\mathbf{Q}$ dengan mengalikan *common denominator*). Dengan $y_i = a_m x_i$ kita kalikan *polynomial* dengan a_m^{m-1} untuk mendapatkan

$$\begin{aligned} (a_m x_i)^m + \ldots + a_1 a_m^{m-2}(a_m x_i) + a_0 a_m^{m-1} &= \\ y_i^m + \ldots - a_1 a_m^{m-2} y_i + a_0 a_m^{m-1} &= 0. \end{aligned}$$

Jadi terdapat basis $y_1, y_2, \ldots, y_n$ untuk $\mathbf{Q}(\alpha)/\mathbf{Q}$ dimana setiap y_i *integral* atas $\mathbf{Z}$ (dengan kata lain setiap y_i adalah elemen dari $\mathfrak{D}$), jadi terdapat basis yang seluruhnya terdiri dari elemen-elemen $\mathfrak{D}$. Selesailah pembuktian teorema 74.

Teorema 75 $\mathbf{Q}(\alpha)$ *merupakan fraction field untuk* $\mathfrak{D}$.

Jika $x \in \mathbf{Q}(\alpha)$, maka terdapat $0 \neq a \in \mathbf{Z}$ dan $y \in \mathfrak{D}$ dimana $x = \frac{y}{a}$ (gunakan pembuktian teorema 74 dengan $y_i = y, x_i = x, a_m = a$). Jadi $\mathbf{Q}(\alpha)$ merupakan *fraction field* untuk $\mathfrak{D}$.

Sifat kedua yang harus dipenuhi oleh $\mathfrak{D}$ adalah *Noetherian ring*. Untuk itu, selain konsep *module* kita gunakan juga *trace form*.

Definisi 40 (Trace Form) *Untuk separable field extension L/K, trace form untuk L/K adalah suatu bilinear form dengan pemetaan sebagai berikut:*

$$(x, y) \mapsto T_K^L(xy)$$

dimana $x, y \in L$.

Teorema 76 *Jika L/K merupakan separable field extension, maka trace form dari L/K non-degenerate. Dengan kata lain, jika $(x, y) \mapsto 0$ untuk semua $y \in L$, maka $x = 0$.*

Untuk membuktikan teorema 76, kita tunjukkan terlebih dahulu bahwa jika L/K merupakan *separable field extension* maka $T_K^L(x)$ tidak mungkin 0 untuk semua $x \in L$. Jika $T_K^L(x) = 0$ untuk semua $x \in L$, maka $\sum_{i=0}^{d} \sigma_i(x) = 0$ untuk semua $x \in L$. Tetapi ini bertentangan dengan teorema 68, jadi tidak mungkin $T_K^L(x) = 0$ untuk semua $x \in L$. Sekarang kita lihat apa konsekuensinya jika $(x, y) \mapsto 0$ untuk semua $y \in L$ dan $x \neq 0$. Kita pilih $x_0 \in L$ dimana $T_K^L(x_0) \neq 0$, lalu pilih $y \in L$ yang membuat $x_0 = xy$. Karena kita dapatkan kontradiksi, yaitu

$$T_K^L(x_0) = T_K^L(xy) = 0$$

dan

$$T_K^L(x_0) \neq 0,$$

maka selesailah pembuktian teorema 76.

Teorema 77 *Jika $b_1, b_2, \ldots, b_d$ adalah suatu basis untuk $\mathbf{Q}(\alpha)/\mathbf{Q}$ sebagai ruang vektor, maka terdapat basis $c_1, c_2, \ldots, c_d$ untuk $\mathbf{Q}(\alpha)/\mathbf{Q}$ (yang didapat menggunakan dual basis) dimana*

$$(b_i, c_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j. \end{cases}$$

Untuk membuktikan teorema 77, pertama perhatikan bahwa untuk setiap $y \in \mathbf{Q}(\alpha)$, pemetaan

$$l = f(y) : x \mapsto (x, y)$$

merupakan *linear form*, atau secara formal:

$$\begin{aligned} l(x_1 + x_2) &= l(x_1) + l(x_2), \\ l(ax_1) &= al(x_1) \end{aligned}$$

untuk setiap $x_1, x_2 \in \mathbf{Q}(\alpha)$ dan setiap $a \in \mathbf{Q}$. Ini dapat ditunjukkan sebagai berikut:

$$\begin{aligned} l(x_1 + x_2) &= \sum_{i=1}^{d} \sigma_i((x_1 + x_2)y) \\ &= \sum_{i=1}^{d} (\sigma_i(x_1 y) + \sigma_i(x_2 y)) \\ &= \sum_{i=1}^{d} \sigma_i(x_1 y) + \sum_{i=1}^{d} \sigma_i(x_2 y) \\ &= l(x_1) + l(x_2) \end{aligned}$$

dan

$$\begin{aligned} l(ax_1) &= \sum_{i=1}^{d} \sigma_i(ax_1) \\ &= \sum_{i=1}^{d} a\sigma_i(x_1) \\ &= a \sum_{i=1}^{d} \sigma_i(x_1) \\ &= al(x_1). \end{aligned}$$

Pemetaan

$$y \mapsto f(y)$$

merupakan suatu *linear map* dari $\mathbf{Q}(\alpha)$ ke $\mathbf{Q}(\alpha)^*$ (ruang untuk *linear form* pada $\mathbf{Q}(\alpha)$). Berikutnya kita tunjukkan bahwa *linear map* $y \mapsto f(y)$ *injective*, yaitu $f(y_1) \neq f(y_2)$ untuk setiap $y_1, y_2 \in \mathbf{Q}(\alpha)$ jika $y_1 \neq y_2$. Mari kita lihat apa konsekuensinya jika $f(y_1) = f(y_2)$ dan $y_1 \neq y_2$. Jadi

$$(x \mapsto \sum_{i=1}^{d} \sigma_i(xy_1)) = (x \mapsto \sum_{i=1}^{d} \sigma_i(xy_2))$$

atau

$$\sum_{i=1}^{d} \sigma_i(x(y_1 - y_2)) = 0$$

yang, karena x sembarang jadi bisa pilih $x \neq 0$, dan berdasarkan teorema 68 berarti $(y_1 - y_2) = 0$ atau $y_1 = y_2$, suatu kontradiksi. Jadi jika $y_1 \neq y_2$ maka $f(y_1) \neq f(y_2)$, membuktikan bahwa $y \mapsto f(y)$ *injective*. Kita juga ingin tunjukkan bahwa *linear map* $y \mapsto f(y)$ *surjective*: untuk setiap *linear form* l pada $\mathbf{Q}(\alpha)$, terdapat $y \in \mathbf{Q}(\alpha)$ dimana $l = f(y)$. Jika $b_1, b_2, \ldots, b_d$ merupakan basis untuk $\mathbf{Q}(\alpha)/\mathbf{Q}$ sebagai ruang vektor maka $x = x_1 b_1 + x_2 b_2 + \ldots + x_d b_d$ dapat ditulis dengan vektor kolom sebagai berikut:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}.$$

Jadi setiap *linear form* dapat ditulis dengan perkalian matrik sebagai berikut:

$$\begin{bmatrix} a_1 & a_2 & \ldots & a_d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

yang menghasilkan $a_1x_1 + a_2x_2 + \ldots a_dx_d$. Jadi kita ingin tunjukkan bahwa terdapat $y \in \mathbf{Q}(\alpha)$ dimana $T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(xy)$ juga menghasilkan $a_1x_1 + a_2x_2 + \ldots a_dx_d$. Jika $y = y_1b_1 + y_2b_2 + \ldots + y_db_d$, maka kita dapatkan

$$\begin{aligned}
T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(xy) &= \sum_{i=1}^{d} \sigma_i(xy) \\
&= \sum_{i=1}^{d} \sigma_i((x_1b_1 + \ldots + x_db_d)(y_1b_1 + \ldots + y_db_d)) \\
&= \sum_{i=1}^{d} \sigma_i(x_1y_1b_1^2) + \sum_{i=1}^{d} \sigma_i(x_1y_2b_1b_2) + \ldots \sum_{i=1}^{d} \sigma_i(x_1y_db_1b_d) + \\
&\quad \sum_{i=1}^{d} \sigma_i(x_2y_1b_1b_2) + \sum_{i=1}^{d} \sigma_i(x_2y_2b_2^2) + \ldots \sum_{i=1}^{d} \sigma_i(x_2y_db_2b_d) + \\
&\quad \vdots \\
&\quad \sum_{i=1}^{d} \sigma_i(x_dy_1b_1b_d) + \sum_{i=1}^{d} \sigma_i(x_dy_2b_2b_d) + \ldots \sum_{i=1}^{d} \sigma_i(x_dy_db_d^2)
\end{aligned}$$

$$\begin{aligned} = \quad & x_1(y_1 T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_1^2) + y_2 T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_1 b_2) + \ldots + y_d T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_1 b_d)) + \\ & x_2(y_1 T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_1 b_2) + y_2 T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_2^2) + \ldots + y_d T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_2 b_d)) + \\ & \vdots \\ & x_d(y_1 T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_1 b_d) + y_2 T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_2 b_d) + \ldots + y_d T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_d^2)). \end{aligned}$$

Agar hasil diatas sama dengan $a_1 x_1 + a_2 x_2 + \ldots a_d x_d$, untuk $1 \leq j \leq d$ kita dapatkan

$$a_j x_j = x_j(\sum_{i=1}^{d} y_i T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_j b_i))$$

atau

$$a_j = \sum_{i=1}^{d} y_i T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_j b_i).$$

Setiap a_j dan $T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_j b_i)$ merupakan konstan, dan karena ada d persamaan dengan d variabel $(y_1, y_2, \ldots, y_d)$ maka setiap y_i dapat ditemukan, jadi terdapat $y \in \mathbf{Q}(\alpha)$ dimana $l = f(y)$, jadi *linear map* $y \mapsto f(y)$ *surjective*. Karena *linear map* tersebut juga *injective* maka $y \mapsto f(y)$ adalah suatu *bijection*. Menggunakan *bijection* ini, kita dapatkan *dual basis* dari $b_1, b_2, \ldots, b_d$ dalam *dual space* (yaitu ruang untuk *linear form*):

$$z_1, z_2, \ldots, z_d.$$

Jadi $z_j(b_i) = \delta_{ij}$. Jika $z_j = f(c_j)$ maka

$$\begin{aligned} (b_i, c_j) &= f(c_j)(b_i) \\ &= z_j(b_i) \\ &= \delta ij. \end{aligned}$$

Selesailah pembuktian teorema 77.

Teorema 78 $\mathfrak{D}$ *merupakan free* $\mathbf{Z}$*-module dengan rank* d*, dimana* d *adalah degree dari* $\min_{\mathbf{Q}}^{\alpha}$.

Mari kita buktikan teorema 78. Teorema 74 mengatakan bahwa terdapat basis $b_1, b_2, \ldots, b_d$ untuk $\mathbf{Q}(\alpha)/\mathbf{Q}$ dimana setiap $b_i \in \mathfrak{D}$. Menurut teorema 77 terdapat basis $c_1, c_2, \ldots, c_d$ dimana $(b_i, c_j) = \delta_{ij}$. Jika $z \in \mathfrak{D}$ maka z dapat ditulis sebagai $z = \sum_{j=1}^{d} a_j c_j$. Kita dapatkan

$$T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_i z) \quad = \quad T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(\sum_{j=1}^{d} a_j b_i c_j)$$

$$\begin{aligned} &= \sum_{j=1}^{d} a_j T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_i c_j) \\ &= \sum_{j=1}^{d} a_j \delta_{ij} \\ &= a_i. \end{aligned}$$

Karena $T_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}(b_i z) \in \mathbf{Z}$ maka setiap $a_i \in \mathbf{Z}$, jadi $\mathfrak{D}$ merupakan *submodule* dari *free* $\mathbf{Z}$-*module* $\bigoplus_{j=1}^{d} \mathbf{Z}c_j$. Karena $\mathfrak{D}$ juga mencakup *free* $\mathbf{Z}$-*module* $\bigoplus_{j=1}^{d} \mathbf{Z}b_j$ maka $\mathfrak{D}$ merupakan *free* $\mathbf{Z}$-*module* dengan rank d, membuktikan teorema 78.

Teorema 79 $\mathfrak{D}$ *adalah suatu Noetherian ring.*

Berdasarkan teorema 78, $\mathfrak{D}$ merupakan suatu *free* $\mathbf{Z}$-*module* dengan *rank* d. Karena $\mathbf{Z}$ adalah suatu *Noetherian ring*, maka berdasarkan teorema 73, $\mathfrak{D}$ yang merupakan *direct product* dari d salinan $\mathbf{Z}$ juga *Noetherian*, membuktikan teorema 79.

Teorema 80 *Setiap non-trivial ideal prima* I *dalam* $\mathfrak{D}$ *maksimal.*

Untuk membuktikan teorema 80, kita pilih $x \in I$ dimana $x \neq 0$. Karena $x \in \mathfrak{D}$ maka terdapat *polynomial*

$$x^m + a_{m-1}x^{m-1} + \ldots + a_1 x + a_0 = 0$$

dimana setiap $a_i \in \mathbf{Z}$ dan m adalah bilangan bulat positif yang sekecil mungkin (minimal). Jadi $a_0 \neq 0$ dan kita dapatkan

$$a_0 \in \mathfrak{D}x \cap \mathbf{Z} \subseteq I \cap \mathbf{Z},$$

jadi $I \cap \mathbf{Z}$ merupakan *non-trivial ideal* dalam $\mathbf{Z}$. Untuk setiap $a, b \in \mathbf{Z}$, jika $ab \in I \cap \mathbf{Z}$, maka $ab \in I$, dan karena I adalah *ideal* prima, maka $a \in I$ atau $b \in I$. Akibatnya

$$a \in I \cap \mathbf{Z} \text{ atau } b \in I \cap \mathbf{Z},$$

jadi $I \cap \mathbf{Z}$ adalah *ideal* prima dalam $\mathbf{Z}$. Berdasarkan teorema 18, $I \cap \mathbf{Z}$ adalah *ideal* maksimal dalam $\mathbf{Z}$. Sekarang kita lihat apa konsekuensinya jika I bukan *ideal* maksimal dalam $\mathfrak{D}$. Berarti terdapat *ideal* J dimana $I \subset J$ dan $J \neq \mathfrak{D}$. Jika kita pilih $y \in J$ dengan $y \notin I$, maka kita akan dapatkan suatu b_0 dimana

$$b_0 \in \mathfrak{D}y \cap \mathbf{Z} \subseteq J \cap \mathbf{Z}.$$

Kita juga ketahui bahwa $b_0 \notin I$, jadi $b_0 \notin I \cap \mathbf{Z}$. Jadi $I \cap \mathbf{Z} \subset J \cap \mathbf{Z}$. Karena $1 \notin J \cap \mathbf{Z}$ maka $J \cap \mathbf{Z} \neq \mathbf{Z}$, jadi $I \cap \mathbf{Z}$ bukan suatu *ideal* maksimal, suatu kontradiksi. Selesailah pembuktian teorema 80.

Sekarang kita tunjukkan bahwa $\mathfrak{D}$ merupakan suatu *Dedekind domain*:

- Berdasarkan definisinya, $\mathfrak{D}$ *integrally closed* dalam $\mathbf{Q}(\alpha)$. Teorema 75 mengatakan bahwa $\mathbf{Q}(\alpha)$ merupakan *fraction field* untuk $\mathfrak{D}$. Jadi $\mathfrak{D}$ *integrally closed*.
- Berdasarkan teorema 79, $\mathfrak{D}$ adalah suatu *Noetherian ring*.
- Yang terahir, berdasarkan teorema 80, setiap *non-trivial ideal* prima dalam $\mathfrak{D}$ maksimal.

Selanjutnya kita jelaskan konsep penguraian *ideal*, karena itulah fokus dari teori mengenai *algebraic numbers*, bukan aritmatika dalam *number field*. Kita mulai dengan penjelasan konsep produk dari *ideal*. Secara formal, produk dari *ideal* I dan J didefinisikan sebagai berikut:

$$IJ = \{a_1b_1 + \ldots a_nb_n | a_i \in I, b_i \in J, i = 1, 2, \ldots, n; n = 1, 2, 3, \ldots\}$$

dengan kata lain produk *ideal* adalah himpunan yang isinya adalah semua penjumlahan produk a_ib_i yang *finite*. Tidak terlalu sulit untuk melihat bahwa:

$$IJ \subseteq I \cap J.$$

Juga, jika

$$IJ \subseteq P$$

dimana P adalah suatu *ideal* prima, maka

$$I \subseteq P \text{ atau } J \subseteq P.$$

Teorema 81 *Jika I merupakan non-trivial ideal dari suatu Noetherian integral domain R, maka I mencakup produk dari non-trivial ideal prima.*

Untuk membuktikan teorema 81 mari kita lihat apa konsekuensinya jika I tidak mencakup produk dari *non-trivial ideal* prima. Jika S merupakan himpunan semua *non-trivial ideal* dari R yang tidak mencakup produk dari *non-trivial ideal* prima maka, karena R adalah suatu *Noetherian ring*, S mempunyai elemen maksimal, sebut saja J. Karena $J \in S$ maka J tidak mungkin prima, jadi terdapat $a, b \in R$ dimana $a \notin J$ dan $b \notin J$ tetapi $ab \in J$. Karena J adalah elemen maksimal dari S, maka $J + aR$ dan $J + bR$ masing-masing merupakan *non-trivial ideal* yang mencakup produk dari *non-trivial ideal* prima, jadi $(J + aR)(J + bR)$ juga mencakup produk dari *non-trivial ideal* prima. Karena

$$(J + aR)(J + bR) \subseteq (J + abR) = J$$

maka J juga mencakup produk dari *non-trivial ideal* prima, suatu kontradiksi. Jadi I harus mencakup produk dari *non-trivial ideal* prima dan selesailah pembuktian teorema 81.

Sebelum kita bahas teorema mengenai penguraian *ideal*, kita perlu konsep *fractional ideal*. Kita gunakan himpunan $I = (\frac{5}{3})\mathbf{Z}$ sebagai motivasi. Karena bukan merupakan *subset* dari $\mathbf{Z}$, I bukan suatu *ideal*. Akan tetapi I mempunyai sifat mirip dengan *ideal* yaitu

- Jika $a, b \in I$ maka $a + b \in I$.
- Jika $a \in I$ dan $n \in \mathbf{Z}$ maka $na \in I$.

Juga, jika kita kalikan setiap elemen I dengan 3 kita akan dapatkan suatu *ideal* yaitu $5\mathbf{Z}$. Kita katakan bahwa I merupakan suatu *fractional ideal* yang mempunyai definisi sebagai berikut:

Definisi 41 (Fractional Ideal) *Jika R merupakan suatu integral domain dengan fraction field K dan I merupakan R-submodule dari K, dan jika terdapat suatu $0 \neq r \in R$ dimana $rI \subseteq R$, maka I disebut fractional ideal dan r merupakan denominator dari I.*

Jadi $(\frac{5}{3})\mathbf{Z}$ merupakan *fractional ideal* dengan *denominator* $r = 3$. Tentunya *ideal* biasa juga merupakan *fractional ideal* dengan *denominator* $r = 1$. Produk untuk *fractional ideal* didefinisikan serupa dengan produk untuk *ideal* biasa.

Teorema 82 *Jika I merupakan non-trivial ideal prima dari suatu Dedekind domain R, K merupakan fraction field dari R, dan $J = \{x \in K | xI \subseteq R\}$, maka J merupakan fractional ideal dan $IJ = R$.*

Mari kita buktikan teorema 82. Karena untuk $0 \neq r \in I$ dan $x \in J$ kita dapatkan $rx \in R$, maka $rJ \subseteq R$ jadi J merupakan suatu *fractional ideal*. Berikutnya kita akan tunjukkan bahwa

$$R \subset J.$$

Jika $x \in R$ maka $xI \subseteq R$ dan $x \in K$, yang berarti $R \subseteq J$. Untuk suatu $0 \neq a \in I$, terdapat *principal ideal* $aR \subseteq I$. Karena R *Noetherian*, teorema 81 menjamin bahwa terdapat bilangan positif n yang terkecil dengan

$$P_1 P_2 \cdots P_n \subseteq aR \subseteq I$$

dimana setiap P_i merupakan *ideal* prima dan $P_i \neq 0$. Karena I prima, maka I mencakup salah satu P_i sebut saja P_1. Untuk $n \geq 2$ kita buat

$$I_1 = P_2 \cdots P_n.$$

Karena n adalah bilangan positif terkecil dengan $P_1 \cdots P_n \subseteq aR$, maka $I_1 \not\subseteq aR$. Jika kita pilih $b \in I_1$ dimana $b \notin aR$, maka karena $II_1 = P_1 P_2 \cdots P_n \subseteq aR$,

$bI \subseteq aR$. Jadi $ba^{-1}I \subseteq R$ yang berarti $ba^{-1} \in J$. Tetapi $ba^{-1} \notin R$ karena $b \notin R$, jadi $R \subset J$. Untuk $n = 1$,

$$P_1 \subseteq aR \subseteq I = P_1,$$

jadi $aR = I$. Karena aR merupakan *ideal* prima, maka terdapat $b \in R$ dimana $b \notin aR$, jadi $ba^{-1} \notin R$. Tetapi

$$ba^{-1}I = ba^{-1}aR = bR \subseteq R,$$

yang berarti $ba^{-1} \in J$. Jadi untuk $n = 1$ kita dapati juga $R \subset J$. Melanjutkan pembuktian teorema 82, karena $IJ \subseteq R$ berdasarkan definisi J, berarti IJ merupakan *ideal* dari R dan kita dapatkan

$$I = IR \subseteq IJ \subseteq R.$$

Karena I adalah maksimal (I prima), maka $IJ = I$ atau $IJ = R$. Jadi kita tinggal tunjukkan bahwa $IJ \neq I$. Untuk itu kita lihat apa konsekuensinya jika $IJ = I$. Jika $x \in J$ maka $xI \subseteq IJ$ dan karena asumsi $IJ = I$ maka $xI \subseteq I$. Menggunakan induksi kita dapatkan

$$x^n I \subseteq I$$

untuk $n = 1, 2, \ldots$. Untuk $0 \neq r \in I$, $rx^n \in x^n I \subseteq I \subseteq R$, jadi $R[x]$ merupakan *fractional ideal*. Karena $rR[x] \subseteq R$ maka $R[x] \subseteq r^{-1}R$, dan karena $r^{-1}R$ *isomorphic* dengan R sebagai R-*module* (yang berarti $r^{-1}R$ *Noetherian* jadi *finitely generated*), maka $R[x]$ merupakan *finitely generated* R-*submodule* dari K. Berarti x *integral* atas R. Karena R *integrally closed* (R adalah *Dedekind domain*) maka $x \in R$, jadi $J \subseteq R$. Tetapi ini merupakan kontradiksi dengan $R \subset J$, jadi tidak mungkin $IJ = I$. Jadi $IJ = R$ dan selesailah pembuktian teorema 82.

Teorema 83 *Jika I merupakan suatu non-trivial ideal dari suatu Dedekind domain R maka I dapat diuraikan secara unik sebagai*

$$I = P_1 P_2 \cdots P_n$$

dimana setiap P_i merupakan ideal prima (dan bisa terdapat repetisi dalam produk).

Mari kita buktikan teorema 83. Untuk membuktikan bahwa setiap *non-trivial ideal* dapat diuraikan sebagai produk, kita buat himpunan S sebagai himpunan dari semua *non-trivial proper ideal* dari R yang tidak dapat diuraikan sebagai produk dari *ideal* prima. Karena R merupakan suatu *Noetherian ring*, jika S tidak kosong, maka S mempunyai elemen yang maksimal, sebut saja I_0. Tentu saja I_0 tercakup dalam suatu *ideal* maksimal (dan prima) I_1 dan berdasarkan

teorema 82, I_1 mempunyai *inverse* berupa *fractional ideal* sebut saja J (jadi $I_1 J = R$). Kita dapatkan

$$I_0 = I_0 R \subseteq I_0 J \subseteq I_1 J = R.$$

Jadi $I_0 J$ merupakan suatu *ideal*. Mengunakan cara yang sama dengan yang berada dalam pembuktian teorema 82 kita dapat tunjukkan bahwa $I_0 \subset I_0 J$. Karena I_0 adalah elemen maksimal S, maka $I_0 J$ dapat diuraikan sebagai produk dari *ideal* prima, sebut saja

$$I_0 J = Q_1 Q_2 \cdots Q_m$$

dimana setiap Q_i merupakan *ideal* prima. Jika kita kalikan persamaan dengan I_1 kita dapatkan

$$\begin{aligned} I_0 I_1 J &= I_1 Q_1 Q_2 \cdots Q_m, \\ I_0 R &= I_1 Q_1 Q_2 \cdots Q_m, \\ I_0 &= I_1 Q_1 Q_2 \cdots Q_m. \end{aligned}$$

Jadi I_0 merupakan produk dari *ideal* prima, suatu kontradiksi karena $I_0 \in S$. Berarti S adalah himpunan kosong, jadi setiap *non-trivial ideal* dalam *Dedekind domain* dapat diuraikan sebagai produk dari *ideal* prima. Untuk menunjukkan bahwa produk tersebut unik (hanya urutannya yang dapat diubah), kita ingin tunjukkan bahwa jika

$$P_1 P_2 \cdots P_n = Q_1 Q_2 \cdots Q_m$$

dimana setiap P_i dan Q_i merupakan *ideal* prima, maka $m = n$ dan urutan faktor bisa diubah hingga $P_i = Q_i$ untuk setiap $1 \leq i \leq n$. Untuk itu kita gunakan induksi. Untuk $n = 1$, $m = 1 = n$ dan $P_1 = Q_1$ karena P_1 tidak mungkin diuraikan sebagai produk *ideal* prima. Untuk $n > 1$, jika

$$P_1 P_2 \cdots P_{n-1} = Q_1 Q_2 \cdots Q_{n-1}$$

dimana $P_i = Q_i$ untuk setiap $1 \leq i \leq n-1$, maka $m = n$ dan $P_n = Q_n$ karena P_n tidak mungkin diuraikan sebagai produk *ideal* prima. Jadi $m = n$ dan $P_i = Q_i$ untuk setiap $1 \leq i \leq n-1$. Selesailah pembuktian teorema 83.

Selanjutnya kita akan bahas hubungan antara *ideal* prima dalam $\mathfrak{D}$ dengan *ideal* prima dalam $\mathbf{Z}$. Jika I merupakan *ideal* prima dalam $\mathbf{Z}$ maka $I\mathfrak{D}$ merupakan ekstensi (disebut juga *lifting*) dari I ke $\mathfrak{D}$. Meskipun $I\mathfrak{D}$ belum tentu prima, berdasarkan teorema 83, $I\mathfrak{D}$ dapat diuraikan menjadi

$$I\mathfrak{D} = \prod_{i=1}^{n} P_i^{e_i}$$

dimana setiap P_i merupakan *ideal* prima yang berbeda (*ideal* prima yang sama dikumpulkan menjadi pemangkatan dari *ideal* tersebut). Sebaliknya jika Q merupakan *ideal* prima dalam $\mathfrak{D}$ maka kita dapatkan

$$P = Q \cap \mathbf{Z}$$

sebagai kontraksi dari Q ke $\mathbf{Z}$. Tidak terlalu sulit untuk melihat bahwa P merupakan *ideal* prima dalam $\mathbf{Z}$.

Teorema 84 *Jika Q merupakan ideal prima dalam $\mathfrak{D}$, maka Q tampil dalam penguraian $P\mathfrak{D}$ jika dan hanya jika $Q \cap \mathbf{Z} = P$.*

Mari kita buktikan teorema 84. Jika $Q \cap \mathbf{Z} = P$ maka $P \subseteq Q$, jadi $P\mathfrak{D} \subseteq Q$ karena Q merupakan *ideal*, yang berarti Q membagi $P\mathfrak{D}$. Jadi Q tampil dalam penguraian $P\mathfrak{D}$. Sebaliknya jika Q membagi $P\mathfrak{D}$ maka $P\mathfrak{D} \subseteq Q$. Jadi

$$P = P \cap \mathbf{Z} \subseteq P\mathfrak{D} \cap \mathbf{Z} \subseteq Q \cap \mathbf{Z}.$$

Karena dalam $\mathbf{Z}$ setiap *ideal* prima juga *ideal* maksimal, maka $P = Q \cap \mathbf{Z}$. Selesailah pembuktian teorema 84.

Selanjutnya kita perlu konsep *norm* dari suatu *ideal*. Jika H adalah suatu *ideal* dalam *ring* $\mathfrak{D}$ maka *norm* dari H adalah banyaknya *coset* H dalam $\mathfrak{D}$ dengan notasi

$$|\mathfrak{D}/H|.$$

Teorema 85 *Jika $\langle u \rangle$ merupakan principal ideal dengan generator u, maka*

$$N(\langle u \rangle) = |N(u)|.$$

Untuk membuktikan teorema 85 kita perlu melihat $\mathfrak{D}$ sebagai *lattice*. Yang dimaksud dengan *lattice* disini adalah ruang titik-titik integral, bukan suatu *partial order*, dan suatu *principal ideal* menjadi *sublattice* dari $\mathfrak{D}$. *Principal ideal* $\langle u \rangle$ ditentukan oleh vektor-vektor yang *linearly independent* sebagai berikut:

$$u, u\alpha, u\alpha^2, \ldots, u\alpha^{d-1}.$$

Jadi $\langle u \rangle$ adalah *sublattice* dari $\mathfrak{D}$ (dengan dimensi d). Lebih dari itu,

$$u, u\alpha, u\alpha^2, \ldots, u\alpha^{d-1}$$

adalah basis untuk $\langle u \rangle$. *Covolume* dari suatu *lattice* adalah determinan dari matrik generator. Untuk $\langle u \rangle$, matrik generator adalah matrik pengali untuk u, sedangkan untuk $\mathfrak{D}$ matrik generator adalah matrik identitas dengan dimensi $d \times d$:

$$\begin{bmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{bmatrix}.$$

Volume dari *lattice* adalah nilai mutlak dari *covolume*, dan dapat dipandang sebagai *unit volume* yang dibuat menggunakan basis dari *lattice*. Banyaknya *coset* $\langle u \rangle$ dalam $\mathfrak{D}$ sama dengan *index* $\langle u \rangle$ dalam $\mathfrak{D}$ yaitu rasio *volume* $\langle u \rangle$ dengan *volume* $\mathfrak{D}$. Jika U adalah matrik pengali untuk u dan I adalah matrik identitas, maka

$$\begin{aligned} |\mathfrak{D}/\langle u \rangle| &= \frac{|\det(U)|}{|\det(I)|} \\ &= |\det(U)|. \end{aligned}$$

Jadi

$$\begin{aligned} N(\langle u \rangle) &= |\det(U)| \\ &= |N(u)|. \end{aligned}$$

Berikutnya kita ingin tunjukkan bahwa *norm* untuk *ideal* juga bersifat *multiplicative*.

Teorema 86 *Jika I dan J merupakan non-trivial ideal dalam ring $\mathfrak{D}$, maka*

$$N(IJ) = N(I)N(J).$$

Untuk membuktikan teorema 86, karena J dapat diuraikan menjadi produk *ideal* prima, kita cukup menunjukkan bahwa

$$N(IP) = N(I)N(P)$$

dimana P merupakan *ideal* prima. Menggunakan teorema 43, kita dapatkan

$$\mathfrak{D}/I \simeq (\mathfrak{D}/IP)/(I/IP).$$

Jadi

$$|\mathfrak{D}/IP| = |\mathfrak{D}/I||I/IP|.$$

Karena $N(IP) = |\mathfrak{D}/IP|$ dan $N(I) = |\mathfrak{D}/I|$, kita tinggal menunjukkan bahwa

$$|I/IP| = |\mathfrak{D}/P| = N(P).$$

Karena *unique factorization* (teorema 83), maka

$$I \neq IP,$$

jadi terdapat $\alpha \in I \setminus IP$. Jika kita buat pemetaan

$$\begin{aligned} f : \mathfrak{D} &\longrightarrow I/IP \\ x &\mapsto x\alpha + IP, \end{aligned}$$

maka tidak terlalu sulit untuk melihat bahwa f merupakan suatu *homomorphism* antara $\mathfrak{D}$-*modules*. Karena P merupakan *ideal* maksimal, maka menggunakan *homomorphism theorem* untuk *modules* yang serupa dengan teorema 39, f *surjective* dengan $\ker(f) = P$, dan kita dapatkan

$$\mathfrak{D}/P \simeq I/IP$$

jadi

$$|I/IP| = |\mathfrak{D}/P| = N(P)$$

dan selesailah pembuktian kita.

Berikutnya adalah teorema yang menghubungkan *ideal* prima dalam $\mathfrak{D}$ dengan bilangan prima dalam $\mathbf{Z}$.

Teorema 87
- *Jika* $\mathfrak{p}$ *adalah ideal dalam* $\mathfrak{D}$ *dengan* $N(\mathfrak{p}) = p$ *dimana* p *merupakan bilangan prima, maka* $\mathfrak{p}$ *adalah ideal prima dalam* $\mathfrak{D}$.
- *Sebaliknya jika* $\mathfrak{p}$ *adalah ideal prima dalam* $\mathfrak{D}$, *maka* $N(\mathfrak{p}) = p^f$ *untuk suatu bilangan bulat positif* f.

Jika $\mathfrak{p}$ merupakan *ideal* dalam $\mathfrak{D}$ dengan $N(\mathfrak{p}) = p$ untuk suatu bilangan prima p maka, karena $|\mathfrak{D}/\mathfrak{p}| = p$,

$$\mathfrak{D}/\mathfrak{p} \simeq \mathbf{Z}/p\mathbf{Z}.$$

Jadi $\mathfrak{D}/\mathfrak{p}$ merupakan suatu *field*, yang berarti $\mathfrak{p}$ adalah *ideal* maksimal, dan karena $\mathfrak{D}$ adalah suatu *Dedekind domain* berarti $\mathfrak{p}$ merupakan *ideal* prima. Jadi kita telah buktikan bagian pertama dari teorema 87. Untuk membuktikan bagian kedua, jika $\mathfrak{p}$ merupakan *ideal* prima dalam $\mathfrak{D}$, maka $\mathfrak{p} \cap \mathbf{Z}$ merupakan *ideal* prima dalam $\mathbf{Z}$ dengan *generator* bilangan prima, sebut saja p (jadi $\mathfrak{p} \cap \mathbf{Z} = p\mathbf{Z}$). Kita dapat membuat *principal ideal* $\langle p \rangle$ dalam $\mathfrak{D}$ dan berdasarkan *unique factorization* maka $\langle p \rangle$ dapat ditulis sebagai

$$\langle p \rangle = P_1^{e_1} P_2^{e_2} \cdots P_n^{e_n}$$

dimana $P_1, P_2, \ldots, P_n$ masing-masing adalah *ideal* prima yang berbeda dalam $\mathfrak{D}$ dan $e_1, e_2, \ldots, e_n$ merupakan bilangan bulat positif. Tentunya $p \in P_i$ untuk setiap $1 \leq i \leq n$. Kita juga dapatkan

$$P_1^{e_1} P_2^{e_2} \cdots P_n^{e_n} \subseteq \mathfrak{p}$$

jadi $P_j \subseteq \mathfrak{p}$ untuk suatu $1 \leq j \leq n$. Karena P_j maksimal maka $P_j = \mathfrak{p}$. Jika kita ambil *norm* dari $\langle p \rangle$ maka kita dapatkan

$$N(\langle p \rangle) = N(P_1)^{e_1} N(P_2)^{e_2} \cdots N(P_n)^{e_n}.$$

Kita juga dapatkan

$$N(\langle p \rangle) = |N(p)| = p^d$$

dimana d merupakan dimensi dari $\mathfrak{D}$. Jadi untuk setiap $1 \leq i \leq n$ terdapat bilangan bulat positif f_i dimana $N(P_i) = p^{f_i}$. Jadi

$$N(\mathfrak{p}) = N(P_j) = p^{f_j}.$$

Selesailah pembuktian teorema 87.

12.5 Ringkasan

Di bab ini kita telah bahas konsep *algebraic number*. Bab ini dimulai dengan pembahasan struktur aljabar untuk ruang vektor dan *module*, kemudian diikuti oleh konsep *separable field extension*, lalu konsep *norm* dan *trace*, dan terahir *algebraic number theory*. Teori mengenai *algebraic numbers* digunakan dalam metode penguraian *number field sieve*.

Bab 13

Matematika VI - Test Bilangan Prima

Dalam *public key cryptography* (lihat bab 16), bilangan prima yang sangat besar yang dipilih secara "acak" kerap dibutuhkan. Suatu bilangan ganjil n yang sangat besar dipilih secara acak dan $n, n+2, n+4, \ldots$ dan seterusnya ditest hingga bilangan prima pertama ditemukan. Cara naif untuk test bilangan prima yang bersifat deterministik (pasti) adalah untuk mencoba membagi bilangan yang sedang ditest (sebut saja n) dengan setiap bilangan ganjil > 2 sampai dengan $\sqrt{n}$. Jika ada yang membagi n maka n bukan bilangan prima, sebaliknya jika tidak ada yang membagi maka n merupakan bilangan prima. Tentu saja cara ini tidak praktis untuk nilai n yang sangat besar. Karena algoritma deterministik untuk test bilangan prima tidak efisien (meskipun cara paling efisien yang sudah ditemukan mempunyai kompleksitas[1] yang tergolong *polynomial* — lihat [agr04]), maka algoritma probabilistik digunakan dalam kriptografi.

13.1 Pseudoprime dan Bilangan Carmichael

Sekarang kita bahas konsep yang digunakan mayoritas algoritma probabilistik untuk test bilangan prima. Karena 2 merupakan satu-satunya bilangan prima genap, test bilangan prima fokus pada bilangan ganjil. Berbagai varian algoritma dengan konsep berikut menggunakan cara berbeda untuk mempercepat komputasi dan juga untuk dapat menjadi efektif terhadap bilangan Carmichael (akan dibahas), tetapi dasar yang digunakan adalah *Fermat's little theorem*:

$$b^{n-1} \equiv 1 \pmod{n} \tag{13.1}$$

[1]Kompleksitas asimtotik untuk test bilangan prima adalah ukuran waktu atau memori yang diperlukan untuk komputasi seiring besarnya (dalam bits) bilangan yang sedang ditest.

jika n prima dan $\gcd(b, n) = 1$. Untuk n komposit (dapat diuraikan, jadi tidak prima), persamaan 13.1 kadang berlaku, akan tetapi ini lebih jarang terjadi daripada situasi dimana persamaan tidak berlaku.

Definisi 42 (Pseudoprime) *Jika n adalah bilangan komposit ganjil dan terdapat b dengan $\gcd(b, n) = 1$ yang mematuhi persamaan 13.1, maka n disebut pseudoprime untuk base b.*

Sebagai contoh, $n = 91$ adalah *pseudoprime* untuk *base* 3 karena $3^{90} \equiv 1 \pmod{91}$, akan tetapi 91 bukan *pseudoprime* untuk *base* 2 karena $2^{90} \equiv 64 \pmod{91}$. Kita akan bahas teori yang diperlukan untuk memberi gambaran mengenai probabilitas untuk *pseudoprime*. Kita mulai dengan teorema mengenai *order* dari *base* untuk *pseudoprime*.

Teorema 88 *Untuk suatu bilangan komposit ganjil n, n adalah pseudoprime untuk base b jika dan hanya jika order dari b dalam $(\mathbf{Z}/n\mathbf{Z})^*$ membagi $n - 1$.*

Kita mulai pembuktian teorema 88 dengan mengumpamakan bahwa n *pseudoprime* untuk *base* b dan d adalah *order* dari b dalam $(\mathbf{Z}/n\mathbf{Z})^*$. Jika d tidak membagi $n - 1$ maka kita dapatkan *remainder* $r < d$ dimana $n - 1 = cd + r$ untuk suatu c, dan karena $b^{n-1} \equiv b^{cd} \equiv 1 \pmod{n}$, maka

$$\begin{aligned} b^r &= b^{n-1-cd} \\ &\equiv 1 \pmod{n}, \end{aligned}$$

sesuatu kontradiksi karena d merupakan pangkat terkecil dari b yang menghasilkan $1 \pmod{n}$, jadi $d|n-1$. Sebaliknya jika $d|n-1$, maka $n-1 = cd$ untuk suatu c, dan kita dapatkan

$$\begin{aligned} b^{n-1} &= b^{cd} \\ &\equiv 1 \pmod{n}, \end{aligned}$$

jadi n merupakan *pseudoprime* untuk *base* b. Selesailah pembuktian teorema 88. Kita buktikan satu teorema lagi sebelum kita bahas teorema utama yang memberi gambaran mengenai probabilitas *pseudoprime*.

Teorema 89 *Untuk suatu bilangan komposit ganjil n, jika n adalah pseudoprime untuk base b_1 dan base b_2, maka n adalah pseudoprime untuk base $b_1 b_2$ dan base $b_1 b_2^{-1}$ dimana b_2^{-1} adalah inverse dari $b_2 \pmod{n}$.*

Untuk menunjukkan bahwa n *pseudoprime* untuk *base* $b_1 b_2$, pertama kita harus tunjukkan bahwa $\gcd(b_1 b_2, n) = 1$. Jika $\gcd(b_1 b_2, n) = d > 1$ maka $d|b_1 b_2$, dan karena $d \not| b_1$ dan $d \not| b_2$ maka $d = d_1 d_2$ dimana $d_1 > 1$, $d_1|b_1$ dan $d_2 > 1$, $d_2|b_2$. Ini berarti $\gcd(b_1, n)$ merupakan kelipatan dari d_1 (karena d_1 juga membagi

n), sesuatu yang tidak mungkin karena $\gcd(b_1, n) = 1$. Jadi $\gcd(b_1 b_2, n) = 1$. Yang kedua, kita harus tunjukkan bahwa $(b_1 b_2)^{n-1} \equiv 1 \pmod{n}$:

$$\begin{aligned}(b_1 b_2)^{n-1} &= b_1^{n-1} b_2^{n-1} \\ &\equiv 1 \pmod{n}.\end{aligned}$$

Jadi n merupakan *pseudoprime* untuk *base* $b_1 b_2$. Berikutnya, kita ingin tunjukkan bahwa n *pseudoprime* untuk *base* b_2^{-1}. Kita mengetahui bahwa $b_2 b_2^{-1} \equiv 1 \pmod{n}$ yang berarti terdapat suatu bilangan bulat c_1 dimana

$$b_2 b_2^{-1} = c_1 n + 1.$$

Jika $\gcd(b_2^{-1}, n) = d > 1$ maka karena $d | b_2 b_2^{-1}$, terdapat bilangan bulat c_2 dimana $c_2 d = b_2 b_2^{-1}$, jadi

$$c_2 d = c_1 n + 1.$$

Karena $d|n$, terdapat bilangan bulat c_3 dimana $n = c_3 d$, jadi

$$c_2 d = c_1 c_3 d + 1$$

yang berarti $d|1$, sesuatu yang tidak mungkin. Jadi $\gcd(b_2^{-1}, n) = 1$. Berikut kita ingin tunjukkan bahwa $(b_2^{-1})^{n-1} \equiv 1 \pmod{n}$:

$$\begin{aligned}1 &\equiv 1^{n-1} \pmod{n} \\ &\equiv (b_2 b_2^{-1})^{n-1} \pmod{n} \\ &\equiv b_2^{n-1} (b_2^{-1})^{n-1} \pmod{n} \\ &\equiv (b_2^{-1})^{n-1} \pmod{n}.\end{aligned}$$

Karena n merupakan *pseudoprime* untuk *base* b_1 dan *base* b_2^{-1} maka n merupakan *pseudoprime* untuk *base* $b_1 b_2^{-1}$. Selesailah pembuktian teorema 89.

Sekarang kita bahas teorema utama yang memberi gambaran mengenai probabilitas *pseudoprime*.

Teorema 90 *Untuk suatu bilangan n ganjil dan komposit, jika persamaan 13.1 tidak berlaku untuk suatu base b, maka persamaan 13.1 tidak berlaku untuk sedikitnya setengah dari semua base untuk* $(\mathbf{Z}/n\mathbf{Z})^*$.

Untuk membuktikan teorema 90, kita buat himpunan $\{b_1, b_2, \ldots, b_s\}$ sebagai himpunan *residue* dari semua *base* yang menjadikan n *pseudoprime*, jadi himpunan terdiri dari semua bilangan $0 < b_i < n$ yang lulus test persamaan 13.1. Jika b merupakan *base* yang gagal menjadikan n *pseudoprime*, maka berdasarkan teorema 89, bb_i juga gagal menjadikan n suatu *pseudoprime*, karena jika bb_i menjadikan n suatu *pseudoprime*, maka n juga merupakan *pseudoprime* untuk:

$$b \equiv (bb_i) b_i^{-1} \pmod{n},$$

suatu kontradiksi. Jadi himpunan *residue*

$$\{bb_1, bb_2, \ldots, bb_s\}$$

merupakan himpunan yang besarnya sama dengan himpunan $\{b_1, b_2, \ldots, b_s\}$ dan setiap elemennya menggagalkan n menjadi *pseudoprime*. Jadi jika terdapat *base* b yang menggagalkan n menjadi *pseudoprime*, maka sedikitnya separuh dari *residue classes* $\pmod n$ akan menggagalkan n menjadi *pseudoprime*. Selesailah pembuktian teorema 90. Kecuali jika n lulus test persamaan 13.1 untuk semua *base* b dengan $\gcd(b, n) = 1$, maka dengan probabilitas sedikitnya 50 persen, test persamaan akan gagal untuk suatu b yang dipilih secara acak. Secara garis besar, algoritma untuk test bilangan prima mengulang langkah-langkah berikut hingga terjadi kegagalan atau kita sudah cukup puas dengan probabilitas yang diberikan bahwa n merupakan bilangan prima.

1. Pilih suatu bilangan b secara acak sebagai *base* dimana $0 < b < n$.
2. Kalkulasi $d = \gcd(b, n)$ menggunakan algoritma Euclid.
3. Jika $d > 1$ maka n adalah bilangan komposit dan d merupakan faktor yang membagi n, kita selesai.
4. Jika $d = 1$ kita lakukan test persamaan 13.1 terhadap b. Jika tidak lulus maka n adalah bilangan komposit dan kita selesai. Jika lulus maka probabilitas bahwa n merupakan bilangan prima semakin besar.

Jika langkah-langkah diatas menghasilkan jawaban komposit, maka kita tahu dengan pasti bahwa n adalah bilangan komposit. Jika tidak menjawab komposit, maka probabilitas bahwa n adalah bilangan komposit yang akan gagal test persamaan 13.1 untuk suatu *base* adalah $\leq \frac{1}{2}$. Jadi jika langkah-langkah diatas diulang sebanyak k kali tanpa jawaban komposit, setiap kali dengan *base* baru yang dipilih secara acak, maka probabilitas bahwa n adalah bilangan komposit yang akan gagal test persamaan 13.1 untuk suatu *base* adalah $\leq \frac{1}{2^k}$, dan probabilitas bahwa n akan lulus test persamaan 13.1 untuk semua *base* adalah $\geq 1 - \frac{1}{2^k}$.

Adakah bilangan komposit yang lulus test persamaan 13.1 untuk semua *base*? Jawabnya ada, yaitu bilangan Carmichael (*Carmichael number*).

Definisi 43 (Carmichael) *Bilangan komposit n adalah Carmichael jika*

$$a^{n-1} \equiv 1 \pmod n$$

untuk semua $a \in (\mathbf{Z}/n\mathbf{Z})^*$ $(1 \leq a \leq n-1)$.

Definisi 44 (Carmichael Lambda Function) *Kita definisikan Carmichael lambda function sebagai*

$$\lambda(n) = exp((\mathbf{Z}/n\mathbf{Z})^*)$$

dimana exp(G) adalah pangkat positif terkecil e dengan $a^e = 1$ untuk setiap a dalam finite group G, jadi e merupakan kelipatan persekutuan terkecil (lowest common multiple) semua order elemen dalam G. Kita definisikan juga bahwa $\lambda(1) = 1$.

Teorema 91 *Suatu bilangan komposit n adalah Carmichael jika dan hanya jika* $\lambda(n)$ *membagi* $n - 1$.

Pembuktian teorema 91 cukup mudah karena $e = \lambda(n)$ adalah pangkat terkecil yang menghasilkan 1 jika dipangkatkan ke setiap elemen dalam $(\mathbf{Z}/n\mathbf{Z})^*$, jadi $a^e \equiv 1 \pmod{n}$ untuk setiap $a \in (\mathbf{Z}/n\mathbf{Z})^*$. Definisi bilangan Carmichael mengatakan bahwa $a^{n-1} \equiv 1 \pmod{n}$ untuk setiap elemen $a \in (\mathbf{Z}/n\mathbf{Z})^*$. Jika e tidak membagi $n - 1$, maka terdapat $0 < r < e$ dimana

$$be + r = n - 1$$

untuk suatu b, dan untuk setiap $a \in (\mathbf{Z}/n\mathbf{Z})^*$:

$$a^r = a^{n-1-be} \equiv 1 \pmod{n}.$$

Ini adalah suatu kontradiksi karena e merupakan pangkat positif terkecil yang menghasilkan 1 untuk setiap $a \in (\mathbf{Z}/n\mathbf{Z})^*$. Jadi $e = \lambda(n)$ membagi $n - 1$, dan selesailah pembuktian teorema 91. Berikut adalah beberapa persamaan mengenai λ:

$$\lambda(p^e) = p^{e-1}(p-1) \text{ untuk bilangan prima ganjil } p. \tag{13.2}$$

$$\lambda(2^e) = 2^{e-2} \text{ untuk } e \geq 3. \tag{13.3}$$

$$\lambda(2) = 1. \tag{13.4}$$

$$\lambda(4) = 2. \tag{13.5}$$

$$\lambda(n) = \text{lcm}_{1 \leq i \leq k} \lambda(p_i^{e_i}) \text{ jika } n = \prod_{i=1}^{k} p_i^{e_i}. \tag{13.6}$$

Persamaan 13.2 didapat karena $(\mathbf{Z}/p^e\mathbf{Z})^*$ dengan bilangan prima ganjil p dengan $e \geq 1$ merupakan *cyclic group* (lihat teorema 38) jadi mempunyai elemen dengan *order* terbesar $\phi(p^e) = p^{e-1}(p-1)$. Persamaan 13.3 didapat karena $(\mathbf{Z}/2^e\mathbf{Z})^*$ merupakan produk dari dua *cyclic group*, dan *order* terbesar 2^{e-2} merupakan kelipatan dari setiap *order* dalam $(\mathbf{Z}/2^e\mathbf{Z})^*$ dengan $e \geq 3$ (lihat teorema 36). Persamaan 13.4 dan 13.5 didapat karena $(\mathbf{Z}/2\mathbf{Z})^*$ dan $(\mathbf{Z}/4\mathbf{Z})^*$ merupakan *cyclic group* dengan *order* terbesar masing-masing 1 dan 2 (lihat teorema 35). Untuk menunjukkan persamaan 13.6, dimana $\prod_{i=1}^{k} p_i^{e_i}$ adalah *prime factorization* dari n, kita buat $t = \text{lcm}_{1 \leq i \leq k} \lambda(p_i^{e_i})$. Jadi

$$x^t \equiv 1 \pmod{p_i^{e_i}}$$

untuk $1 \leq i \leq k$, jadi berdasarkan *Chinese Remainder Theorem* (teorema 31) kita dapatkan

$$x^t \equiv 1 \pmod{n},$$

jadi $\lambda(n)|\text{lcm}_{1\leq i\leq k}\lambda(p_i^{e_i})$. Sekarang kita pilih a_i dengan *order* $\lambda(p_i^{e_i})$ dalam $(\mathbf{Z}/p_i^{e_i}\mathbf{Z})^*$ untuk setiap $1 \leq i \leq k$, dan kita buat $x \equiv a_i \pmod{p_i^{e_i}}$. Kita ingin tunjukkan bahwa x mempunyai *order* $t = \text{lcm}_{1\leq i\leq k}\lambda(p_i^{e_i})$ dalam $(\mathbf{Z}/n\mathbf{Z})^*$. Jika $x^{t/l} \equiv 1 \pmod{n}$ untuk $1 < l \leq t$, maka ada $\lambda(p_j^{e_j})$ yang tidak membagi t/l, dimana $1 \leq j \leq k$. Jadi $x^{\gcd(\lambda(p_j^{e_j}),t/l)} \equiv 1 \pmod{p_j^{e_j}}$. Karena $1 \leq \gcd(\lambda(p_j^{e_j}), t/l) < \lambda(p_j^{e_j})$, ini mengkontradiksi fakta bahwa $\lambda(p_j^{e_j})$ adalah pangkat terkecil dari x yang menghasilkan 1 dalam $(\mathbf{Z}/p_j^{e_j}\mathbf{Z})^*$. Jadi x mempunyai *order*

$$t = \text{lcm}_{1\leq i\leq k}\lambda(p_i^{e_i})$$

dalam $(\mathbf{Z}/n\mathbf{Z})^*$. Selesailah pembuktian persamaan 13.6. Sebagai aplikasi dari persamaan 13.6, kita tunjukkan bahwa 561 adalah bilangan Carmichael:

$$561 = 3 \cdot 11 \cdot 17$$

jadi

$$\begin{aligned} \lambda(561) &= \text{lcm}(\lambda(3), \lambda(11), \lambda(17)) \\ &= \text{lcm}(2, 10, 16) \\ &= 80. \end{aligned}$$

Karena 80 membagi $561 - 1 = 560$, maka berdasarkan teorema 91, 561 adalah suatu bilangan Carmichael.

Teorema 92 *Jika n merupakan bilangan Carmichael, maka n adalah bilangan ganjil, bebas kuadrat, dan merupakan produk dari sedikitnya 3 bilangan prima.*

Suatu bilangan disebut bebas kuadrat (*square-free*) jika tidak bisa dibagi oleh kuadrat suatu bilangan $p > 1$. Untuk menunjukkan bahwa $n > 2$ ganjil, berdasarkan persamaan 13.2 — 13.6, tidak terlalu sukar untuk melihat bahwa $2|\lambda(n)$. Karena $\lambda(n)|n-1$ (teorema 91), maka $2|n-1$, yang berarti n ganjil. Selanjutnya, jika $p^2|n$ untuk suatu bilangan ganjil $p > 2$, kita dapatkan $p|\lambda(n)$, jadi $p|n-1$ berdasarkan teorema 91. Jadi $p|n$ dan $p|n-1$ yang berarti $p|1$, suatu kontradiksi dengan $p > 2$. Yang terahir, jika $n = pq$ dengan dua bilangan prima ganjil p, q yang berlainan, maka $p-1|\lambda(n)$, jadi karena $\lambda(n)|n-1$, kita dapatkan $pq - 1 = n - 1 \equiv 0 \pmod{p-1}$. Tetapi $p \equiv 1 \pmod{p-1}$, jadi $q \equiv 1 \pmod{p-1}$, dan oleh karena itu $q \geq p$ (jika $q < p$ maka $q < p-1$, dan karena $q > 1$ maka $q \not\equiv 1 \pmod{p-1}$). Sebaliknya, dengan $q-1|\lambda(n)$ kita dapatkan $pq - 1 = n - 1 \equiv 0 \pmod{q-1}$. Tetapi $q \equiv 1 \pmod{q-1}$, jadi $p \equiv 1 \pmod{q-1}$, dan oleh karena itu $p \geq q$ (jika $p < q$ maka $p < q-1$, dan karena $p > 1$ maka $p \not\equiv 1 \pmod{q-1}$). Jadi $p = q$, suatu kontradiksi karena n bebas kuadrat. Selesailah pembuktian teorema 92.

13.2 Metode Solovay-Strassen

Banyaknya bilangan Carmichael tidak terbatas. Oleh sebab itu beberapa algoritma untuk test bilangan prima mencoba test yang lebih ketat, satu diantaranya adalah test untuk *Euler pseudoprime*. Untuk n suatu bilangan komposit ganjil, $\left(\frac{b}{n}\right)$ merupakan simbol Jacobi. Jika n merupakan bilangan prima, maka teorema 54 memberikan

$$b^{(n-1)/2} \equiv \left(\frac{b}{n}\right) \pmod{n}. \tag{13.7}$$

Definisi 45 (Euler Pseudoprime) *Bilangan komposit ganjil n yang lulus test persamaan 13.7 untuk base b disebut Euler pseudoprime untuk base b.*

Teorema 93 *Jika n merupakan Euler pseudoprime untuk base b, maka n merupakan pseudoprime untuk base b.*

Untuk membuktikan teorema 93 kita harus tunjukkan bahwa jika persamaan 13.7 berlaku, maka persamaan 13.1 juga berlaku. Ini dapat dilakukan dengan mengkuadratkan kedua sisi dari persamaan 13.7:

$$\begin{aligned} (b^{(n-1)/2})^2 &\equiv \left(\frac{b}{n}\right)^2 \pmod{n} \\ b^{n-1} &\equiv 1 \pmod{n}. \end{aligned}$$

Jika n adalah bilangan komposit ganjil, kita ingin tunjukkan bahwa persamaan 13.7 akan gagal untuk sedikitnya 50 persen dari semua *base* $b \in (\mathbf{Z}/n\mathbf{Z})^*$. Kita mulai dengan menunjukkan bahwa jika *base* b_1 lulus test persamaan 13.7 dan *base* b_2 gagal, maka *base* $b_1 b_2$ akan gagal. Jika persamaan 13.7 berlaku untuk b_1 dan $b_1 b_2$:

$$\begin{aligned} b_1^{(n-1)/2} &\equiv \left(\frac{b_1}{n}\right) \pmod{n} \\ (b_1 b_2)^{(n-1)/2} &\equiv \left(\frac{b_1 b_2}{n}\right) \pmod{n} \end{aligned}$$

maka karena

$$\begin{aligned} (b_1 b_2)^{(n-1)/2} &= b_1^{(n-1)/2} b_2^{(n-1)/2} \text{ dan} \\ \left(\frac{b_1 b_2}{n}\right) &= \left(\frac{b_1}{n}\right)\left(\frac{b_2}{n}\right), \end{aligned}$$

kita dapatkan

$$b_2^{(n-1)/2} \equiv \left(\frac{b_2}{n}\right) \pmod{n}.$$

Jadi jika

$$b_2^{(n-1)/2} \not\equiv \left(\frac{b_2}{n}\right) \pmod{n}$$

maka

$$(b_1 b_2)^{(n-1)/2} \not\equiv \left(\frac{b_1 b_2}{n}\right) \pmod{n}.$$

Berikutnya kita ingin tunjukkan bahwa jika bilangan n komposit dan ganjil, maka terdapat *base* b dimana persamaan 13.7 gagal. Jika suatu bilangan komposit dan ganjil n lulus persamaan 13.7 untuk semua *base*, maka

$$\left(\frac{b}{n}\right)^2 \equiv b^{n-1} \equiv 1 \pmod{n}$$

untuk semua b, jadi n merupakan bilangan Carmichael. Menurut teorema 92, n bebas kuadrat. Kita dapat uraikan n menjadi $n = pr$ dimana p adalah bilangan prima dan $\gcd(p, r) = 1$. Kita ambil satu *quadratic non-residue* g dalam $(\mathbf{Z}/p\mathbf{Z})^*$ dan kita pilih a:

$$\begin{aligned} a &\equiv g \pmod{p}, \\ a &\equiv 1 \pmod{r}. \end{aligned}$$

Berdasarkan *Chinese Remainder Theorem* (teorema 31), a dapat dipilih. Kita dapatkan

$$\left(\frac{a}{n}\right) = \left(\frac{a}{pr}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{r}\right) = \left(\frac{g}{p}\right)\left(\frac{1}{r}\right) = (-1)(+1) = -1$$

menggunakan persamaan 11.12. Dengan asumsi persamaan 13.7 lulus untuk semua *base*, kita dapatkan

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \equiv -1 \pmod{n}.$$

Karena $r|n$ maka

$$a^{(n-1)/2} \equiv -1 \pmod{r}.$$

Akan tetapi ini adalah suatu kontradiksi dengan $a \equiv 1 \pmod{r}$, jadi tidak mungkin semua *base* lulus test persamaan 13.7, jadi terdapat *base* b yang gagal. Yang terahir, kita tunjukkan bahwa jika ada *base* b yang gagal test persamaan 13.7, maka sedikitnya 50 persen dari *base* $b \in (\mathbf{Z}/n\mathbf{Z})^*$ gagal. Jika $\{b_1, b_2, \ldots, b_s\}$ merupakan himpunan semua *base* yang lulus test persamaan 13.7, maka $\{bb_1, bb_2, \ldots, bb_s\}$ merupakan himpunan yang elemennya semua gagal dan besarnya sama dengan besar himpunan $\{b_1, b_2, \ldots, b_s\}$. Jadi sedikitnya 50 persen dari semua *base* akan gagal test persamaan 13.7. Ini menjadi dasar dari algoritma Solovay-Strassen:

1. Pilih suatu bilangan b secara acak sebagai *base* dimana $0 < b < n$.
2. Kalkulasi $d = \gcd(b, n)$ menggunakan algoritma Euclid.
3. Jika $d > 1$ maka n adalah bilangan komposit dan d merupakan faktor yang membagi n, kita selesai.
4. Jika $d = 1$ kita lakukan test persamaan 13.7 terhadap b. Jika tidak lulus maka n adalah bilangan komposit dan kita selesai. Jika lulus maka probabilitas bahwa n merupakan bilangan prima semakin besar.

Jika langkah-langkah diatas menghasilkan jawaban komposit, maka kita tahu dengan pasti bahwa n adalah bilangan komposit. Jika tidak menghasilkan jawaban komposit, maka probabilitas bahwa n adalah bilangan komposit adalah $\leq \frac{1}{2}$. Jadi jika langkah-langkah diatas diulang sebanyak k kali tanpa jawaban komposit[2], setiap kali dengan *base* baru yang dipilih secara acak, maka probabilitas bahwa n adalah bilangan komposit adalah $\leq \frac{1}{2^k}$, dan probabilitas bahwa n merupakan bilangan prima adalah $\geq 1 - \frac{1}{2^k}$. Algoritma ini adalah contoh dari metode Monte Carlo yaitu algoritma probabilistik yang hasilnya hampir selalu benar. Kita dapat mengulang langkah-langkah sebanyak mungkin hingga probabilitas mendapatkan jawaban yang salah (tidak ada jawaban komposit, tetapi n komposit) adalah sangat kecil; sebagai contoh kita dapat targetkan agar probabilitas mendapatkan jawaban yang salah jauh lebih kecil dari probabilitas malfungsi hardware untuk komputasi.

13.3 Metode Miller-Rabin

Kita telah perkenalkan konsep *pseudoprime* dan *Euler pseudoprime*. Kita perkenalkan satu konsep lagi yaitu *strong pseudoprime* yang akan digunakan dalam algoritma Miller-Rabin, yang seperti Solovay-Strassen, merupakan algoritma Monte Carlo. Jika n adalah suatu bilangan komposit ganjil, maka terdapat s dan t dimana

$$n - 1 = 2^s t$$

dengan t berupa bilangan ganjil.

Definisi 46 (Strong Pseudoprime) *n adalah suatu strong pseudoprime untuk base b jika*

$$b^t \equiv 1 \pmod{n} \text{ atau} \tag{13.8}$$
$$b^{2^r t} \equiv -1 \pmod{n} \tag{13.9}$$

untuk suatu r dimana $0 \leq r < s$.

[2]Tentunya sekali jawaban komposit ditemukan, tidak ada gunanya untuk mengulang kembali langkah-langkah algoritma Solovay-Strassen.

Ide dibalik *strong pseudoprime* adalah fakta bahwa dalam $\mathbf{Z}/p^m\mathbf{Z}$ dimana p adalah bilangan prima ganjil dan $m \geq 1$, 1 mempunyai tidak lebih dan tidak kurang dari dua akar kuadrat: 1 dan -1. Jika n bukan merupakan pemangkatan bilangan prima ganjil, maka terdapat $2^{\omega(n)}$ akar kuadrat dari 1 yang berbeda, dimana $\omega(n)$ adalah banyaknya bilangan prima yang berbeda yang membagi n. Jadi *strong pseudoprime* lebih ketat lagi dari *pseudoprime*: bukan hanya $a^{n-1} \equiv 1 \pmod{n}$ saja yang ditest, akan tetapi akar kuadrat dari 1 juga ditest.

Teorema 94 *Jika n merupakan strong pseudoprime untuk base b, maka n merupakan Euler pseudoprime untuk base b.*

Kita harus buktikan untuk bilangan komposit ganjil n bahwa jika persamaan 13.8 atau 13.9 berlaku untuk *base b*, maka persamaan 13.7 juga berlaku. Jika persamaan 13.8 berlaku, maka

$$\begin{aligned} b^{(n-1)/2} &\equiv 1 \pmod{n} \\ &= \left(\frac{1}{n}\right) \\ &= \left(\frac{b^t}{n}\right) \\ &= \left(\frac{b}{n}\right)^t. \end{aligned}$$

Karena t ganjil, maka $\left(\frac{b}{n}\right) = 1$, jadi

$$b^{(n-1)/2} \equiv \left(\frac{b}{n}\right) \pmod{n},$$

yang berarti persamaan 13.7 juga berlaku. Jika persamaan 13.9 berlaku dengan $r = s - 1$, maka

$$b^{(n-1)/2} = b^{2^{(s-1)}t} \equiv -1 \pmod{n},$$

jadi kita ingin tunjukkan bahwa $\left(\frac{b}{n}\right) = -1$. Jika p adalah bilangan prima ganjil yang membagi n, kita dapat uraikan $p - 1 = 2^{s'}t'$ dimana t' adalah bilangan ganjil. Kita ingin tunjukkan bahwa $s' \geq s$ dan

$$\left(\frac{b}{p}\right) = \begin{cases} -1, & \text{jika } s' = s; \\ 1 & \text{jika } s' > s. \end{cases}$$

Karena $b^{2^{s-1}t} \equiv -1 \pmod{n}$, kita pangkatkan kedua sisi persamaan dengan t' yang merupakan bilangan ganjil dan mendapatkan:

$$\begin{aligned} (b^{2^{s-1}t})^{t'} &\equiv (-1)^{t'} \pmod{n}; \\ (b^{2^{s-1}t'})^{t} &\equiv -1 \pmod{n}. \end{aligned}$$

Karena $p|n$ maka

$$(b^{2^{s-1}t'})^t \equiv -1 \pmod{p}.$$

Jika $s' < s$, maka

$$b^{p-1} = b^{2^{s'}t'} \not\equiv 1 \pmod{p},$$

sesuatu yang bertentangan dengan *Fermat's little theorem* (teorema 30). Jadi $s' \geq s$. Jika $s' = s$, maka

$$\begin{aligned}\left(\frac{b}{p}\right) &\equiv b^{(p-1)/2} \pmod{p} \text{ karena } p \text{ prima} \\ &\equiv b^{2^{s'-1}t'} \pmod{p} \\ &\equiv -1 \pmod{p} \text{ (tidak mungkin 1 karena } (b^{2^{s'-1}t'})^t \equiv -1).\end{aligned}$$

Jika $s' > s$, maka

$$\begin{aligned}((b^{2^{s-1}t'})^t)^{2^{s'-s}} &\equiv (-1)^{2^{s'-s}} \pmod{p}; \\ (b^{2^{s'-1}t'})^t &\equiv 1 \pmod{p},\end{aligned}$$

jadi

$$\begin{aligned}\left(\frac{b}{p}\right) &\equiv b^{(p-1)/2} \pmod{p} \text{ karena } p \text{ prima} \\ &\equiv b^{2^{s'-1}t'} \pmod{p} \\ &\equiv 1 \pmod{p} \text{ (tidak mungkin } -1 \text{ karena } (b^{2^{s'-1}t'})^t \equiv 1).\end{aligned}$$

Kita tulis n sebagai produk dari bilangan-bilangan prima (tidak harus semua berbeda):

$$n = \prod_{i=1}^{j} p_i.$$

Jika k merupakan banyaknya p_i yang menghasilkan $s' = s$ jika kita tulis $p_i - 1 = 2^{s'}t'$ dengan t' berupa bilangan ganjil, maka

$$\begin{aligned}\left(\frac{b}{n}\right) &= \prod_{i=1}^{j}\left(\frac{b}{p_i}\right) \\ &= (-1)^k.\end{aligned}$$

Karena $p_i = 1 + 2^{s'}t'$, maka

$$p_i = \begin{cases} 1 + 2^s t' \equiv 1 + 2^s \pmod{2^{s+1}} & \text{jika } s' = s; \\ 1 + 2^{s'}t' \equiv 1 \pmod{2^{s+1}} & \text{jika } s' > s. \end{cases}$$

Karena $n = 1 + 2^s t = \prod_{i=1}^{j} p_i$, maka

$$\begin{aligned} 1 + 2^s t &= \prod_{i=1}^{j} p_i; \\ 1 + 2^s &\equiv (1 + 2^s)^k \pmod{2^{s+1}} \\ &\equiv 1 + k2^s + \ldots \pmod{2^{s+1}} \end{aligned}$$

dimana ... merepresentasikan suku-suku yang dapat dibagi oleh 2^{s+1}. Jadi k harus berupa bilangan ganjil, akibatnya

$$\left(\frac{b}{n}\right) = (-1)^k = -1.$$

Jadi kita sudah tunjukkan bahwa jika persamaan 13.9 berlaku dengan $r = s - 1$ maka persamaan 13.7 juga berlaku. Terahir, jika persamaan 13.9 berlaku dengan $r = r'$, dimana $r' < s - 1$, maka

$$\begin{aligned} b^{2^{s-1}t} &= (b^{2^{r'}t})^{2^{s-1-r'}} \\ &\equiv (-1)^{2^{s-1-r'}} \pmod{n} \\ &\equiv 1 \pmod{n}, \end{aligned}$$

dan

$$b^{(n-1)/2} = b^{2^{s-1}t} \equiv 1 \pmod{n},$$

jadi kita ingin tunjukkan bahwa $\left(\frac{b}{n}\right) = 1$. Jika p adalah bilangan prima ganjil yang membagi n, kita dapat uraikan $p - 1 = 2^{s'}t'$ dimana t' adalah bilangan ganjil. Kita ingin tunjukkan bahwa $s' \geq r' + 1$ dan

$$\left(\frac{b}{p}\right) = \begin{cases} -1, & \text{jika } s' = r' + 1; \\ 1 & \text{jika } s' > r' + 1. \end{cases}$$

Karena $b^{2^{r'}t} \equiv -1 \pmod{n}$, kita pangkatkan kedua sisi persamaan dengan t' yang merupakan bilangan ganjil dan mendapatkan:

$$\begin{aligned} (b^{2^{r'}t})^{t'} &\equiv (-1)^{t'} \pmod{n}; \\ (b^{2^{r'}t'})^{t} &\equiv -1 \pmod{n}. \end{aligned}$$

Karena $p|n$ maka

$$(b^{2^{r'}t'})^t \equiv -1 \pmod{p}.$$

Jika $s' < r' + 1$, maka

$$b^{p-1} = b^{2^{s'}t'} \not\equiv 1 \pmod{p},$$

sesuatu yang bertentangan dengan *Fermat's little theorem* (teorema 30). Jadi $s' \geq r' + 1$. Jika $s' = r' + 1$, maka

$$\begin{aligned}\left(\frac{b}{p}\right) &\equiv b^{(p-1)/2} \pmod{p} \text{ karena } p \text{ prima} \\ &\equiv b^{2^{s'-1}t'} \pmod{p} \\ &\equiv b^{2^{r'}t'} \pmod{p} \\ &\equiv -1 \pmod{p} \text{ (tidak mungkin 1 karena } (b^{2^{r'}t'})^t \equiv -1 \pmod{p}).\end{aligned}$$

Jika $s' > r' + 1$, maka

$$\begin{aligned}((b^{2^{r'}t'})^t)^{2^{s'-r'-1}} &\equiv (-1)^{2^{s'-r'-1}} \pmod{p}; \\ (b^{2^{s'-1}t'})^t &\equiv 1 \pmod{p},\end{aligned}$$

jadi

$$\begin{aligned}\left(\frac{b}{p}\right) &\equiv b^{(p-1)/2} \pmod{p} \text{ karena } p \text{ prima} \\ &\equiv b^{2^{s'-1}t'} \pmod{p} \\ &\equiv 1 \pmod{p} \text{ (tidak mungkin } -1 \text{ karena } (b^{2^{s'-1}t'})^t \equiv 1 \pmod{p}).\end{aligned}$$

Kita tulis n sebagai produk dari bilangan-bilangan prima (tidak harus semua berbeda):

$$n = \prod_{i=1}^{j} p_i.$$

Jika k merupakan banyaknya p_i yang menghasilkan $s' = r' + 1$ jika kita tulis $p_i - 1 = 2^{s'}t'$ dengan t' berupa bilangan ganjil, maka

$$\begin{aligned}\left(\frac{b}{n}\right) &= \prod_{i=1}^{j}\left(\frac{b}{p_i}\right) \\ &= (-1)^k.\end{aligned}$$

Karena $p_i = 1 + 2^{s'}t'$, maka

$$p_i = \begin{cases} 1 + 2^{r'+1}t' \equiv 1 + 2^{r'+1} \pmod{2^{r'+2}} & \text{jika } s' = r' + 1; \\ 1 + 2^{s'}t' \equiv 1 \pmod{2^{r'+2}} & \text{jika } s' > r' + 1. \end{cases}$$

Karena $n = 1 + 2^s t = \prod_{i=1}^{j} p_i$, maka

$$1 + 2^s t = \prod_{i=1}^{j} p_i;$$

$$\begin{aligned} 1 &\equiv (1+2^{r'+1})^k \pmod{2^{r'+2}} \\ &\equiv 1+k2^{r'+1}+\ldots \pmod{2^{r'+2}} \end{aligned}$$

dimana ... merepresentasikan suku-suku yang dapat dibagi oleh $2^{r'+2}$. Jadi k harus berupa bilangan genap, akibatnya

$$\left(\frac{b}{n}\right) = (-1)^k = 1.$$

Jadi kita sudah tunjukkan bahwa jika persamaan 13.9 berlaku dengan $r = r' < s-1$ maka persamaan 13.7 juga berlaku. Selesailah pembuktian teorema 94. Sebelum kita bahas teorema mengenai probabilitas *strong pseudoprime*, akan kita buktikan lebih dahulu dua teorema yang akan digunakan.

Teorema 95 *Terdapat $d = \gcd(k, m)$ elemen dalam group $\{g, g^2, g^3, \ldots, g^m = 1\}$ (g merupakan generator) yang mematuhi persamaan $x^k = 1$.*

Untuk membuktikan teorema 95, kita mengetahui bahwa elemen g^j (dengan $1 \le j \le m$) mematuhi persamaan diatas jika dan hanya jika $g^{jk} = 1$, dengan kata lain jika dan hanya jika $m|jk$, jadi

$$g^{jk} = 1 \iff m|jk \iff \frac{m}{d}|j\frac{k}{d}.$$

Karena $\frac{m}{d}$ koprima dengan $\frac{k}{d}$, maka

$$g^{jk} = 1 \iff \frac{m}{d}|j \ (j \text{ merupakan kelipatan } \frac{m}{d}).$$

Terdapat d bilangan $i\frac{m}{d}$ dimana $1 \le i\frac{m}{d} \le m$:

$$\frac{m}{d}, \frac{2m}{d}, \frac{3m}{d}, \ldots, \frac{dm}{d}.$$

Selesailah pembuktian teorema 95.

Teorema 96 *Jika p adalah bilangan prima ganjil, dan kita tuliskan $p-1 = 2^{s'}t'$ dimana t' adalah bilangan ganjil, maka banyaknya elemen $x \in (\mathbf{Z}/p\mathbf{Z})^*$ yang mematuhi persamaan*

$$x^{2^r t} \equiv -1 \pmod{p},$$

dimana t adalah bilangan ganjil, adalah 0 jika $r \ge s'$, atau $2^r \gcd(t, t')$ jika $r < s'$. Dalam notasi formal:

$$\sharp\{x \in (\mathbf{Z}/p\mathbf{Z})^* | x^{2^r t} \equiv -1 \pmod{p}\} = \begin{cases} 0 & \text{jika } r \ge s', \\ 2^r \gcd(t, t') & \text{jika } r < s'. \end{cases}$$

Untuk membuktikan teorema 96, jika g adalah *generator* untuk $(\mathbf{Z}/p\mathbf{Z})^*$, maka $x = g^j$ untuk suatu j dengan $0 \leq j < p-1$. Karena $g^{(p-1)/2} \equiv -1 \pmod{p}$ dan $p - 1 = 2^{s'}t'$, maka relasi *congruence* dalam teorema ekuivalen dengan

$$2^r tj \equiv 2^{s'-1}t' \pmod{2^{s'}t'}, \tag{13.10}$$

dimana kita harus mencari j. Persamaan 13.10 mempunyai solusi jika dan hanya jika $2^{s'}t'$ membagi $2^r tj - 2^{s'-1}t'$, jadi harus ada bilangan bulat m dengan

$$\begin{aligned} 2^{s'}t'm &= 2^r tj - 2^{s'-1}t'; \\ m &= 2^{r-s'}\frac{t}{t'}j - \frac{1}{2}. \end{aligned}$$

Jika $r \geq s'$ maka tidak ada bilangan bulat j yang dapat membuat m menjadi bilangan bulat (karena $2^{r-s'}$ adalah bilangan bulat dan t' adalah bilangan ganjil, jadi $2^{r-s'}\frac{t}{t'}j$ tidak mungkin menjadi kelipatan dari $\frac{1}{2}$). Jadi persamaan 13.10 tidak mempunyai solusi jika $r \geq s'$. Jika $r < s'$, kita bagi persamaan 13.10 dengan $2^r d$ dimana $d = \gcd(t, t')$ mendapatkan:

$$\frac{t}{d}j \equiv 2^{s'-r-1}\frac{t'}{d} \pmod{2^{s'-r}\frac{t'}{d}}, \tag{13.11}$$

dimana $\frac{t}{d}$ dan $\frac{t'}{d}$ merupakan bilangan bulat. Karena $\gcd(\frac{t}{d}, 2^{s'-r-1}\frac{t'}{d}) = 1$ maka persamaan 13.11 mempunyai solusi yang unik untuk j. Jadi persamaan 13.10 mempunyai $2^r d$ solusi untuk j. Selesailah pembuktian teorema 96.

Teorema 97 *Suatu bilangan komposit ganjil n adalah strong pseudoprime untuk maksimum 25 persen dari base b dengan $0 < b < n$.*

Pembuktian teorema 97 terdiri dari tiga situasi yang berbeda:

1. Situasi dimana n dapat dibagi oleh kuadrat dari suatu bilangan prima ganjil p ($p^2|n$).

2. Situasi dimana n merupakan produk dari dua bilangan prima ganjil p dan q yang berbeda ($n = pq$).

3. Situasi dimana n merupakan produk dari tiga atau lebih bilangan prima ganjil yang berbeda ($n = p_1 p_2 \ldots p_k$, $k \geq 3$).

Jika p adalah suatu bilangan prima ganjil dan $p^2|n$, kita ingin tunjukkan bahwa n merupakan *strong pseudoprime* untuk tidak lebih dari $(n-1)/4$ *base* b, dengan $0 < b < n$. Ini kita lakukan dengan menunjukkan bahwa n merupakan *pseudoprime* untuk tidak lebih dari $(n-1)/4$ *base* b (ingat bahwa suatu *strong pseudoprime* juga merupakan *Euler pseudoprime* menurut teorema 94,

jadi menurut teorema 93 juga merupakan *pseudoprime*). Teorema 38 mengatakan bahwa $(\mathbf{Z}/p^2\mathbf{Z})^*$ merupakan suatu *cyclic group*, jadi terdapat *generator* g dimana

$$(\mathbf{Z}/p^2\mathbf{Z})^* = \{g, g^2, g^3, \ldots g^{p(p-1)}\}.$$

Menurut teorema 95, banyaknya b dengan $0 \leq b < p^2$ dimana

$$b^{n-1} \equiv 1 \pmod{p^2} \tag{13.12}$$

adalah $d = \gcd(p(p-1), n-1)$. Karena $p|n$ maka $p \not| n-1$, jadi $p \not| d$. Akibatnya $d \leq p-1$, jadi proporsi b dari 1 sampai dengan $n-1$ yang mematuhi persamaan 13.12 adalah

$$\frac{p-1}{p^2-1} = \frac{1}{p+1} \leq \frac{1}{4}.$$

Karena proporsi b dari 1 sampai dengan $n-1$ yang mematuhi $b^{n-1} \equiv 1 \pmod{n}$ tidak lebih dari proporsi b yang mematuhi persamaan 13.12, maka n merupakan *pseudoprime* untuk tidak lebih dari $1/4$ *base* b dengan $0 < b < n$. Selesailah pembuktian untuk situasi $p^2|n$.

Jika $n = pq$, kita tulis $p - 1 = 2^{s'}t'$ dan $q - 1 = 2^{s''}t''$, dengan t', t'' berupa bilangan ganjil dan $s' \leq s''$ (kita dapat memilih p dan q sedemikian rupa). $0 < b < n$ merupakan *base* yang menjadikan n *strong pseudoprime* jika

$$b^t \equiv 1 \pmod{p} \text{ dan } b^t \equiv 1 \pmod{q}$$

atau

$$b^{2^r t} \equiv -1 \pmod{p} \text{ dan } b^{2^r t} \equiv -1 \pmod{q}$$

untuk suatu r dengan $0 \leq r < s$, dimana $n - 1 = 2^s t$ dengan t berupa bilangan ganjil. Banyaknya $0 < b < n$ untuk kemungkinan pertama adalah banyaknya $0 < b < p$ yang mengakibatkan $b^t \equiv 1 \pmod{p}$ dikalikan dengan banyaknya $0 < b < q$ yang mengakibatkan $b^t \equiv 1 \pmod{q}$. Menggunakan teorema 95, banyaknya b untuk kemungkinan pertama adalah

$$\begin{aligned} \gcd(t, p-1)\gcd(t, q-1) &= \gcd(t, 2^{s'}t')\gcd(t, 2^{s''}t'') \\ &= \gcd(t, t')\gcd(t, t'') \\ &\leq t't''. \end{aligned}$$

Untuk setiap r dengan $r < s' \leq s'' \leq s$, banyaknya b yang mematuhi $b^{2^r t} \equiv -1 \pmod{n}$, menggunakan teorema 96, adalah

$$2^r \gcd(t, t')2^r \gcd(t, t'') < 4^r t't''.$$

Karena $n - 1 > \phi(n) = 2^{s'+s"}t't''$, maka proporsi b dalam $0 < b < n$ yang menjadikan n *strong pseudoprime* adalah $\leq$

$$\frac{t't'' + \sum_{r=0}^{s'-1} 4^r t't''}{2^{s'+s''}t't''} = \frac{1 + (1 + 4 + 4^2 + \ldots 4^{s'-1})}{2^{s'+s''}}$$

$$= \quad 2^{-s'-s''}\left(1+\frac{4^{s'}-1}{4-1}\right).$$

Jika $s'' > s'$ maka ini adalah $\leq$

$$\begin{aligned} 2^{-2s'-1}\left(1+\frac{4^{s'}-1}{3}\right) &= 2^{-2s'-1}\left(\frac{2}{3}+\frac{4^{s'}}{3}\right) \\ &\leq 2^{-3}\frac{2}{3}+\frac{1}{6} \\ &= \frac{1}{12}+\frac{1}{6} \\ &= \frac{1}{4}. \end{aligned}$$

Jika $s'' = s'$ maka

$$\gcd(t,t') < t' \text{ atau } \gcd(t,t'') < t''$$

karena jika $t'|t$ dan $t''|t$ maka dari

$$n-1 = 2^s t = pq-1 = q(p-1)+q-1 \equiv q-1 \equiv 0 \pmod{t'}$$

kita dapatkan $t'|q-1$, jadi $t'|t''$, dan dari

$$n-1 = 2^s t = pq-1 = p(q-1)+p-1 \equiv p-1 \equiv 0 \pmod{t''}$$

kita dapatkan $t''|p-1$, jadi $t''|t'$, alhasil $t' = t''$ dan $p = q$. Tetapi ini bertentangan dengan asumsi $p \neq q$, jadi

$$\gcd(t,t') < t' \text{ atau } \gcd(t,t'') < t''.$$

Jika $\gcd(t,t') < t'$ maka $\gcd(t,t') \leq \frac{1}{3}t'$ karena gcd membagi t', lebih kecil dari t' dan merupakan bilangan ganjil. Demikian juga jika $\gcd(t,t'') < t''$ maka $\gcd(t,t'') \leq \frac{1}{3}t''$. Jadi proporsi b dalam $0 < b < n$ yang menjadikan n *strong pseudoprime* adalah $\leq$

$$\begin{aligned} \frac{t't''+\sum_{r=0}^{s'-1}4^r t't''}{3(2^{2s'}t't'')} &= \frac{1}{3}2^{-2s'}\left(1+\frac{4^{s'}-1}{4-1}\right) \\ &\leq \frac{1}{18}+\frac{1}{9} \\ &= \frac{1}{6} \\ &< \frac{1}{4}. \end{aligned}$$

Selesailah pembuktian untuk situasi $n = pq$.

Untuk $n = p_1 p_2 \dots p_k$, $k \geq 3$, dan $p_i \neq p_j$ jika $i \neq j$, kita tulis

$$p_i - 1 = 2^{s_i} t_i$$

dengan t_i bilangan ganjil untuk $1 \leq i \leq k$. Kita pilih urutan p_i sedemikian rupa sehingga $s_1 \leq s_i$ untuk $2 \leq i \leq k$. Ini merupakan generalisasi dari situasi $n = pq$, dan proporsi b dalam $0 < b < n$ yang menjadikan n *strong pseudoprime* adalah $\leq$

$$\begin{aligned} 2^{-s_1-s_2-\dots-s_k}\left(1+\frac{2^{ks_1}-1}{2^k-1}\right) &\leq 2^{-ks_1}\left(\frac{2^k-2}{2^k-1}+\frac{2^{ks_1}}{2^k-1}\right) \\ &= 2^{-ks_1}\frac{2^k-2}{2^k-1}+\frac{1}{2^k-1} \\ &\leq 2^{-ks}\frac{2^k-2}{2^k-1}+\frac{1}{2^k-1} \\ &= 2^{1-k} \end{aligned}$$

jadi $\leq \frac{1}{4}$ untuk $k \geq 3$. Selesailah pembuktian teorema 97. Teorema 97 menjadi dasar dari algoritma Miller-Rabin dengan $n-1 = 2^s t$ dimana t adalah bilangan ganjil:

1. Pilih suatu bilangan b secara acak sebagai *base* dimana $0 < b < n$.

2. Kalkulasi

   $$b_0 = b^t \pmod{n}, b_1 = b_0^2 \pmod{n}, \dots b_k = b_{k-1}^2 \pmod{n}$$

   hingga $k = s$ atau $b_k \equiv 1 \pmod{n}$.

3. Jika $k = s$ dan $b_k \not\equiv 1 \pmod{n}$ maka n adalah bilangan komposit dan kita selesai.

4. Jika $k \neq 0$ dan $b_{k-1} \not\equiv -1 \pmod{n}$ maka n adalah bilangan komposit dan kita selesai.

Jika $b^t \equiv 1 \pmod{n}$ maka $k = 0$ dan langkah 3 dan langkah 4 tidak menghasilkan jawaban komposit. Jika terdapat r dengan $0 \leq r < s$ dimana $b^{2^r t} \equiv -1 \pmod{n}$ maka $k = r + 1$ dan langkah 3 dan langkah 4 tidak menghasilkan jawaban komposit. Jika $b^t \not\equiv 1 \pmod{n}$ dan tidak ada r dengan $0 \leq r < s$ dimana $b^{2^r t} \equiv -1 \pmod{n}$ maka

- $k = s$ dan $b^k \not\equiv 1 \pmod{n}$, jadi langkah 3 menghasilkan jawaban komposit; atau

- $k \neq s$, $b^k \equiv 1 \pmod{n}$ tetapi $b^{k-1} \not\equiv -1 \pmod{n}$, jadi langkah 4 menghasilkan jawaban komposit.

Jadi jika langkah-langkah diatas menghasilkan jawaban komposit, maka kita tahu dengan pasti bahwa n adalah bilangan komposit. Jika tidak menghasilkan jawaban komposit, maka probabilitas bahwa n adalah bilangan komposit adalah $\leq \frac{1}{4}$. Jadi jika langkah-langkah diatas diulang sebanyak k kali tanpa jawaban komposit[3], setiap kali dengan *base* baru yang dipilih secara acak, maka probabilitas bahwa n adalah bilangan komposit adalah $\leq \frac{1}{4^k}$, dan probabilitas bahwa n merupakan bilangan prima adalah $\geq 1 - \frac{1}{4^k}$. Algoritma Miller-Rabin dianggap lebih baik dari algoritma Solovay-Strassen karena

- algoritma Solovay-Strassen lebih sukar untuk diprogram karena perlu kalkulasi dengan simbol Jacobi;
- probabilitas jawaban yang salah jika n adalah bilangan komposit lebih kecil dengan algoritma Miller-Rabin; dan
- jika algoritma Solovay-Strassen memberi jawaban komposit untuk suatu *base* b, maka algoritma Miller-Rabin juga memberikan jawaban komposit, sedangkan sebaliknya tidak selalu.

13.4 Test Deterministik

Jika Extended Riemann Hypothesis (ERH)[4] benar, maka algoritma Solovay-Strassen atau algoritma Miller-Rabin dapat digunakan untuk test bilangan prima secara deterministik dengan melakukan test untuk setiap *base* b:

$$2 \leq b \leq 2(\log n)^2.$$

Juga terdapat algoritma Cohen-Lenstra untuk test bilangan prima secara deterministik (lihat [coh84]) yang, walaupun secara teoritis bukan *order polynomial* dalam $\log n$, tetapi dalam prakteknya dapat melakukan test bilangan prima untuk bilangan dengan ratusan *digit* dalam hitungan detik dengan komputer masa kini.

Ada juga algoritma deterministik yang dapat dengan cepat menentukan bahwa bilangan adalah prima jika memang benar bilangan prima. Algoritma yang banyak digunakan pertama dikembangkan oleh Goldwasser dan Kilian (lihat [gol86]), dan kemudian dibuat lebih efisien Atkin dan diimplementasi oleh Atkin dan Morain (lihat [atk93]). Algoritma yang didasarkan pada *elliptic curve* ini bisa digunakan untuk memastikan bahwa suatu bilangan yang telah

[3]Tentunya sekali jawaban komposit ditemukan, tidak ada gunanya untuk mengulang kembali langkah-langkah algoritma Miller-Rabin.

[4]Kebenaran dari ERH belum pernah dibuktikan.

ditest oleh algoritma probabilistik (contohnya Miller-Rabin) memang benar prima. Algoritma ini dijuluki Atkin-Goldwasser-Kilain-Morain *certificate*.

13.5 Ringkasan

Bab ini telah membahas test bilangan prima, topik yang sangat penting untuk kriptografi *public key*. Meskipun telah ditemukan algoritma test bilangan prima yang bersifat deterministik dengan kompleksitas *polynomial*, dalam prakteknya algoritma yang bersifat probabilistik jauh lebih cepat. Algoritma probabilistik (Monte Carlo) untuk test bilangan prima menggunakan konsep *pseudoprime* dan dua diantaranya adalah algoritma Solovay-Strassen [sol77] dan algoritma Miller-Rabin [rab80]. Algoritma Miller-Rabin secara umum dianggap lebih baik dibandingkan algoritma Solovay-Strassen. Untuk memastikan bahwa bilangan yang telah ditest menggunakan algoritma probabilistik benar prima, algoritma Atkin-Goldwasser-Kilain-Morain dapat digunakan.

Bab 14

Matematika VII - Penguraian Bilangan Bulat

Mengalikan dua bilangan yang sangat besar relatif merupakan sesuatu yang mudah. Sebaliknya, menguraikan suatu bilangan yang sangat besar untuk mendapatkan faktor-faktornya, secara umum merupakan sesuatu yang sulit. Untuk bilangan n yang tidak terlalu besar, kita dapat mencoba membagi n dengan setiap bilangan prima $\leq \sqrt{n}$, akan tetapi ini tidak praktis dan akan memakan waktu yang terlalu lama jika n sangat besar. Algoritma untuk test bilangan prima jika gagal hanya menyatakan bahwa bilangan adalah komposit, tetapi tidak menolong mencarikan faktor-faktornya. Keamanan algoritma RSA (lihat bagian 16.1), satu algoritma kriptografi *public key*, didasarkan pada pengamatan bahwa belum ada algoritma yang efisien untuk menguraikan bilangan yang sangat besar. Meskipun secara umum penguraian bilangan besar sangat sulit, ada beberapa kondisi yang dapat menyebabkan suatu bilangan besar mudah untuk diuraikan. Juga, kemajuan dibidang teknologi komputer dan teknik penguraian membuat jangkauan bilangan yang dapat diuraikan semakin besar, dan beberapa prediksi dimasa lalu ternyata jauh meleset. Sebagai contoh, tahun 1976, Martin Gardner menulis dalam Scientific American bahwa kunci RSA sebesar 129 digit akan aman untuk sekitar 40 *quadrillion* tahun. Ternyata kunci tersebut dapat diuraikan menggunakan metode *quadratic sieve* tahun 1994.

Kita akan bahas beberapa algoritma penguraian dengan harapan bahwa ini akan membantu kita memahami betapa sukarnya penguraian, dan menolong kita untuk mewaspadai kondisi yang dapat membuat suatu bilangan besar mudah untuk diuraikan.

14.1 Metode Rho

Metode Rho (*Rho method*) adalah algoritma Las Vegas untuk menemukan faktor suatu bilangan besar secara probabilistik. Algoritma Las Vegas adalah algoritma probabilistik yang jika sukses maka hasilnya dijamin benar. Untuk mencari faktor dari n, metode Rho menggunakan pemetaan $f(x)$ dari $\mathbf{Z}/n\mathbf{Z}$ ke $\mathbf{Z}/n\mathbf{Z}$, contohnya *polynomial* $f(x) = x^2 + 1$. Kita pilih nilai awal untuk $x = x_0$ (contohnya $x_0 = 1$ atau $x_0 = 2$), lalu kalkulasi secara bertahap $x_1, x_2, \ldots, x_m$ sebagai berikut

$$\begin{aligned} x_1 &= f(x_0) \\ x_2 &= f(x_1) \\ \ldots & \\ x_m &= f(x_{m-1}), \end{aligned}$$

jadi

$$x_{j+1} = f(x_j), j = 0, 1, 2, \ldots, m-1.$$

Langkah berikutnya adalah mencari pasangan x_j, x_k dimana x_j dan x_k berada dalam *congruence class* yang berbeda modulo n, tetapi berada dalam *congruence class* yang sama modulo d untuk suatu d yang membagi n. Ini dapat dilakukan menggunakan gcd, jadi jika

$$d = \gcd(x_k - x_j, n),$$

maka d merupakan faktor dari n. Sebagai contoh, jika $n = 119$, $f(x) = x^2 + 1$ dan $x_0 = 1$, maka

$$x_1 = 2, x_2 = 5, x_3 = 26, \ldots,$$

dan kita dapatkan

$$d = \gcd(x_3 - x_2, 119) = \gcd(21, 119) = 7.$$

Jadi 7 merupakan faktor dari 119. Tentunya 119 bukan suatu bilangan yang besar, dan dapat diuraikan dengan mencoba membagi 119 dengan setiap bilangan prima $\leq \sqrt{119}$, contoh diatas hanya menjelaskan bagaimana faktor dicari.

Untuk metode Rho, pemetaan $f(x)$ sebaiknya adalah sesuatu pemetaan yang "acak," jadi bukan *polynomial* linear dan sebaiknya bukan pemetaan yang *bijective*. Untuk mendapatkan gambaran mengenai tingkat kesuksesan metode Rho dan waktu yang dibutuhkan untuk menemukan faktor, kita butuh konsep *"average map"* yang merepresentasikan pemetaan acak. Metode Rho mencari indeks pertama k dimana terdapat indeks j dengan $j < k$ dan

$$x_j \equiv x_k \pmod{d}.$$

Kita pelajari ini dengan memperlakukan pemetaan $f(x)$ sebagai pemetaan dari $\mathbf{Z}/d\mathbf{Z}$ ke $\mathbf{Z}/d\mathbf{Z}$ dan mencari tahu probabilitas bahwa d tidak ditemukan setelah x_k dikalkulasi dan diperiksa terhadap setiap x_i dengan $0 \leq i < k$.

Teorema 98 *Jika f adalah suatu pemetaan dari $\mathbf{Z}/d\mathbf{Z}$ ke $\mathbf{Z}/d\mathbf{Z}$, $x_0 \in \mathbf{Z}/d\mathbf{Z}$, $x_{j+1} = f(x_j)$ untuk $j = 0, 1, 2, \ldots$, dan indeks k merupakan bilangan bulat positif dan λ adalah bilangan nyata positif dengan*

$$k = 1 + \sqrt{2\lambda d},$$

maka proporsi pasangan f, x_0 yang menghasilkan

$$x_0, x_1, x_2, \ldots, x_k$$

yang berbeda modulo d ($x_i \not\equiv x_j \pmod{d}$ untuk $i \neq j$) adalah $< e^{-\lambda}$, dimana semua kemungkinan pemetaan f dari $\mathbf{Z}/d\mathbf{Z}$ ke $\mathbf{Z}/d\mathbf{Z}$ dan semua kemungkinan $x_0 \in \mathbf{Z}/d\mathbf{Z}$ diperhitungkan.

Mari kita buktikan teorema 98. Ada d kemungkinan untuk x_0 dan d^d kemungkinan untuk pemetaan f dari $\mathbf{Z}/d\mathbf{Z}$ ke $\mathbf{Z}/d\mathbf{Z}$, jadi total ada d^{d+1} kemungkinan pasangan f, x_0. Dari semua kemungkinan pasangan f, x_0 kita pilih pasangan-pasangan yang menghasilkan $x_0, x_1, x_2, \ldots, x_k$ yang berbeda modulo d. Ada d kemungkinan untuk x_0, $d-1$ kemungkinan untuk $f(x_0) = x_1$ untuk setiap x_0 karena x_1 harus berbeda dari x_0, dan seterusnya sampai dengan $d-k$ kemungkinan untuk $f(x_{k-1}) = x_k$ untuk setiap x_{k-1}. Sisanya untuk $d-k$ elemen $x_{k+1}, \ldots, x_d$ tidak ada syarat untuk f selain harus menghasilkan elemen dalam $\mathbf{Z}/d\mathbf{Z}$, jadi ada d^{d-k} kemungkinan hasil f untuk $x_{k+1}, \ldots, x_d$. Jadi total jumlah pasangan f, x_0 yang menghasilkan $x_0, x_1, x_2, \ldots, x_k$ yang berbeda modulo d ada

$$d^{d-k} \prod_{j=0}^{k} (d-j),$$

jadi proporsi pasangan f, x_0 yang menghasilkan $x_0, x_1, x_2, \ldots, x_k$ yang berbeda modulo d adalah

$$\begin{aligned} \frac{d^{d-k} \prod_{j=0}^{k}(d-j)}{d^{d+1}} &= d^{-k-1} \prod_{j=0}^{k} (d-j) \\ &= \prod_{j=1}^{k} (1 - \frac{j}{d}). \end{aligned}$$

Berdasarkan fakta bahwa $\log(1-x) < -x$ untuk $0 < x < 1$, maka

$$\log \left(\prod_{j=1}^{k} (1 - \frac{j}{d}) \right) < \sum_{j=1}^{k} -\frac{j}{d}$$

$$\begin{aligned} &= \frac{-k(k+1)}{2d} \\ &< \frac{-k^2}{2d} \\ &< \frac{-(\sqrt{2\lambda d})^2}{2d} \\ &= -\lambda. \end{aligned}$$

Jadi proporsi pasangan f, x_0 yang menghasilkan $x_0, x_1, x_2, \ldots, x_k$ yang berbeda modulo d adalah

$$\prod_{j=1}^{k}(1 - \frac{j}{d}) < e^{-\lambda}.$$

Selesailah pembuktian teorema 98. Teorema 98 memberi gambaran mengenai tingkat kesuksesan metode Rho: jika d adalah faktor dari n, probabilitas bahwa metode Rho tidak menemukan d setelah x_k dikalkulasi adalah $< e^{-\lambda}$ jika f direratakan untuk semua kemungkinan, dengan kata lain jika f merupakan suatu *"average map."*

Dalam implementasi metode Rho, jika untuk setiap x_k kita periksa setiap x_j dengan $j < k$, akan sangat tidak efisien jika kita sedang menguraikan n yang besar. Kita akan modifikasi metode Rho sehingga untuk setiap x_k kita hanya periksa satu x_j: jika x_k merupakan bilangan bulat dengan h bit ($2^{h-1} \leq x_k < 2^h$), maka kita hanya periksa x_j dimana $j = 2^{h-1}-1$. Sebagai contoh, untuk x_2 dan x_3, kita hanya periksa x_1, dan untuk x_4, x_5, x_6 dan x_7, kita hanya periksa x_3. Meskipun x_k yang ditemukan bukanlah yang pertama yang menghasilkan

$$x_k \equiv x_j \pmod{d}$$

untuk $j < k$, kita dapat tunjukkan bahwa jika metode Rho tanpa modifikasi menghasilkan

$$x_{k_0} \equiv x_{j_0} \pmod{d}$$

dengan $j_0 < k_0$, maka metode Rho yang telah dimodifikasi menghasilkan

$$x_k \equiv x_j \pmod{d}$$

dengan $k - j = k_0 - j_0$ dan $k \geq k_0$. Ini karena jika $k = k_0 + m$ untuk suatu $m \geq 0$, maka jika kita aplikasikan *polynomial* f ke dua sisi dari $x_{k_0} \equiv x_{j_0} \pmod{d}$ kita akan dapatkan $x_k \equiv x_j \pmod{d}$. Sebagai contoh, jika metode Rho tanpa modifikasi menemukan

$$x_3 \equiv x_2 \pmod{d}$$

maka metode Rho dengan modifikasi akan menemukan

$$x_4 \equiv x_3 \pmod{d}.$$

Jika k_0 merupakan bilangan dengan h bit, maka $j = 2^h - 1$ (contoh diatas $k_0 = 3$ adalah bilangan dengan $h = 2$ bit, jadi $j = 2^2 - 1 = 3$) dan $k = j + (k_0 - j_0)$ (contoh diatas $k = 3 + (3 - 2) = 4$). Perhatikan bahwa k adalah bilangan dengan $h + 1$ bit, jadi

$$k < 2^{h+1} = 4(2^{h-1}) \leq 4k_0.$$

Teorema 99 *Jika n adalah bilangan komposit ganjil dan d adalah faktor non-trivial dari n ($d|n$ dan $1 < d < n$) dengan $d < \sqrt{n}$, maka metode Rho yang telah dimodifikasi akan menemukan faktor d dengan probabilitas $1 - e^{-\lambda}$ menggunakan $C\sqrt{\lambda}\sqrt[4]{n}\log^3 n$ operasi bit, dimana C adalah suatu konstan.*

Untuk membuktikan teorema 99, kita umpamakan bahwa $\gcd(x - y, n)$ (dengan $n > x - y$) dapat dikalkulasi menggunakan algoritma Euclid dengan $C_1 \log^3 n$ operasi bit, dimana C_1 adalah suatu konstan (ini berdasarkan pengamatan bahwa algoritma Euclid melakukan $O(\log(n))$ operasi pembagian dan setiap pembagian memerlukan $O(\log^2 n)$ operasi bit, jadi secara total diperlukan $O(\log^3 n)$ operasi bit). Kita juga umpamakan bahwa komputasi $f(x)$ modulo n memerlukan $C_2 \log^2 n$ operasi bit, dimana C_2 adalah suatu konstan (karena komputasi *polynomial* memerlukan konstan perkalian dengan setiap perkalian membutuhkan $O(\log^2 n)$ operasi bit, dan operasi modulo atau pembagian juga membutuhkan $O(\log^2 n)$ operasi bit). Jika k_0 merupakan indeks pertama yang menghasilkan

$$x_{k_0} \equiv x_{j_0} \pmod{d}$$

untuk suatu j_0 dengan $j_0 < k_0$, maka metode Rho yang telah dimodifikasi akan menemukan d saat memeriksa x_k dimana $k < 4k_0$. Sebetulnya ada kemungkinan bahwa $\gcd(x_k - x_j, n)$ menghasilkan sesuatu yang lebih besar dari d, dengan kata lain $\gcd((x_k - x_j)/d, n/d) > 1$, tetapi kemungkinan ini sangat kecil dan dapat diperhitungkan dengan membuat C cukup besar. Jadi jika d ditemukan saat memeriksa x_k, maka penemuan d memerlukan tidak lebih dari $4k_0(C_1 \log^3 n + C_2 \log^2 n)$ operasi bit. Jika $k_0 \leq 1 + \sqrt{2\lambda d}$ maka untuk menemukan d diperlukan

$$\begin{aligned} &< 4(1 + \sqrt{2\lambda d})(C_1 \log^3 n + C_2 \log^2 n) \\ &< 4(1 + \sqrt{2}\sqrt{\lambda}\sqrt[4]{n})(C_1 \log^3 n + C_2 \log^2 n) \end{aligned}$$

operasi bit. Jika kita buat

$$C = 4\sqrt{2}\left(\frac{(C_1 + C_2)}{\sqrt{2}\sqrt{\lambda}\sqrt[4]{n}} + (C_1 + C_2)\right)$$

kita dapatkan bahwa d ditemukan dengan menggunakan tidak lebih dari

$$C\sqrt{\lambda}\sqrt[4]{n}\log^3 n$$

operasi bit dengan probabilitas $1 - e^{-\lambda}$ (menurut teorema 98). Selesailah pembuktian teorema 99. Tentunya teorema 99 hanya berlaku jika f yang digunakan merupakan suatu *"average map."* Dalam prakteknya, beberapa *polynomial* yang sangat sederhana bersifat *"average map,"* termasuk $f(x) = x^2 + 1$.

Teorema 98 memberikan gambaran mengenai kompleksitas algoritma untuk metode Rho. Dalam bidang teori kompleksitas, pengukuran kompleksitas algoritma biasanya didasarkan pada besarnya input dalam ukuran bit. Jika r merupakan banyaknya bit dalam n, maka kompleksitas algoritma untuk metode Rho diperkirakan sekitar

$$O(e^{C\sqrt{r}})$$

dimana $C = \frac{1}{4}log2$. Jadi untuk r yang sangat besar, metode Rho lebih lambat dari algoritma dengan kompleksitas *polynomial-time*[1], meskipun tidak selambat algoritma dengan kompleksitas *exponential-time*.

14.2 Fermat Factorization

Jika suatu bilangan komposit ganjil n merupakan produk dari dua bilangan yang berdekatan, jadi $n = pq$ dengan $p \geq q > 0$, $p - q$ tidak terlalu besar, dan p dan q keduanya ganjil, maka p dan q dapat dicari dengan mudah menggunakan *Fermat factorization*. Teknik ini didasarkan pada fakta bahwa jika

$$s = \tfrac{p+q}{2}, \quad t = \tfrac{p-q}{2},$$

jadi

$$p = s + t, \quad q = s - t,$$

maka

$$\begin{aligned} n &= pq \\ &= (s+t)(s-t) \\ &= s^2 - t^2. \end{aligned}$$

Sebelum kita lanjutkan pembahasan *Fermat factorization*, kita buktikan dahulu teorema berikut.

Teorema 100 *Jika $p \geq q \geq 0$ dimana p dan q adalah bilangan bulat, maka*

$$p^2 + q^2 \geq 2pq$$

[1]Istilah *super-polynomial-time* kerap digunakan untuk kompleksitas yang lebih lambat dari *polynomial-time*.

Kita buktikan teorema 100 menggunakan induksi pada p dengan *base case* $p = q$ (jadi sebenarnya induksi adalah pada $p - q$). Untuk $p = q$ kita dapatkan

$$\begin{aligned} p^2 + q^2 &= 2q^2 \\ &= 2pq, \end{aligned}$$

jadi $p^2 + q^2 \geq 2pq$. Untuk langkah induksi kita dapatkan

$$\begin{aligned} (p+1)^2 + q^2 &= p^2 + 2p + 1 + q^2 \\ &\geq 2pq + 2p + 1 \qquad \text{(menggunakan hipotesis induksi)} \\ &> 2pq + 2q \qquad \text{(karena } p \geq q\text{)} \\ &= 2(p+1)q. \end{aligned}$$

Jadi $(p+1)^2+q^2 \geq 2(p+1)q$ dan selesailah pembuktian teorema 100 menggunakan induksi. Teorema 100 kita gunakan untuk menunjukkan bahwa $(p+q)/2 \geq \sqrt{n}$ dengan menunjukkan bahwa $(p+q)^2/4 \geq n$:

$$\begin{aligned} \frac{(p+q)^2}{4} &= \frac{p^2 + 2pq + q^2}{4} \\ &\geq \frac{4pq}{4} \qquad \text{(menggunakan teorema 100)} \\ &= n. \end{aligned}$$

Jadi jika p dan q berdekatan, maka $t = (p-q)/2$ merupakan bilangan yang kecil dan $s = (p+q)/2 \geq \sqrt{n}$ dan perbedaan antara s dan $\sqrt{n}$ tidak terlalu besar. Jadi kita dapat mencari s mulai dari $s = \lfloor\sqrt{n}\rfloor + 1$, lalu $s = \lfloor\sqrt{n}\rfloor + 2$, dan seterusnya hingga kita temukan nilai $s^2 - n$ yang merupakan *perfect square* (s^2-n merupakan kuadrat) yang kita jadikan nilai untuk t^2 (karena $n = s^2-t^2$). Sebagai contoh kita coba uraikan 200819:

$$\begin{array}{l|l|l} s = \lfloor\sqrt{200819}\rfloor + 1 = 449 & 449^2 - 200819 = 782 & \text{bukan kuadrat;} \\ s = \lfloor\sqrt{200819}\rfloor + 2 = 450 & 450^2 - 200819 = 1681 = 41^2 & t = 41. \end{array}$$

Kita dapatkan $p = s + t = 450 + 41 = 491$ dan $q = s - t = 450 - 41 = 409$, jadi $n = pq = 491 \cdot 409 = 200819$. Jadi jika suatu bilangan komposit ganjil n merupakan produk dari dua bilangan ganjil p dan q yang berdekatan, maka p dan q dapat ditemukan dengan mudah, termasuk jika p dan q merupakan bilangan prima ganjil yang berdekatan. Jika n bukan merupakan produk dari dua bilangan ganjil yang berdekatan, maka *Fermat factorization* akan mencoba banyak nilai s sebelum menemukan nilai s yang mendapatkan faktor, jadi bukan merupakan suatu algoritma yang efisien.

14.3 Metode Dixon

Fermat factorization secara umum bukan merupakan algoritma yang efisien untuk menguraikan bilangan yang sangat besar, tetapi lebih berupa konsep bahwa

suatu bilangan komposit ganjil merupakan perbedaan dari kuadrat (*difference of squares*). Sekitar tahun 1920an, Maurice Kraitchik menyarankan ide bahwa yang dicari adalah perbedaan kuadrat modulo bilangan yang diuraikan. Jadi kita bukan mencari s dan t yang menghasilkan $n = s^2 - t^2$, tetapi cari s dan t yang menghasilkan

$$s^2 \equiv t^2 \pmod{n}.$$

Ada kalanya ini akan menghasilkan

$$s \equiv \pm t \pmod{n}$$

yang membuat kita harus mencari pasangan s dan t yang lain. Akan tetapi jika

$$s \not\equiv \pm t \pmod{n}$$

maka kita dapatkan faktor dari n dengan mengkalkulasi

$$\gcd(s+t, n) \qquad \text{atau} \qquad \gcd(s-t, n).$$

Ini karena n membagi $s^2 - t^2 = (s+t)(s-t)$ tetapi n tidak membagi $s+t$ atau $s-t$ (karena $s \not\equiv \pm t \pmod{n}$), jadi $a = \gcd(s+t, n)$ merupakan faktor *non-trivial* dari n ($a|n$ dan $1 < a < n$). Kita juga dapatkan $b = n/a$ membagi $\gcd(s-t, n)$. Sebagai contoh, dengan $n = 4633$, $s = 118$ dan $t = 5$, kita temukan

$$118^2 \equiv 5^2 \pmod{4633} \qquad \text{dan} \qquad 118 \not\equiv \pm 5 \pmod{4633},$$

jadi

$$\gcd(118+5, 4633) = 41 \qquad \text{dan} \qquad \gcd(118-5, 4633) = 113$$

merupakan faktor dari 4633.

Ide inilah yang menjadi dasar dari berbagai metode modern untuk menguraikan bilangan yang besar, termasuk metode Dixon, metode *continued fraction*, metode *quadratic sieve* dan metode *number field sieve*.

Contoh diatas tidak menunjukkan bagaimana kita menemukan $s = 118$ yang jika dikuadratkan (s^2) mempunyai *least non-negative residue*[2] modulo 4633 yaitu 25 yang juga merupakan kuadrat ($t^2 = 5^2$). Jika n merupakan bilangan yang sangat besar, maka probabilitas bahwa suatu bilangan yang dipilih secara acak jika dikuadratkan mempunyai *least non-negative residue* modulo n yang juga merupakan kuadrat, adalah sangat kecil.

Metode Dixon termasuk metode yang mencoba menemukan s dan t secara sistematis menggunakan *factor base*. Ide yang digunakan untuk mendapatkan s adalah untuk memilih beberapa b_i dimana $b_i^2 \bmod n$ merupakan

[2]Di bagian ini kita juga akan menggunakan konsep *least absolute residue* jadi kita harus bedakan kedua konsep.

produk dari beberapa pemangkatan bilangan prima kecil ($p_1^{\alpha_1} p_2^{\alpha_2} \dots p_m^{\alpha_m}$ dimana $p_1, p_2, \dots, p_m$ semua merupakan bilangan prima kecil), dan produk dari $b_i \bmod n$ menghasilkan s. Sebelum membahas metode secara rinci, kita perlu definisikan dahulu konsep *least absolute residue*. Suatu bilangan b merupakan *least absolute residue* dari a modulo n, dimana n merupakan bilangan ganjil, jika:

$$a \equiv b \pmod{n} \qquad \text{dan} \qquad -(n-1)/2 \leq b \leq (n-1)/2.$$

Sekarang kita definisikan konsep *factor base*:

Definisi 47 (Factor Base) *Suatu factor base merupakan himpunan bilangan prima yang berbeda* $B = \{p_1, p_2, \dots, p_h\}$, *kecuali* p_1 *dapat berupa* -1.

Kuadrat dari bilangan bulat b, b^2 merupakan *B-number* modulo n jika *least absolute residue* dari $b^2 \bmod n$ dapat diuraikan menjadi produk dari elemen-elemen B.

Definisi 48 (B-number)

$$b^2 \equiv t \pmod{n}$$

merupakan B-number jika t *adalah least absolute residue modulo* n, *dan*

$$t = \prod_{i=1}^{h} p_i^{\alpha_i}$$

dimana $B = \{p_1, p_2, \dots, p_h\}$ *adalah factor base dan* $\alpha_i \geq 0$ *untuk setiap* i.

Sebagai contoh, dengan $n = 4633$ dan $B = \{-1, 2, 3\}$, maka 67^2, 68^2 dan 69^2 masing-masing merupakan *B-number* karena

$$\begin{array}{lcrcl} 67^2 & \equiv & -144 \pmod{4633} & = & -1 \cdot 2^4 \cdot 3^2; \\ 68^2 & \equiv & -9 \pmod{4633} & = & -1 \cdot 3^2; \\ 69^2 & \equiv & 128 \pmod{4633} & = & 2^7. \end{array}$$

Untuk menentukan apakah suatu kuadrat merupakan *B-number* tentunya perlu penguraian, meskipun setiap faktor harus merupakan pemangkatan dari bilangan dalam *factor base*. Penguraian ini dapat dilakukan dengan *trial division* (mencoba membagi) menggunakan elemen-elemen *factor base*. Ada cara untuk mempercepat proses ini, misalnya menggunakan metode Pollard-Strassen, akan tetapi kita tidak akan bahas cara-cara untuk mempercepat proses ini.

Jika untuk $1 \leq j \leq m$, setiap t_j merupakan *least absolute residue* dari $b_j^2 \bmod n$ dimana b_j^2 adalah *B-number*, dan produk dari semua t_j,

$$\prod_{j=1}^{m} t_j = \prod_{j=1}^{m} \prod_{i=1}^{h} p_{i,j}^{\alpha_{i,j}}$$

mempunyai setiap $\alpha_{i,j}$ berupa bilangan genap, maka produk semua t_j merupakan kuadrat dari

$$t = \prod_{i=1}^{h} p_i^{\gamma_i}$$

dimana $\gamma_i = \frac{1}{2}\sum_{j=1}^{m} \alpha_{i,j}$. Kita buat juga s sebagai produk dari semua b_j:

$$s = \prod_{j=1}^{m} b_j$$

dan kita dapatkan $s^2 \equiv t^2 \pmod{n}$. Jika $s \not\equiv \pm t \pmod{n}$ maka kita dapatkan faktor *non-trivial* dari n menggunakan $\gcd(s+t, n)$ atau $\gcd(s-t, n)$ seperti diawal bagian ini. Jika $s \equiv \pm t \pmod{n}$ maka kita harus cari produk dari kumpulan lain. Bagaimana kita mencari kumpulan b_i untuk *factor base*? Tiga cara untuk mencari *factor base* akan dibahas dalam buku ini:

1. tentukan *factor base* sebagai beberapa bilangan prima pertama dan -1, dan kemudian cari beberapa b_i secara acak, dimana b_i^2 merupakan *B-number*;
2. cari beberapa b_i yang mempunyai *least absolute residue* dari $b_i^2 \bmod n$ yang kecil, dan kemudian cari *factor base* sehingga setiap b_i^2 merupakan *B-number*; atau
3. tentukan *factor base* sebagai semua bilangan prima $p \leq P$ dimana $\left(\frac{n}{p}\right) = 1$ jika p ganjil, dan P adalah suatu batas yang dipilih secara optimal.

Metode Dixon menggunakan cara pertama dalam menentukan *factor base*; metode *continued fraction* (lihat bagian 14.4) menggunakan cara kedua; dan metode *quadratic sieve* (lihat bagian 14.5) menggunakan cara ketiga.

Setelah *factor base* dan beberapa b_i terpilih, kita pilih *subset* dari kumpulan b_i yang jika dikalikan menghasilkan produk dari pemangkatan genap elemen-elemen *factor base* (setiap elemen dipangkatkan dengan suatu bilangan genap). Karena kita hanya perlu fokus pada kegenapan pangkat elemen *factor base*, kita gunakan konsep ruang vektor $\mathbf{F}_2^h$. Untuk suatu bilangan komposit ganjil n, *factor base* B dengan h elemen, suatu *B-number* b_i^2, dengan $t_i = \prod_{j=1}^{h} p_j^{\alpha_j}$ sebagai *least absolute residue* dari $b_i \bmod n$, maka vektor $\overrightarrow{\epsilon_i}$ didefinisikan sebagai berikut

Definisi 49

$$\overrightarrow{\epsilon_i} = (\epsilon_{i,1}, \epsilon_{i,2}, \ldots, \epsilon_{i,h})$$

dengan

$$\epsilon_{i,j} = \begin{cases} 0 & \textit{jika } \alpha_j \textit{ genap} \\ 1 & \textit{jika } \alpha_j \textit{ ganjil} \end{cases}$$

untuk $1 \leq j \leq h$.

Sebagai contoh, dengan $n = 4633$ dan $B = \{-1, 2, 3\}$, kita dapatkan:

$$\begin{array}{l|c|l} b_1 = 67 & t_1 = -1 \cdot 2^4 \cdot 3^2 & \overrightarrow{\epsilon_1} = (1,0,0), \\ b_2 = 68 & t_2 = -1 \cdot 3^2 & \overrightarrow{\epsilon_2} = (1,0,0), \\ b_3 = 69 & t_3 = 2^7 & \overrightarrow{\epsilon_3} = (0,1,0). \end{array}$$

Jadi mencari subset dari berbagai b_i yang menghasilkan produk yang diinginkan (setiap elemen $p_i \in B$ dipangkatkan dengan bilangan genap) sama dengan mencari kombinasi linear dari berbagai $\overrightarrow{\epsilon_i}$ yang menghasilkan $(0, 0, \ldots, 0)$. Ini dapat dilakukan, jika terdapat *linear dependence* diantara berbagai $\overrightarrow{\epsilon_i}$, menggunakan teknik *row reduction* dari aljabar linear. Jika banyaknya b_i melebihi h $(> h)$ maka dijamin terdapat *linear dependence* diantara berbagai $\overrightarrow{\epsilon_i}$.

Tingkat kesuksesan dari metode Dixon sangat tergantung pada mudahnya mencari berbagai b_i dengan *least absolute residue* dari $b_i^2 \bmod n$ yang dapat diuraikan sebagai produk dari pemangkatan elemen-elemen *factor base*. Perlu diperhatikan bahwa b_i yang menghasilkan $\overrightarrow{\epsilon_i} = (0, 0, \ldots, 0)$ tidak ada gunanya karena menghasilkan *trivial congruence* $s^2 \equiv s^2 \pmod{n}$, jadi $b_i < \sqrt{n/2}$ tidak ada gunanya. Strategi yang kerap digunakan adalah mencari b_i yang dekat dengan $\sqrt{kn}$ untuk berbagai kelipatan k yang kecil. Kompleksitas metode Dixon, jika diimplementasikan dengan optimal, diestimasikan adalah

$$O(e^{C\sqrt{\log n \log\log n}}).$$

Karena terlalu rumit, kita tidak akan bahas bagaimana estimasi kompleksitas diatas didapatkan. Untuk yang ingin mengetahui bagaimana estimasi didapatkan, dan juga untuk mempelajari metode Dixon lebih lanjut, dipersilahkan untuk membaca [dix81]. Seperti halnya dengan metode Rho, metode Dixon mempunyai kompleksitas *super-polynomial-time* meskipun tidak seburuk *exponential-time*, akan tetapi kompleksitas metode Dixon lebih baik dibandingkan dengan kompleksitas metode Rho.

14.4 Metode Continued Fraction

Metode Dixon efektif jika terdapat cara yang efisien untuk mencari b_i antara 1 dan n dengan *least absolute residue* dari $b_i^2 \bmod n$ berupa produk dari bilangan-bilangan prima yang kecil. Tingkat kesuksesan akan semakin besar jika $b_i^2 \bmod n$ adalah bilangan kecil. Metode *continued fraction* adalah metode untuk mencari berbagai b_i dimana $|b_i^2 \bmod n| < 2\sqrt{n}$. Metode ini detemukan oleh Morisson dan Brillhart [mor75].

Kita mulai dengan penjelasan konsep *continued fraction*. Suatu bilangan

nyata x dapat dituliskan dalam bentuk *continued fraction* sebagai berikut:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ldots + \cfrac{1}{a_i + x_{i+1}}}}$$

dimana

- $x_0 = x$, dan
- untuk $i \geq 0$, $a_i = \lfloor x_i \rfloor$ adalah bilangan bulat terbesar yang tidak lebih besar dari x_i, dan $x_{i+1} = \frac{1}{x_i - a_i}$.

Deretan $a_0, a_1, a_2, \ldots a_i$ diatas dapat dikomputasi menggunakan algoritma $CFA(x)$ (*continued fraction algorithm*) sebagai berikut:

1. $i \leftarrow 0$
2. $x_0 \leftarrow x$
3. $a_0 \leftarrow \lfloor x_0 \rfloor$
4. output(a_0)
5. jika $(x_i = a_i)$ berhenti disini
6. $x_{i+1} \leftarrow \frac{1}{x_i - a_i}$
7. $i \leftarrow i + 1$
8. $a_i \leftarrow \lfloor x_i \rfloor$
9. output(a_i)
10. kembali ke langkah 5

Tidak terlalu sulit untuk membuktikan bahwa algoritma $CFA(x)$ akan berhenti pada suatu $i \geq 0$ jika dan hanya jika x merupakan bilangan rasional. Mari kita bahas pembuktiannya. Jika algoritma $CFA(x)$ berhenti pada suatu $i \geq 0$, maka sangat jelas bahwa

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ldots + \cfrac{1}{a_i}}}$$

adalah bilangan rasional karena $a_0, a_1, \ldots a_i$ semua merupakan bilangan bulat. Sebaliknya jika x merupakan bilangan rasional, maka setiap x_i dengan $i \geq 0$

merupakan bilangan rasional (karena x_0 rasional dan jika x_i rasional maka $x_{i+1} = \frac{1}{x_i - a_i}$ juga rasional untuk $i \geq 0$), sebut saja

$$x_i = u_i / u_{i+1}$$

dimana u_i dan u_{i+1} merupakan bilangan bulat. Jadi $a_i = \lfloor x_i \rfloor = \lfloor u_i / u_{i+1} \rfloor$ dan

$$\begin{aligned} x_{i+1} &= \frac{1}{x_i - a_i} \\ &= \frac{1}{u_i/u_{i+1} - \lfloor u_i/u_{i+1} \rfloor} \\ &= \frac{u_{i+1}}{u_i - u_{i+1}\lfloor u_i/u_{i+1} \rfloor} \\ &= \frac{u_{i+1}}{u_i \bmod u_{i+1}}. \end{aligned}$$

Jadi dari x_i ke x_{i+1}, pasangan (u_i, u_{i+1}) berubah menjadi $(u_{i+1}, u_i \bmod u_{i+1})$. Tetapi ini persis sama dengan transformasi yang dilakukan algoritma Euclid (lihat 3.4) dimana pasangan (r_i, r_{i+1}) berubah menjadi $(r_{i+1}, r_i \bmod r_{i+1})$ (operasi mod sama dengan *residue*). Karena kita telah buktikan bahwa algoritma Euclid selalu berhenti, maka kita telah buktikan juga bahwa algoritma $CFA(x)$ selalu berhenti jika x adalah bilangan rasional. Jadi telah kita buktikan bahwa algoritma $CFA(x)$ selalu berhenti jika dan hanya jika x rasional.

Jika x merupakan bilangan irasional, maka

$$\frac{b_i}{c_i} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ldots + \cfrac{1}{a_i}}} \tag{14.1}$$

dinamakan juga konvergen ke-i (*i-th convergent*) dari *continued fraction* dari x. Sangat jelas bahwa konvergen $\frac{b_i}{c_i}$ adalah bilangan rasional.

Pembaca mungkin bertanya bagaimana kita gunakan algoritma $CFA(x)$ terhadap bilangan irasional untuk mencari konvergen. Nanti kita kan bahas bagaimana caranya mencari konvergen untuk akar dari bilangan bulat, untuk sekarang anggap saja algoritma $CFA(x)$ sebagai algoritma konseptual, jadi bukan suatu algoritma yang dapat digunakan secara mentah.

Teorema 101 (Definisi Konvergen) *Dengan menggunakan notasi konvergen seperti diatas, maka*

$$\begin{aligned} \frac{b_0}{c_0} &= \frac{a_0}{1} \qquad (b_0 = a_0, c_0 = 1); \\ \frac{b_1}{c_1} &= \frac{a_0 a_1 + 1}{a_1} \qquad (b_1 = a_0 a_1 + 1, c_1 = a_1); \text{ dan} \end{aligned}$$

$$\frac{b_i}{c_i} = \frac{a_i b_{i-1} + b_{i-2}}{a_i c_{i-1} + c_{i-2}} \qquad (b_i = a_i b_{i-1} + b_{i-2}, c_i = a_i c_{i-1} + c_{i-2}) \text{ jika } i \geq 2.$$

Kita buktikan teorema ini dengan induksi, dengan tidak menggunakan asumsi bahwa setiap a_j berupa bilangan bulat, hanya bahwa persamaan 14.1 berlaku untuk konvergen. Untuk $i = 0$ dan $i = 1$ sangat jelas bahwa

$$\begin{aligned}
\frac{b_0}{c_0} &= a_0 \\
&= \frac{a_0}{1}; \\
\frac{b_1}{c_1} &= a_0 + \frac{1}{a_1} \\
&= \frac{a_0 a_1 + 1}{a_1}.
\end{aligned}$$

Untuk $i = 2$ kita dapatkan

$$\begin{aligned}
\frac{b_i}{c_i} &= \frac{b_2}{c_2} \\
&= a_0 + \frac{1}{a_1 + \frac{1}{a_2}} \\
&= a_0 + \frac{a_2}{a_1 a_2 + 1} \\
&= \frac{a_0 a_1 a_2 + a_0 + a_2}{a_1 a_2 + 1} \\
&= \frac{a_2(a_0 a_1 + 1) + a_0}{a_2 c_1 + c_0} \\
&= \frac{a_2 b_1 + b_0}{a_2 c_1 + c_0} \\
&= \frac{a_i b_{i-1} + b_{i-2}}{a_i c_{i-1} + c_{i-2}}.
\end{aligned}$$

Untuk langkah induksi, kita umpamakan bahwa teorema 101 berlaku untuk i, jadi kita mempunyai

$$\frac{b_i}{c_i} = \frac{a_i b_{i-1} + b_{i-2}}{a_i c_{i-1} + c_{i-2}} \qquad (b_i = a_i b_{i-1} + b_{i-2}, c_i = a_i c_{i-1} + c_{i-2})$$

sebagai hipotesis induksi, dan kita harus buktikan bahwa teorema berlaku untuk $i+1$. Konvergen ke $i+1$ didapat dengan menukar a_i dengan $a_i + \frac{1}{a_{i+1}}$ pada konvergen ke i, jadi dengan menggunakan hipotesis induksi, kita dapatkan

$$\frac{b_{i+1}}{c_{i+1}} = \frac{(a_i + \frac{1}{a_{i+1}})b_{i-1} + b_{i-2}}{(a_i + \frac{1}{a_{i+1}})c_{i-1} + c_{i-2}}$$

$$\begin{aligned} &= \frac{a_{i+1}(a_i b_{i-1} + b_{i-2}) + b_{i-1}}{a_{i+1}(a_i c_{i-1} + c_{i-2}) + c_{i-1}} \\ &= \frac{a_{i+1} b_i + b_{i-1}}{a_{i+1} c_i + c_{i-1}}. \end{aligned}$$

Selesailah pembuktian teorema 101.

Teorema 102

$$b_i c_{i-1} - b_{i-1} c_i = (-1)^{i-1} \qquad \textit{untuk } x \geq 1.$$

Kita buktikan teorema 102 dengan induksi. Untuk $i = 1$ kita dapatkan

$$\begin{aligned} b_i c_{i-1} - b_{i-1} c_i &= b_1 c_0 - b_0 c_1 \\ &= (a_0 a_1 + 1) - a_0 a_1 \\ &= 1 \\ &= (-1)^0 \\ &= (-1)^{i-1}. \end{aligned}$$

Untuk langkah induksi, dengan hipotesis

$$b_i c_{i-1} - b_{i-1} c_i = (-1)^{i-1},$$

kita dapatkan

$$\begin{aligned} b_{i+1} c_i - b_i c_{i+1} &= (a_{i+1} b_i + b_{i-1}) c_i - b_i (a_{i+1} c_i + c_{i-1}) \text{ (dari teorema 101)} \\ &= a_{i+1} b_i c_i + b_{i-1} c_i - a_{i+1} b_i c_i - b_i c_{i-1} \\ &= b_{i-1} c_i - b_i c_{i-1} \\ &= -(-1)^{i-1} \text{ (menggunakan hipotesis induksi)} \\ &= (-1)^i. \end{aligned}$$

Selesailah pembuktian teorema 102.

Teorema 103 *Menggunakan teorema 101 sebagai definisi konvergen, maka*

$$\gcd(b_i, c_i) = 1.$$

Untuk $i = 0$ pembuktian teorema 103 sangat mudah karena $\gcd(a_0, 1) = 1$. Untuk $i \geq 1$, dari teorema 102 kita dapat menyimpulkan bahwa pembagi b_i dan c_i harus membagi $(-1)^i$ yang mempunyai nilai ± 1, jadi $\gcd(b_i, c_i) = 1$. Selesailah pembuktian teorema 103.

Jika kita bagi persamaan dalam teorema 102 dengan $c_i c_{i-1}$ maka kita dapatkan

$$\frac{b_i}{c_i} - \frac{b_{i-1}}{c_{i-1}} = \frac{(-1)^i}{c_i c_{i-1}}.$$

Jadi konvergen berosilasi dengan amplitudo yang mengecil, semakin besar i. Mari kita tunjukkan bahwa semakin besar i, konvergen semakin mendekati x. Perhatikan bahwa kita bisa mendapatkan x dari konvergen ke $i+1$ dengan menukar a_{i+1} dengan x_{i+1}. Jadi menggunakan teorema 101 (ingat bahwa pembuktian teorema 101 tidak mengasumsikan bahwa a_i adalah bilangan bulat) kita dapatkan

$$\begin{aligned} x &= \frac{b_i/x_{i+1} + b_{i-1}}{c_i/x_{i+1} + c_{i-1}} \\ &= \frac{b_i + x_{i+1}b_{i-1}}{c_i + x_{i+1}c_{i-1}}, \end{aligned}$$

Jika $\mathbf{u} = (c_i, b_i)$ dan $\mathbf{v} = (c_{i-1}, b_{i-1})$ dianggap sebagai vektor dalam bidang (dua dimensi) maka kedua vektor berada pada kuadran yang sama, dan kemiringan dari vektor $\mathbf{u} + x_{i+1}\mathbf{v} = (c_i + x_{i+1}c_{i-1}, b_i + x_{i+1}b_{i-1})$ adalah

$$\frac{b_i + x_{i+1}b_{i-1}}{c_i + x_{i+1}c_{i-1}} = x$$

dan berada antara kemiringan $\mathbf{u}$ yaitu $\frac{b_i}{c_i}$ dan kemiringan $\mathbf{v}$ yaitu $\frac{b_{i-1}}{c_{i-1}}$. Jadi deretan $\frac{b_i}{c_i}$ (konvergen) dengan $i = 0, 1, 2, \ldots$ berosilasi antara bawah dan atas x dan mendekati x secara monoton (jadi limit dari konvergen adalah x).

Teorema 104 *Jika $x > 1$ merupakan bilangan irasional dengan konvergen $\frac{b_i}{c_i}$ untuk $i = 0, 1, 2, \ldots$, maka untuk setiap i, $|b_i^2 - x^2c_i^2| < 2x$.*

Untuk membuktikan teorema 104 kita gunakan fakta bahwa nilai x berada diantara $\frac{b_i}{c_i}$ dan $\frac{b_{i+1}}{c_{i+1}}$, dan karena menurut teorema 102 perbedaan antara kedua konvergen adalah $1/c_ic_{i+1}$, maka kita dapatkan

$$\begin{aligned} |b_i^2 - x^2c_i^2| &= c_i^2|x - \frac{b_i}{c_i}||x + \frac{b_i}{c_i}| \\ &< c_i^2\frac{1}{c_ic_{i+1}}(x + (x + \frac{1}{c_ic_{i+1}})). \end{aligned}$$

Berarti

$$\begin{aligned} |b_i^2 - x^2c_i^2| - 2x &< 2x(-1 + \frac{c_i}{c_{i+1}} + \frac{1}{2xc_{i+1}^2}) \\ &< 2x(-1 + \frac{c_i}{c_{i+1}} + \frac{1}{c_{i+1}}) \\ &< 2x(-1 + \frac{c_{i+1}}{c_{i+1}}) \\ &= 0. \end{aligned}$$

Jadi $|b_i^2 - x^2c_i^2| < 2x$ dan selesailah pembuktian teorema 104.

Teorema 105 *Jika n merupakan bilangan bulat positif yang bukan merupakan kuadrat (n bukan perfect square) dan deretan* $\frac{b_i}{c_i}$ *dengan* $i = 0, 1, 2, \ldots$ *merupakan deretan konvergen dari* $\sqrt{n}$*, maka least absolute residue dari* $b_i^2 \bmod n$ *lebih kecil dari* $2\sqrt{n}$*.*

Teorema 105 didapat dari teorema 104 dengan $x = \sqrt{n}$. Teorema 105 menjadi dasar dari penggunaan metode *continued fraction* untuk menguraikan bilangan, dengan mengatakan bahwa terdapat deretan b_i dengan kuadrat yang mempunyai *residue* yang kecil.

Sekarang kita bahas akar dari bilangan bulat. Kita akan tunjukkan bahwa jika suatu bilangan bulat n bukan berupa *perfect square* (kuadrat dari bilangan bulat), maka $\sqrt{n}$ adalah bilangan irasional.

Teorema 106 *Jika n adalah bilangan bulat positif yang bukan berupa kuadrat bilangan bulat juga (n bukan perfect square) maka* $\sqrt{n}$ *adalah bilangan irasional.*

Pembuktian teorema ini sangat mudah, karena jika $\sqrt{n}$ adalah bilangan rasional, sebut saja $\frac{a}{b}$, maka $\frac{a^2}{b^2} = n$ adalah bilangan bulat (a^2 dapat dibagi b^2), dan $\frac{a}{b}$ juga bilangan bulat (karena jika a^2 dapat dibagi oleh b^2 maka a dapat dibagi oleh b). Jadi jika $\sqrt{n}$ adalah bilangan rasional maka n berupa *perfect square*. Sebaliknya jika n berupa *perfect square* maka $\sqrt{n}$ adalah bilangan bulat yang tentu saja juga rasional. Berarti $\sqrt{n}$ merupakan bilangan rasional jika dan hanya jika n merupakan *perfect square*. Jadi jika n bukan *perfect square* maka $\sqrt{n}$ adalah bilangan irasional. Selesailah pembuktian kita.

Kita lanjutkan dengan penjelasan mengenai kalkulasi $CFA(x)$ jika x merupakan akar dari bilangan bulat, jadi $x = \sqrt{n}$ dimana n adalah suatu bilangan bulat. Jika n merupakan *perfect square*, maka algoritme $CFA(x)$ langsung berhenti karena $x = \sqrt{n}$ merupakan bilangan bulat. Jika n bukan merupakan *perfect square* maka menurut teorema 106, $\sqrt{n}$ merupakan bilangan irasional dan algoritma $CFA(x)$ dapat berjalan terus tanpa berahir. Yang perlu dijelaskan adalah bagaimana melakukan langkah 3, 6 dan 8 dari algoritma $CFA(x)$. Untuk langkah 3, $x_0 = \sqrt{n}$, jadi $a_0 = \lfloor x_0 \rfloor$ merupakan bilangan bulat terbesar yang jika dikuadratkan tidak melebihi n ($a_0^2 < n$ dan $(a_0 + 1)^2 > n$). Cara yang cukup efisien untuk mencari a_0 adalah dengan teknik *binary search* pada semua bilangan antara 1 dan n. Pertama kali langkah 6 dilakukan, $i = 0$, jadi

$$\begin{aligned} x_1 &= \frac{1}{\sqrt{n} - a_0} \\ &= \frac{\sqrt{n} + a_0}{n - a_0^2} \\ &= \frac{\sqrt{n} + a_0}{m} \end{aligned}$$

untuk suatu bilangan bulat $m > 0$ (karena $n > a_0^2$). Tidak terlalu sulit untuk

melihat bahwa

$$\begin{aligned} a_1 &= \lfloor x_1 \rfloor \\ &= \lfloor \frac{\lfloor \sqrt{n} \rfloor + a_0}{m} \rfloor \end{aligned}$$

dan

$$\begin{aligned} x_1 - a_1 &= \frac{\sqrt{n} + a_0 - km}{m} \\ &= \frac{\sqrt{n} - j}{m} \end{aligned}$$

untuk suatu bilangan bulat $j = km - a_0$. Jadi

$$\begin{aligned} x_2 &= \frac{1}{x_1 - a_1} \\ &= \frac{m}{\sqrt{n} - j} \end{aligned}$$

yang dapat dikalikan dengan $\frac{\sqrt{n}+j}{\sqrt{n}+j}$ agar denominator menjadi bilangan bulat. Proses ini dapat diulang untuk mengkalkulasi $a_2, a_3, \ldots$ dan seterusnya.

Sekarang kita tiba pada penjelasan mengenai bagaimana metode *continued fraction* dapat digunakan untuk menguraikan bilangan bulat. Algoritma untuk menguraikan bilangan bulat menggunakan metode *continued fraction* adalah sebagai berikut:

1. $i \leftarrow 0$
2. $b_{-1} \leftarrow 1$, $x_0 \leftarrow \sqrt{n}$
3. $b_0 \leftarrow a_0 \leftarrow \lfloor x_0 \rfloor$
4. $x_{i+1} \leftarrow \frac{1}{x_i - a_i}$
5. $i \leftarrow i + 1$
6. $a_i \leftarrow \lfloor x_i \rfloor$
7. $b_i \leftarrow a_i b_{i-1} + b_{i-2} \pmod{n}$
8. kalkulasi $b_i^2 \bmod n$
9. kembali ke langkah 4

Setelah langkah-langkah 4 sampai dengan 9 diulang beberapa kali, kita periksa bilangan-bilangan hasil kalkulasi langkah 8. Kita buat *factor base* terdiri dari -1 dan semua bilangan prima yang merupakan faktor dari lebih dari satu $b_i^2 \bmod n$ atau merupakan faktor dengan pangkat genap dari hanya satu $b_i^2 \bmod n$. Selanjutnya, seperti halnya dengan metode Dixon, kita cari kumpulan dari b_i yang jika dikalikan menghasilkan produk dari pemangkatan genap elemen-elemen *factor base* dan menghasilkan

$$\begin{aligned} b^2 &\equiv c^2 \pmod{n} \text{ dan} \\ b &\not\equiv \pm c \pmod{n}. \end{aligned}$$

Jika kumpulan tidak dapat ditemukan, maka langkah-langkah 4 sampai dengan 9 harus diteruskan hingga kita mendapatkan kumpulan yang diinginkan.

Sebagai contoh penggunaan metode *continued fraction* mari kita coba uraikan 17873. Jika kita kalkulasi a_i, b_i dan $b_i^2 \bmod n$ untuk $i = 0, 1, 2, 3, 4, 5$ kita dapatkan tabel berikut:

i	0	1	2	3	4	5
a_i	133	1	2	4	2	3
b_i	133	134	401	1738	3877	13369
$b_i^2 \bmod n$	-184	83	-56	107	-64	161

Kita dapatkan *factor base* $B = \{-1, 2, 7, 23\}$ dan mendapatkan *B-number* untuk $i = 0, 2, 4, 5$ dengan $\overrightarrow{\epsilon_i}$ masing-masing $(1, 1, 0, 1)$, $(1, 1, 1, 0)$, $(1, 0, 0, 0)$ dan $(0, 0, 1, 1)$. Jika kita tambahkan vektor pertama, kedua dan keempat menggunakan aritmatika modulo 2, kita dapatkan vektor nol. Tetapi jika kita kalkulasi:

$$\begin{aligned} b &= \prod b_i \bmod n \\ &= 133 \cdot 401 \cdot 13369 \\ &\equiv 1288 \pmod{17873} \end{aligned}$$

dan

$$\begin{aligned} c &= \prod p_j^{\gamma_j} \\ &= 2^3 \cdot 7 \cdot 23 \\ &= 1288 \end{aligned}$$

kita dapatkan $b \equiv c \pmod{17873}$, jadi produk tidak bisa digunakan. Karena tidak ada produk lain yang dapat menghasilkan vektor nol dari tabel diatas, kita lanjutkan langkah-langkah 4 sampai dengan 9 untuk $i = 6, 7, 8$ untuk memperbesar tabel:

i	6	7	8
a_i	1	2	1
b_i	17246	12115	11488
$b_i^2 \bmod n$	-77	149	-88

Kita dapatkan *factor base* yang lebih besar yaitu $\{-1, 2, 7, 11, 23\}$ dan *B-number* untuk $i = 0, 2, 4, 5, 6, 8$ masing-masing dengan $\overrightarrow{\epsilon_i}$ sebagai berikut: $(1, 1, 0, 0, 1)$, $(1, 1, 1, 0, 0)$, $(1, 1, 0, 0, 0)$, $(0, 0, 1, 0, 1)$, $(1, 0, 1, 1, 0)$, $(1, 1, 0, 1, 0)$. Jika kita tambahkan vektor kedua, ketiga, kelima dan keenam menggunakan aritmatika modulo 2, kita dapatkan vektor nol, tetapi sekarang kita dapatkan $b = 7272$ dan $c = 4928$, jadi $\gcd(7272 + 4928, 17873) = 61$ merupakan faktor dari 17873. Kita dapatkan penguraian $17873 = 61 \cdot 293$.

Kompleksitas metode *continued fraction* seperti metode Dixon diestimasikan adalah

$$O(e^{C\sqrt{log\, n\, log\, log\, n}})$$

tetapi dengan konstan C yang lebih kecil. Untuk analisa yang cukup mendalam mengenai kompleksitas metode ini, silahkan membaca [pom82].

14.5 Metode Quadratic Sieve

Seperti halnya dengan metode *continued fraction*, metode *quadratic sieve* juga membuat himpunan *B-number* secara sistematis. Perbedaan terdapat pada cara menghasilkan himpunan. Metode *continued fraction* menggunakan algoritma $CFA(x)$ untuk menghasilkan himpunan *B-number*, sedangkan metode *quadratic sieve* menggunakan proses penyaringan untuk menghasilkan himpunan tersebut. Kita akan bahas metode *quadratic sieve* dan proses penyaringan yang digunakannya.

Untuk *factor base*, metode *quadratic sieve* menggunakan himpunan bilangan prima yang relatif kecil dan untuk bilangan prima p yang ganjil, p mematuhi persamaan

$$\left(\frac{n}{p}\right) = 1$$

jadi n (bilangan yang hendak diuraikan) adalah *quadratic residue* modulo p. Biasanya batas P untuk himpunan dipilih dengan nilai sekitar

$$e^{\sqrt{\log n \log \log n}}.$$

Jadi *factor base* terdiri dari 2 dan semua bilangan prima ganjil $p \leq P$ dimana n adalah *quadratic residue* modulo p.

Untuk membatasi banyaknya *B-number*, dipilih A yang lebih besar dari P tetapi tidak lebih dari pemangkatan kecil dari P, sebagai contoh:

$$P < A < P^2.$$

Kandidat untuk menghasilkan *B-number* yang diberi notasi t dibatasi sebagai berikut:

$$t = \lfloor\sqrt{n}\rfloor + 1, \lfloor\sqrt{n}\rfloor + 2, \ldots, \lfloor\sqrt{n}\rfloor + A.$$

Tentunya kandidat harus disaring lagi sehingga yang lolos benar merupakan *B-number*. Untuk proses penyaringan, kita gunakan tabel dengan kolom-kolom untuk t, $t^2 - n$, setiap bilangan dalam *factor base*, dan hasil pembagian $t^2 - n$ dengan faktor-faktor dalam *factor base*. Karena tujuan penyaringan adalah setiap $t^2 - n$ yang lolos adalah suatu *B-number*, maka $t^2 - n$ yang lolos harus merupakan produk dari faktor-faktor dalam *factor base*. Istilah matematikanya adalah $t^2 - n$ *smooth over the factor base*. Mari kita pelajari satu varian dari algoritma penyaringan untuk metode *quadratic sieve*.

1. Buat tabel dengan kolom-kolom untuk t, $t^2 - n$, x, dan bilangan-bilangan prima yang berada dalam *factor base*. Awalnya ada baris-baris untuk $t = \lfloor\sqrt{n}\rfloor + 1, \lfloor\sqrt{n}\rfloor + 2, \ldots, \lfloor\sqrt{n}\rfloor + A$, dan nilai $t^2 - n$ dimasukkan ke kolom untuk $t^2 - n$ dan kolom untuk x.

2. Untuk setiap p ganjil dalam *factor base*, dapatkan solusi persamaan $t^2 \equiv n \pmod{p^\alpha}$ dengan $\alpha = 1, 2, \ldots, \beta$ dimana β adalah pangkat terbesar yang memberi solusi dengan $\lfloor\sqrt{n}\rfloor + 1 \leq t \leq \lfloor\sqrt{n}\rfloor + A$. Ini dapat dilakukan, misalnya dengan cara yang dibahas di bagian 11.2. Solusi untuk persamaan $t^2 \equiv n \pmod{p^\beta}$ ada dua, sebut saja t_1 dan t_2, yang keduanya tidak harus berada dalam interval $(\lfloor\sqrt{n}\rfloor + 1, \lfloor\sqrt{n}\rfloor + A)$. Dengan $\alpha = 1$ lantas 2 dan seterusnya hingga β, untuk setiap t yang berbeda kelipatan p^α dari t_1 dan setiap t yang berbeda kelipatan p^α dari t_2 kita bagi x dengan p dan taruh α dalam kolom untuk p.

3. Untuk $p = 2$, jika $n \equiv 1 \pmod 8$ maka prosedurnya sama dengan untuk p ganjil, tetapi untuk $\beta \geq 3$ solusinya ada 4 (t_1, t_2, t_3 dan t_4). Jika $n \equiv 5 \pmod 8$ maka prosedurnya sama dengan untuk p ganjil, tetapi $\beta = 2$ dan solusinya ada 2. Jika $n \not\equiv 1 \pmod 4$ maka untuk setiap t yang ganjil, kita taruh 1 dalam kolom untuk $p = 2$ dan kita bagi x dengan 2.

4. Buang t yang tidak menghasilkan $x = 1$.

Jadi setelah langkah-langkah diatas dilakukan, yang tersisa adalah baris-baris untuk t dimana $t^2 - n$ terbagi habis oleh faktor-faktor dalam *factor base* ($t^2 - n$ *smooth over the factor base*), dengan kata lain $t^2 - n$ merupakan *B-number*. Kita tinggal mencari kombinasi linear beberapa t yang menghasilkan pemangkatan genap elemen-elemen *factor base*. Caranya sama dengan metode Dixon dan metode *continued fraction* yang telah dibahas.

Sedikit penjelasan untuk $p = 2$. Persamaan yang harus dicari solusinya adalah:

$$t^2 \equiv n \pmod{2^\alpha}.$$

Karena n ganjil maka t harus berupa bilangan ganjil. Jadi t dapat ditulis dalam bentuk $t = 2m + 1$ dimana m adalah bilangan bulat. Jadi

$$n \equiv (2m+1)^2 \pmod{2^\alpha}$$

$$\begin{aligned} &\equiv 4m^2 + 4m + 1 \pmod{2^\alpha} \\ &\equiv 1 + 4m(m+1) \pmod{2^\alpha}. \end{aligned}$$

Karena m adalah bilangan bulat, maka $m(m+1)$ merupakan bilangan genap, jadi dapat ditulis dalam bentuk $m(m+1) = 2j$ dimana j adalah bilangan bulat. Jadi

$$\begin{aligned} n &\equiv 1 + 4(2j) \pmod{2^\alpha} \\ &\equiv 1 + 8j \pmod{2^\alpha}. \end{aligned}$$

Untuk $\alpha \geq 3$, ini berarti $n \equiv 1 \pmod 8$. Untuk $\alpha = 2$, kita dapatkan

$$\begin{aligned} n &\equiv 1 + 8j \pmod 4 \\ &\equiv 1 \pmod 4. \end{aligned}$$

Jika $n \equiv 5 \pmod 8$ maka ada dua solusi untuk

$$t^2 \equiv 1 \pmod 4$$

yaitu $t_1 \equiv 1 \pmod 4$ dan $t_2 \equiv 3 \pmod 4$. Jika $n \not\equiv 1 \pmod 4$ maka solusi untuk

$$t^2 \equiv 1 \pmod 2$$

adalah $t \equiv 1 \pmod 2$ yang berarti t ganjil.

Sebagai contoh penggunaan metode *quadratic sieve*, mari kita coba uraikan $n = 1042387$. Kita dapatkan $\lfloor \sqrt{n} \rfloor = 1020$ dan $e^{\sqrt{\log n \log\log n}} \approx 418$. Kita pilih $P = 50$ dan $A = 500$. Kita dapatkan *factor base* $\{2, 3, 11, 17, 19, 23, 43, 47\}$ (n adalah *quadratic residue* untuk setiap bilangan prima dalam *factor base*). Untuk $p = 2$, karena $n \not\equiv 1 \pmod 4$, maka kolom berisi 1 untuk setiap t ganjil dan kosong (atau 0) untuk setiap t genap. Untuk kolom $p = 3$, kita perlu solusi

$$t_1 = t_{1,0} + 3t_{1,1} + 3^2 t_{1,2} + \cdots + 3^{\beta-1} t_{1,\beta-1}$$

untuk persamaan

$$t_1^2 \equiv 1042387 \pmod{3^\beta}$$

dimana setiap $t_{1,i}$ merupakan *ternary digit* ($t_{1,i} \in \{0, 1, 2\}$). Karena $p = 3$ tidak terlalu besar, kita dapat melakukan *trial and error* untuk mendapatkan $t_{1,0} = 1$ dari

$$t_{1,0}^2 \equiv 1042387 \pmod 3.$$

Jika p terlalu besar, kita dapat menggunakan teknik yang telah dibahas di bagian 11.2. Berikutnya, kita cari $t_{1,1}$ dari

$$(1 + 3t_{1,1})^2 \equiv 1042387 \pmod{3^2},$$

mendapatkan

$$1 + 6t_{1,1} \equiv 7 \pmod 9.$$

Jadi $6t_{1,1} \equiv 6 \pmod 9$ atau $2t_{1,1} \equiv 2 \pmod 3$, jadi kita dapatkan $t_{1,1} = 1$. Selanjutnya untuk $t_{1,2}$ kita gunakan

$$(1 + 3 + 9t_{1,2})^2 \equiv 1042387 \pmod{3^3},$$

dan seterusnya hingga kita dapatkan $t_1 \equiv (210211)_3 \pmod{3^7}$ dan $t_2 = 3^7 - (210211)_3 = (2012012)_3$ jadi $t_2 \equiv (2012012)_3 \pmod{3^7}$. Namun tidak ada t diantara 1021 dan 1520 inklusif yang mematuhi persamaan $t \equiv t_1 \pmod{3^7}$ atau $t \equiv t_2 \pmod{3^7}$. Jadi $\beta = 6$ dan kita ambil

$$\begin{aligned} t_1 &\equiv (210211)_3 \pmod{3^6} \\ &\equiv 1318 \pmod{3^6} \end{aligned}$$

dan $t_2 = 3^6 - 1318 = 140$, dan karena tidak ada $t \equiv t_2 \pmod{3^6}$ dengan $1021 \leq t \leq 1050$ maka

$$\begin{aligned} t_2 &\equiv 140 \pmod{3^5} \\ &\equiv 1112 \pmod{3^5}. \end{aligned}$$

Proses penyaringan dimulai dengan melakukan langkah-langkah berikut untuk setiap t dengan $1021 \leq t \leq 1050$ dan $t - t_1 \equiv 0 \pmod 3$:

1. taruh 1 di kolom $p = 3$,
2. bagi x dengan 3,

lalu untuk setiap t dengan $1021 \leq t \leq 1050$ dan $t - t_1 \equiv 0 \pmod{3^2}$ kita lakukan langkah-langkah berikut:

1. ganti 1 menjadi 2 di kolom $p = 3$,
2. bagi x dengan 3,

dan seterusnya hingga untuk setiap t dengan $1021 \leq t \leq 1050$ dan $t - t_1 \equiv 0 \pmod{3^6}$ kita lakukan langkah-langkah berikut:

1. ganti 5 menjadi 6 di kolom $p = 3$,
2. bagi x dengan 3.

Proses dilanjutkan dengan melakukan langkah-langkah berikut untuk setiap t dengan $1021 \leq t \leq 1050$ dan $t - t_2 \equiv 0 \pmod 3$:

1. taruh 1 di kolom $p = 3$,

2. bagi x dengan 3,

dan seterusnya hingga untuk setiap t dengan $1021 \leq t \leq 1050$ dan $t - t_2 \equiv 0 \pmod{3^5}$ kita lakukan langkah-langkah berikut:

1. ganti 4 menjadi 5 di kolom $p = 3$,
2. bagi x dengan 3.

Proses penyaringan dilakukan dengan setiap p ganjil dalam *factor base*. Setelah kita buang setiap t dengan $x \neq 1$ kita dapatkan tabel 14.1. Dari baris ke 5 dan

t	$t^2 - n$	x	2	3	11	17	19	23	43	47
1021	54	1	1	3	–	–	–	–	–	–
1027	12342	1	1	1	2	1	–	–	–	–
1030	18513	1	–	2	2	1	–	–	–	–
1061	83334	1	1	1	–	1	1	–	1	–
1112	194157	1	–	5	–	1	–	–	–	1
1129	232254	1	1	3	1	1	–	1	–	–
1148	275517	1	–	2	3	–	–	1	–	–
1175	338238	1	1	2	–	–	1	1	1	–
1217	438702	1	1	1	1	2	–	1	–	–
1390	889713	1	–	2	2	–	1	–	1	–
1520	1268013	1	–	1	–	1	–	2	–	1

Tabel 14.1: Tabel Quadratic Sieve untuk $n = 1042387$

baris terahir kita dapatkan

$$(1112 \cdot 1520)^2 \equiv (3^{(5+1)/2} \cdot 17^{(1+1)/2} \cdot 23^{2/2} \cdot 47^{(1+1)/2})^2 \pmod{1042387}$$

jadi

$$647853^2 \equiv 496179^2 \pmod{1042387}.$$

Kita dapatkan faktor $\gcd(647853 - 496179, 1042387) = 1487$.

Kompleksitas metode *quadratic sieve*, seperti halnya dengan metode *continued fraction* dan metode Dixon, diestimasikan adalah

$$O(e^{C\sqrt{\log n \log\log n}})$$

tetapi dengan konstan C yang lebih kecil lagi. Untuk analisa yang cukup mendalam mengenai kompleksitas metode ini, silahkan membaca [pom82].

Dalam prakteknya, metode *quadratic sieve* merupakan cara tercepat untuk menguraikan bilangan besar sampai dengan 100 digit. Untuk menguraikan bilangan lebih dari 100 digit, metode *number field sieve* yang akan dibahas di bagian berikut, lebih cepat.

14.6 Metode Number Field Sieve

Yang dimaksud dengan metode *number field sieve* disini adalah metode *general number field sieve*. Ini berbeda dengan metode *special number field sieve* yang digunakan untuk mencari faktor dari bilangan-bilangan tertentu seperti *Fermat numbers*. Secara garis besar, metode *number field sieve* untuk menguraikan bilangan n adalah sebagai berikut. Pilih *degree* d untuk *polynomial* $f(x)$:

$$d \approx \left(\frac{3 \log n}{2 \log \log n} \right)^{\frac{1}{3}}.$$

Untuk n 100 hingga 200 digit, biasanya dipilih $d = 5$. Berikutnya, dengan m sebagai bagian bulat akar pangkat d dari n:

$$m = \lfloor \sqrt[d]{n} \rfloor,$$

tuliskan n sebagai bilangan dengan basis m:

$$n = m^d + c_{d-1}m^{d-1} + \ldots + c_1 m + c_0,$$

dimana $0 \leq c_i < m$. Maka *polynomial* $f(x)$ didefinisikan sebagai berikut:

$$f(x) = x^d + c_{d-1}x^{d-1} + \ldots + c_1 x + c_0,$$

jadi $n = f(m)$. Jika $f(x)$ *reducible*, maka $f(x)$ dapat diuraikan menjadi $f(x) = g(x)h(x)$, dimana $g(x)$ dan $h(x)$ merupakan *polynomial* non-konstan. Jadi

$$n = f(m) = g(m) \cdot h(m)$$

yang akan menghasilkan penguraian n dan kita selesai. Jadi untuk selanjutnya kita umpamakan $f(x)$ *irreducible*. Algoritma untuk penguraian *polynomial*, contohnya yang dijelaskan di [len82], dapat digunakan untuk melakukan test apakah $f(x)$ *irreducible* (dan menguraikannya jika $f(x)$ *reducible*). Algoritma tersebut menguraikan *polynomial* dalam $\mathbf{Q}$, tetapi, seperti akan dijelaskan, *polynomial* yang *irreducible* dalam $\mathbf{N}$ berarti juga *irreducible* dalam $\mathbf{Q}$. Kompleksitas dari algoritma tersebut adalah *polynomial*.

Jika metode *quadratic sieve* menyaring kandidat t dimana $t^2 - n$ *smooth over factor base* (dapat diuraikan menjadi produk elemen-elemen *factor base*), maka metode *number field sieve* menyaring kandidat pasangan (a, b) dimana $a + mb$ dan $(-b)^d f(-\frac{a}{b})$ keduanya dapat diuraikan menjadi produk elemen-elemen *factor base*. Penyaringan dilakukan untuk memperkecil ruang pencarian subset kandidat yang menghasilkan kuadrat. Akan tetapi, untuk metode *number field sieve*, perlu ada penyaringan tambahan dengan berbagai alasan yang akan dibahas dalam penjelasan berikut.

Motivasi untuk metode *number field sieve* adalah pengamatan bahwa dalam metode *quadratic sieve*, fungsi

$$f(t) = t^2 - n,$$

menghasilkan *ring homomorphism* dari $\mathbf{Z}$ ke $\mathbf{Z}/n\mathbf{Z}$, yang memetakan dua kuadrat yang berbeda dalam $\mathbf{Z}$ (yaitu $\prod_{t_i \in U} f(t_i)$ dan $\prod_{t_i \in U} t_i^2$) ke kuadrat yang sama dalam $\mathbf{Z}/n\mathbf{Z}$. Yang menjadi pertanyaan adalah apakah f harus berupa *polynomial* dengan *degree* 2 dan apakah *ring homomorphism* harus dari $\mathbf{Z}$ ke $\mathbf{Z}/n\mathbf{Z}$? Jawabannya ternyata tidak.

Mari kita mulai dengan $f(x)$ berupa *monic polynomial* yang *irreducible* sebagai *polynomial* bilangan bulat. Kita gunakan *Gauss's Lemma 2* untuk menunjukkan bahwa *polynomial* tersebut juga *irreducible* sebagai *polynomial* bilangan rasional.

Teorema 107 (Gauss's Lemma 2) *1. Produk dari dua polynomial primitif adalah polynomial primitif.*

2. Jika suatu polynomial irreducible sebagai polynomial bilangan bulat, maka polynomial tersebut juga irreducible sebagai polynomial bilangan rasional.

Suatu *polynomial* disebut primitif jika setiap koefisien berupa bilangan bulat dan gcd (pembagi persekutuan terbesar) dari semua koefisien adalah 1. Pembuktian bagian pertama kita lakukan dengan dua cara: cara pertama adalah pembuktian kongkrit, sedangkan cara kedua lebih abstrak. Jika $f(x)$ dan $g(x)$ keduanya merupakan *polynomial* primitif, maka jelas bahwa setiap koefisien produk $f(x) \cdot g(x)$ merupakan bilangan bulat. Jika produk bukan merupakan *polynomial* primitif maka terdapat bilangan prima p yang membagi setiap koefisien dari produk. Karena $f(x)$ dan $g(x)$ keduanya primitif, maka terdapat suku-suku dalam $f(x)$ dan $g(x)$ yang koefisiennya tidak dapat dibagi oleh p. Jika $a_r x^r$ merupakan suku pertama dalam $f(x)$ yang tidak dapat dibagi oleh p dan $b_s x^s$ merupakan suku pertama dalam $g(x)$ yang tidak dapat dibagi oleh p, maka suku x^{r+s} dalam produk mempunyai koefisien dengan rumus:

$$a_r b_s + a_{r+1} b_{s-1} + a_{r+2} b_{s-2} + \ldots + a_{r-1} b_{s+1} + a_{r-2} b_{s+2} + \ldots$$

Suku pertama dalam koefisien tidak dapat dibagi oleh p (karena a_r dan b_s keduanya tidak dapat dibagi p), sedangkan suku-suku lainnya dapat dibagi p (karena $b_{s-1}, b_{s-1}, \ldots$ dan $a_{r-1}, a_{r-2}, \ldots$ semua dapat dibagi p), jadi koefisien tidak dapat dibagi p. Jadi produk harus berupa *polynomial* primitif.

Cara pembuktian kedua lebih abstrak. Jika $f(x) \cdot g(x)$ tidak primitif, maka semua koefisien produk dapat dibagi oleh satu bilangan prima, sebut saja p. Berarti $f(x) \cdot g(x) = 0$ dalam *ring* $(\mathbf{Z}/p\mathbf{Z})[X]$. Karena $\mathbf{Z}/p\mathbf{Z}$ merupakan *integral domain*, maka $f(x)$ atau $g(x)$ harus 0 dalam $(\mathbf{Z}/p\mathbf{Z})[X]$, jadi p harus membagi

setiap koefisien $f(x)$ atau $g(x)$. Tetapi ini tidak mungkin karena $f(x)$ dan $g(x)$ keduanya primitif. Jadi produk juga harus primitif.

Mari kita buktikan bagian kedua dari teorema 107 secara kontra-positif menggunakan bagian pertama. Jika *polynomial* $f(x)$ tidak primitif, maka $f(x)$ dapat dibagi oleh gcd semua koefisien $f(x)$ menghasilkan $f'(x)$ yang primitif; dan, jika $f(x)$ dapat diuraikan menjadi produk dua *polynomial* bilangan bulat non-konstan, maka $f'(x)$ juga dapat diuraikan menjadi produk dua *polynomial* bilangan bulat non-konstan. Jadi untuk pembuktian, kita asumsikan bahwa $f(x)$ primitif. Jika $f(x)$ *reducible* sebagai *polynomial* bilangan rasional maka $f(x)$ dapat diuraikan menjadi produk dua *polynomial* bilangan rasional non-konstan $g(x)$ dan $h(x)$ ($f(x) = g(x) \cdot h(x)$). Terdapat $a, b \in \mathbf{Z}$ dimana $a \cdot g(x)$ dan $b \cdot h(x)$ keduanya primitif (dalam $\mathbf{Z}[X]$). Menggunakan bagian pertama, berarti

$$(a \cdot g(x)) \cdot (b \cdot h(x)) = (ab)f(x)$$

juga primitif. Berarti ab merupakan unit dalam $\mathbf{Z}$, jadi a dan b masing-masing merupakan unit dalam $\mathbf{Z}$. Jadi $f(x)$ dapat diuraikan menjadi

$$f(x) = (b^{-1} \cdot g(x)) \cdot (a^{-1} \cdot h(x))$$

dimana koefisien-koefisien $(b^{-1} \cdot g(x))$ dan $(a^{-1} \cdot h(x))$ semua bilangan bulat, yang berarti $f(x)$ *reducible* dalam $\mathbf{Z}[X]$. Jadi jika $f(x)$ *reducible* dalam $\mathbf{Q}[X]$ (sebagai *polynomial* bilangan rasional) maka $f(x)$ juga *reducible* dalam $\mathbf{Z}[X]$ (sebagai *polynomial* bilangan bulat). Selesailah pembuktian kita.

Kembali ke $f(x)$ berupa *monic polynomial* dengan *degree* d yang *irreducible* sebagai *polynomial* bilangan bulat (dan didefinisikan berdasarkan m dan d). Menurut *Gauss's Lemma 2* maka $f(x)$ juga *irreducible* sebagai *polynomial* bilangan rasional. Berdasarkan *Fundamental Theorem of Algebra* (lihat [fin97] atau cari di internet), $f(x)$ dapat diuraikan sebagai berikut:

$$f(x) = (x - \alpha_1)(x - \alpha_2) \cdots (x - \alpha_d)$$

dimana $\alpha_i \in \mathbf{C}$ (bilangan kompleks). Kita dapat memilih $\alpha = \alpha_i$ akar mana saja dan membuat *field* $\mathbf{Q}(\alpha)$ (lihat bagian 10.6). Elemen-elemen dari $\mathbf{Q}(\alpha)$ dapat direpresentasikan oleh kombinasi linear elemen-elemen

$$S = \{1, \alpha, \alpha^2, \ldots, \alpha^{d-1}\}.$$

Kita akan fokus pada subset dari $\mathbf{Q}(\alpha)$ yang dapat direpresentasikan oleh kombinasi $\mathbf{Z}$-linear elemen-elemen S (jadi semua koefisien bilangan bulat). Subset ini membentuk suatu *ring* $\mathbf{Z}[\alpha]$ dan merupakan *subring* dari $\mathfrak{D}$, algebraic integers dalam $\mathbf{Q}(\alpha)$ (lihat bagian 12.4). Lalu bagaimana kita mendapatkan perbedaan kuadrat dari $\mathbf{Z}[\alpha]$? Karena $f(\alpha) = 0$ dan $f(m) \equiv 0 \pmod{n}$ maka terdapat *ring homomorphism* φ dari $\mathbf{Z}[\alpha]$ ke $\mathbf{Z}/n\mathbf{Z}$ dimana:

- $i \mapsto [i]$ untuk $i \in \mathbf{Z}$ dan
- $\alpha \mapsto m \pmod{n}$.

Kita buat U berupa himpunan *finite* dari pasangan bilangan bulat (a, b) dimana:

1. Produk dari $a + \alpha b$ untuk setiap pasangan (a, b) dalam U merupakan kuadrat dalam $\mathbf{Z}[\alpha]$, sebut saja γ^2.
2. Produk dari $a + mb$ untuk setiap pasangan (a, b) dalam U merupakan kuadrat dalam $\mathbf{Z}$, sebut saja v^2.

Karena γ dapat direpresentasikan sebagai *polynomial* dalam α, dan α dipetakan ke m oleh φ, kita bisa dapatkan bilangan bulat u dimana $\varphi(\gamma) \equiv u \pmod{n}$. Jadi

$$\begin{aligned} u^2 &\equiv \varphi(\gamma)^2 \pmod{n} \\ &\equiv \varphi(\gamma^2) \pmod{n} \\ &\equiv \varphi\left(\prod_{(a,b)\in U} (a + \alpha b)\right) \pmod{n} \\ &\equiv \prod_{(a,b)\in U} \varphi(a + \alpha b) \pmod{n} \\ &\equiv \prod_{(a,b)\in U} (a + mb) \pmod{n} \\ &\equiv v^2 \pmod{n}. \end{aligned}$$

Setelah kita dapatkan $u^2 \equiv v^2 \pmod{n}$, jika $u \not\equiv \pm v \pmod{n}$ maka kita tahu apa yang harus dilakukan untuk mendapatkan penguraian n.

Serupa dengan metode *quadratic sieve*, penyaringan harus dilakukan terhadap pasangan (a, b). Akan tetapi penyaringan harus dilakukan dari dua sisi:

- Berdasarkan nilai $a + mb$ dalam $\mathbf{Z}$.
- Berdasarkan nilai $a + \alpha b$ dalam $\mathbf{Z}[\alpha]$.

Dari sisi nilai $a+mb$ dalam $\mathbf{Z}$, penyaringan untuk mendapatkan produk berupa kuadrat dapat dilakukan serupa dengan metode *quadratic sieve*, tetapi menggunakan dua variabel: a dan b. Dari sisi nilai $a + \alpha b$ dalam $\mathbf{Z}[\alpha]$, penyaringan lebih rumit. Pada prinsipnya kita dapat saja melakukan komputasi menggunakan *ring* $\mathbf{Z}[\alpha]$. Akan tetapi ini akan sangat tidak efisien dan kita akan menghadapi banyak kesulitan lainnya. Kita akan jelaskan cara penyaringan

yang lebih efisien, akan tetapi, sebelum melanjutkan, kita cari rumus *norm* untuk $a + \alpha b$:

$$\begin{aligned} N(a+\alpha b) &= \sigma_1(a+\alpha b)\sigma_2(a+\alpha b)\cdots\sigma_d(a+\alpha b) \\ &= (a+\alpha_1 b)(a+\alpha_2 b)\cdots(a+\alpha_d b) \\ &= b^d\left[(\frac{a}{b}+\alpha_1)(\frac{a}{b}+\alpha_2)\cdots(\frac{a}{b}+\alpha_d)\right] \\ &= (-b)^d\left[(-\frac{a}{b}-\alpha_1)(-\frac{a}{b}-\alpha_2)\cdots(-\frac{a}{b}-\alpha_d)\right] \\ &= (-b)^d f(-\frac{a}{b}). \end{aligned}$$

Sebetulnya *norm* berlaku pada *field extension*, jadi *norm* yang kita maksud adalah *norm* pada $\mathbf{Q}(\alpha)/\mathbf{Q}$ dengan notasi $N_{\mathbf{Q}}^{\mathbf{Q}(\alpha)}$. Kita gunakan notasi N karena lebih ringkas dan cukup jelas apa yang dimaksud.

Sekarang kita jelaskan cara penyaringan dari sisi nilai $a + \alpha b$ dalam $\mathbf{Z}[\alpha]$. Karena $\mathbf{Z}[\alpha]$ bukan seluruh *algebraic integers* dalam $\mathbf{Q}(\alpha)/\mathbf{Q}$, maka kita tidak akan gunakan elemen-elemen prima dalam $\mathbf{Z}[\alpha]$ sebagai *factor base*. Untuk *factor base* kita akan gunakan *prime ideals* tertentu dalam $\mathbf{Z}[\alpha]$ yang dinamakan *first degree prime ideals*. Suatu *ideal* dalam $\mathbf{Z}[\alpha]$ adalah suatu *first degree prime ideal* jika *norm* dari *ideal* tersebut adalah bilangan prima.

Teorema 108 *Terdapat suatu bijective mapping antara first degree prime ideals dari* $\mathbf{Z}[\alpha]$ *dengan pasangan-pasangan* (r, p) *dimana* p *adalah bilangan prima,* $r \in \mathbf{Z}/p\mathbf{Z}$*, dan* $f(r) = 0 \pmod p$.

Jika $\mathfrak{p}$ merupakan *first degree prime ideal* dalam $\mathbf{Z}[\alpha]$ maka $|\mathbf{Z}[\alpha]/\mathfrak{p}| = p$ untuk suatu bilangan prima p, jadi

$$\mathbf{Z}[\alpha]/\mathfrak{p} \simeq \mathbf{Z}/p\mathbf{Z}.$$

Terdapat *canonical homomorphism* $\varphi : \mathbf{Z}[\alpha] \longrightarrow \mathbf{Z}[\alpha]/\mathfrak{p}$ yang *surjective* (*homomorphism* yang *surjective* dinamakan *epimorphism*) dan $\ker(\varphi) = \mathfrak{p}$. Karena $\mathbf{Z}[\alpha]/\mathfrak{p} \simeq \mathbf{Z}/p\mathbf{Z}$ maka φ dapat dipandang sebagai *epimorphism*:

$$\varphi : \mathbf{Z}[\alpha] \longrightarrow \mathbf{Z}/p\mathbf{Z}$$

dengan $\ker(\varphi) = \mathfrak{p}$, jadi $\varphi(a) = a \pmod p$ untuk setiap bilangan bulat a. Jika $r = \varphi(\alpha) \in \mathbf{Z}/p\mathbf{Z}$ maka

$$\begin{aligned} 0 &\equiv \varphi(f(\alpha)) \\ &\equiv \varphi(\alpha^d + c_{d-1}\alpha^{d-1} + \ldots + c_1\alpha + c_0) \\ &\equiv \varphi(\alpha)^d + c_{d-1}\varphi(\alpha)^{d-1} + \ldots + c_1\varphi(\alpha) + c_0 \\ &\equiv r^d + c_{d-1}r^{d-1} + \ldots c_1 r + c_0 \\ &\equiv f(r) \pmod p \end{aligned}$$

jadi r merupakan akar dari $f(x) \pmod p$ dan *ideal* $\mathfrak{p}$ menentukan pasangan unik (r, p). Sebaliknya, jika p adalah bilangan prima dan $r \in \mathbf{Z}/p\mathbf{Z}$ dengan $f(r) \equiv 0 \pmod p$ maka terdapat *ring epimorphism*

$$\varphi : \mathbf{Z}[\alpha] \longrightarrow \mathbf{Z}/p\mathbf{Z}$$

dimana

$$\varphi(a) \equiv a \pmod p$$

untuk setiap $a \in \mathbf{Z}$, dan

$$\varphi(\alpha) \equiv r \pmod p.$$

Jika kita buat $\mathfrak{p} = \ker(\varphi)$ maka $\mathfrak{p}$ merupakan *ideal* dalam $\mathbf{Z}[\alpha]$. Karena φ *surjective* dan $\ker(\varphi) = \mathfrak{p}$ maka

$$\mathbf{Z}[\alpha]/\mathfrak{p} \simeq \mathbf{Z}/p\mathbf{Z},$$

yang berarti $|\mathbf{Z}[\alpha]/\mathfrak{p}| = p$, jadi $\mathfrak{p}$ merupakan *first degree prime ideal* dalam $\mathbf{Z}[\alpha]$. Jadi pasangan (r, p) menentukan *first degree prime ideal* yang unik dan selesailah pembuktian teorema 108.

Untuk mencari kuadrat berdasarkan *factor base* dalam $\mathbf{Z}[\alpha]$ kita dapat menggunakan vektor pemangkatan untuk $N(\langle a + \alpha b\rangle)$. Akan tetapi meskipun ini menghasilkan produk kuadrat dalam $\mathbf{Z}$, produk dalam $\mathbf{Z}[\alpha]$ belum tentu kuadrat, jadi diperlukan mekanisme tambahan. Kita jelaskan terlebih dahulu teori dibalik penggunaan vektor pemangkatan untuk $N(\langle a+\alpha b\rangle)$. Jika $a, b \in \mathbf{Z}$ dan a koprima dengan b ($\gcd(a, b) = 1$) dan pasangan (r, p) merepresentasikan *first degree prime ideal*, maka kita definisikan $e_{r,p}(a + \alpha b)$ sebagai berikut:

$$e_{r,p}(a + \alpha b) = \begin{cases} \operatorname{ord}_p(N(\langle a + \alpha b\rangle)) & \text{jika } a + rb \equiv 0 \pmod p \\ 0 & \text{jika } a + rb \not\equiv 0 \pmod p \end{cases}$$

dimana $\operatorname{ord}_p(k)$ adalah banyaknya p sebagai faktor dalam k. Tentunya

$$N(\langle a + \alpha b\rangle) = \prod_{r,p} p^{e_{r,p}(a+\alpha b)}$$

dimana produk meliputi semua pasangan (r, p). Penggunaan $e_{r,p}(a + \alpha b)$ didasarkan pada pengamatan bahwa jika U merupakan himpunan *finite* berbagai pasangan bilangan bulat (a, b) dimana a koprima dengan b, dan

$$\prod_{(a,b)\in U} (a + \alpha b)$$

merupakan kuadrat dalam $\mathbf{Q}(\alpha)$, maka untuk setiap pasangan (r, p) kita dapatkan

$$\sum_{(a,b)\in U} e_{r,p}(a + \alpha b) \equiv 0 \pmod 2. \tag{14.2}$$

Kita akan jelaskan persamaan 14.2. Jika $\mathbf{Z}[\alpha]$ sama dengan $\mathfrak{D}$ maka jelas apa yang dimaksud dengan $e_{r,p}(a+\alpha b)$ karena setiap $\beta \in \mathfrak{D}$ dapat diuraikan sepenuhnya menjadi produk prima dalam $\mathfrak{D}$. Akan tetapi biasanya $\mathbf{Z}[\alpha]$ tidak sama dengan $\mathfrak{D}$ jadi kita perlukan teorema berikut.

Teorema 109 *Untuk setiap ideal prima P dalam $\mathbf{Z}[\alpha]$ terdapat suatu group homomorphism $l_P : \mathbf{Q}(\alpha)^* \longrightarrow \mathbf{Z}$ dimana*

1. *$l_P(x) \geq 0$ untuk setiap $x \in \mathbf{Z}[\alpha]$, $x \neq 0$.*
2. *Jika $x \in \mathbf{Z}[\alpha]$, $x \neq 0$, maka $l_P(x) > 0$ jika dan hanya jika $x \in P$.*
3. *Untuk setiap $x \in \mathbf{Q}(\alpha)^*$, $l_P(x) = 0$ kecuali untuk beberapa P yang banyaknya finite, dan kita dapatkan*

$$\prod_P (N(P))^{l_P(x)} = N(\langle x \rangle)$$

dimana P meliputi semua ideal prima dalam $\mathbf{Z}[\alpha]$.

Untuk membuktikan teorema 109, kita definisikan terlebih dahulu fungsi l_p. Jika P merupakan *ideal* prima dalam $\mathbf{Z}[\alpha]$, $x \in \mathbf{Z}[\alpha]$ dan $x \neq 0$, maka karena $x\mathbf{Z}[\alpha]$ mempunyai *finite index* dalam $\mathbf{Z}[\alpha]$, terdapat *finite chain*

$$\mathbf{Z}[\alpha] = I_0 \supset I_1 \supset I_2 \cdots I_{t-1} \supset I_t = x\mathbf{Z}[\alpha]$$

terdiri dari *ideals* yang berbeda, dan untuk $1 \leq i \leq t$, tidak terdapat *ideal* J dimana $I_{i-1} \supset J \supset I_i$. Kita definisikan $l_p(x)$ sebagai banyaknya $i \in \{1, 2, \ldots, t\}$ dimana

$$I_{i-1}/I_i \simeq \mathbf{Z}[\alpha]/P$$

sebagai $\mathbf{Z}[\alpha]$-*module*. Berdasarkan teorema Jordan-Hölder (lihat [wae66] bagian 51, atau cari di internet), $l_p(x)$ *well-defined* karena tidak tergantung pada bagaimana *finite chain* dipilih. Jika $0 \neq y \in \mathbf{Z}[\alpha]$ maka *finite chain* untuk x dapat disambung dengan *finite chain* untuk y:

$$\mathbf{Z}[\alpha] = J_0 \supset J_1 \supset J_2 \cdots J_{u-1} \supset J_u = y\mathbf{Z}[\alpha]$$

dan kita dapatkan *finite chain* untuk xy:

$$\mathbf{Z}[\alpha] = I_0 \supset I_1 \cdots I_t = xJ_0 \supset xJ_1 \cdots xJ_u = xy\mathbf{Z}[\alpha].$$

Jadi $l_p(xy) = l_p(x) + l_p(y)$. Dengan mendefinisikan $l_p(x/z) = l_p(x) - l_p(z)$ untuk $x, z \in \mathbf{Z}[\alpha]$ dimana x dan z tidak sama dengan 0, kita dapat memperluas *domain* l_p menjadi $\mathbf{Q}(\alpha)^*$. Sangat jelas bahwa bagian 1 dari teorema 109 berlaku. Untuk membuktikan $l_p(x) > 0 \Longleftarrow x \in P$ di bagian 2, kita buat $I_1 =$

P. Untuk membuktikan $l_p(x) > 0 \Longrightarrow x \in P$, kita lihat apa konsekuensinya jika $x \notin P$. Karena P maksimal, maka *ideal* $x\mathbf{Z}[\alpha] + P = \mathbf{Z}[\alpha]$. Jadi

$$\exists y \in \mathbf{Z}[\alpha], z \in P : xy + z = 1.$$

Efeknya mengalikan dengan z adalah *identity map* $\mathbf{Z}[\alpha]/x\mathbf{Z}[\alpha] \longrightarrow \mathbf{Z}[\alpha]/x\mathbf{Z}[\alpha]$. Akibatnya,

$$z \cdot (I_{i-1}/I_i) = (I_{i-1}/I_i),$$

dan karena $z \in P$ maka I_{i-1}/I_i tidak bisa *isomorphic* dengan $\mathbf{Z}[\alpha]/x\mathbf{Z}[\alpha]$, jadi $l_p(x) = 0$. Untuk membuktikan bagian 3, karena $l_p(x) = 0$ jika $x \notin \mathbf{Z}[\alpha]$, maka kita tinggal menunjukkan persamaan untuk $0 \neq x \in \mathbf{Z}[\alpha]$. Karena

$$|N(x)| = |\mathbf{Z}[\alpha]/x\mathbf{Z}[\alpha]| = \prod_{i=1}^{t} |I_{i-1}/I_i|$$

maka kita tinggal tunjukkan bahwa untuk setiap i terdapat *ideal* prima P yang unik dimana $I_{i-1}/I_i \simeq \mathbf{Z}[\alpha]/P$. Kita pilih $y \in I_{i-1}$ dimana $y \notin I_i$. Karena tidak terdapat *ideal* J dimana

$$I_{i-1} \supset J \supset I_i$$

maka $y\mathbf{Z}[\alpha]+I_i = I_{i-1}$, jadi efek dari perkalian dengan y adalah suatu *surjective map* $\mathbf{Z}[\alpha] \longrightarrow I_{i-1}/I_i$. Jadi terdapat suatu *ideal* P dimana

$$\mathbf{Z}[\alpha]/P \simeq I_{i-1}/I_i.$$

Karena $\mathbf{Z}[\alpha]/P$ tidak memiliki *non-trivial submodule* maka P maksimal, yang berarti P prima. Juga, karena P merupakan *annihilator* untuk I_{i-1}/I_i sebagai $\mathbf{Z}[\alpha]$-*module*:

$$P = \{r \in \mathbf{Z}[\alpha] \mid \forall m \in I_{i-1}/I_i : rm = 0\},$$

maka P unik. Selesailah pembuktian teorema 109.

Sebagai konsekuensi dari teorema 109 kita dapatkan teorema berikut.

Teorema 110 *Untuk a dan b dua bilangan bulat yang koprima dan P suatu ideal prima dalam $\mathbf{Z}(\alpha)$:*

- *Jika P bukan first degree prime ideal maka $l_P(a + \alpha b) = 0$.*
- *Jika P adalah first degree prime ideal yang direpresentasikan dengan pasangan (r, p) maka $l_P(a + \alpha b) = e_{r,p}(a + \alpha b)$.*

Mari kita buktikan teorema 110. Jika P merupakan *ideal* prima dalam $\mathbf{Z}[\alpha]$ dengan $l_P(a + \alpha b) > 0$, maka berdasarkan teorema 109, $a + \alpha b$ dipetakan ke 0 oleh *canonical homomorphism* $\mathbf{Z}[\alpha] \longrightarrow \mathbf{Z}[\alpha]/P$. Terdapat bilangan prima p

dalam P. Jika p membagi b, maka αb juga dipetakan ke 0, dengan demikian a juga dipetakan ke 0, jadi p membagi a, suatu kontradiksi dengan $\gcd(a, b) = 1$. Jadi b dipetakan ke elemen $b' \neq 0$ dalam $\mathbf{Z}[\alpha]/P$. Bukan itu saja, $b' \equiv b \pmod{p}$ berada didalam *prime field* F_p dan mempunyai *inverse* b'^{-1}. Demikian juga a dipetakan ke elemen $a' \in F_p$. Karena $a + \alpha b$ dipetakan ke 0, maka α dipetakan ke $a'b'^{-1}$ yang merupakan elemen dari F_p. Jadi seluruh $\mathbf{Z}[\alpha]$ dipetakan ke F_p yang berarti P merupakan *first degree prime ideal*. Selesailah pembuktian bagian pertama. Untuk bagian kedua, kita gunakan

$$\prod_p (N(P))^{l_P(\beta)} = N(\langle\beta\rangle)$$

dari teorema 109 dan periksa pangkat dari p di kedua sisi persamaan. Selesailah pembuktian teorema 110.

Sekarang kita bisa dapatkan persamaan 14.2. Kita buat

$$\prod_{(a,b)\in U} a + \alpha b = \gamma^2$$

dan P merupakan *first degree prime ideal* dengan representasi (r, p). Jadi

$$\begin{aligned}
\sum_{(a,b)\in U} e_{r,p}(a + \alpha b) &= \sum_{(a,b)\in U} l_P(a + \alpha b) \\
&= l_P\left(\prod_{(a,b)\in U} (a + \alpha b)\right) \\
&= l_P(\gamma^2) \\
&= 2l_P(\gamma) \\
&\equiv 0 \pmod{2}.
\end{aligned}$$

Teknik diatas membuat perumpamaan bahwa $\mathbf{Z}[\alpha]$ adalah suatu *Dedekind domain*, contohnya jika $\mathbf{Z}[\alpha] = \mathfrak{D}$. Biasanya $\mathbf{Z}[\alpha] \subset \mathfrak{D}$ dan $\mathbf{Z}[\alpha]$ bukan merupakan *Dedekind domain*. Jadi tidak ada jaminan bahwa produk merupakan kuadrat dalam $\mathbf{Z}[\alpha]$. Untuk lebih memberi jaminan, meskipun tidak 100 persen, digunakan *quadratic characters*. Sebagai motivasi untuk konsep *quadratic characters*, kita gunakan contoh yang sederhana yaitu kuadrat dalam $\mathbf{Z}$. Jika x adalah kuadrat dalam $\mathbf{Z}$ ($x = y^2$, $x, y \in \mathbf{Z}$), maka x juga merupakan *perfect square* modulo setiap bilangan prima. Jadi menggunakan simbol Legendre,

$$\left(\frac{x}{p}\right) = 1$$

untuk setiap bilangan prima p. Fakta ini dapat digunakan sebagai test untuk mengetahui apakah suatu bilangan bulat z merupakan kuadrat dalam $\mathbf{Z}$. Jika

kita test $\left(\frac{z}{p}\right)$ dengan beberapa bilangan prima dan hasilnya semua 1 maka besar kemungkinan bahwa z merupakan kuadrat. Semakin banyak bilangan prima yang digunakan, semakin besar jaminan bahwa z merupakan kuadrat. Konsep ini dapat digeneralisasi untuk test kuadrat dalam $\mathbf{Q}(\alpha)$.

Teorema 111 *Jika U merupakan himpunan pasangan (a, b) dimana*

$$\prod_{(a,b)\in U} (a + \alpha b) = \beta^2$$

untuk suatu $\beta \in \mathbf{Q}(\alpha)$, dan $\mathfrak{p}$ adalah suatu first degree prime ideal dengan representasi (r, p) dimana

$$\begin{array}{c} a + rb \not\equiv 0 \pmod{p} \text{ untuk setiap } (a, b) \in U, \\ f'(r) \not\equiv 0 \pmod{p} \end{array}$$

maka

$$\prod_{(a,b)\in U} \left(\frac{a + rb}{p}\right) = 1.$$

Pembuktian teorema 111 menggunakan fakta bahwa jika

$$\prod_{(a,b)\in U} (a + \alpha b) = \beta^2$$

dimana $\beta \in \mathbf{Q}(\alpha)$, maka $\beta \in \mathfrak{D}$ dan $\beta f'(\alpha) \in \mathbf{Z}[\alpha]$. Kita tidak akan buktikan fakta ini disini, pembaca yang berminat dipersilahkan membaca [wei63] (Proposition 3-7-14). Pertama, kita buat *canonical ring epimorphism* φ

$$\varphi : \mathbf{Z}[\alpha] \longrightarrow \mathbf{Z}/p\mathbf{Z}$$

dengan $\varphi(\alpha) \equiv r \pmod{p}$. Jadi $\mathfrak{p} = \ker(\varphi)$ adalah suatu *first degree prime ideal*. Karena φ memetakan elemen-elemen diluar $\mathfrak{p}$ ke elemen-elemen yang bukan 0, kita dapat membuat pemetaan

$$\begin{array}{rcl} \chi_{\mathfrak{p}} : \mathbf{Z}[\alpha] - \mathfrak{p} & \longrightarrow & \{-1, 1\} \\ \delta & \mapsto & \left(\frac{\varphi(\delta)}{p}\right). \end{array}$$

Menggunakan fakta diatas, terdapat suatu $\gamma = \beta f'(\alpha) \in \mathbf{Z}[\alpha]$ dimana

$$f'(\alpha)^2 \prod_{(a,b)\in U} (a + \alpha b) = \gamma^2.$$

Karena $\langle a + \alpha b\rangle$ tidak dapat dibagi oleh $\mathfrak{p}$ berarti $a + \alpha b \notin \mathfrak{p}$. Demikian juga, berdasarkan hipotesis dalam teorema, $f'(r)$ tidak dapat dibagi oleh p, jadi

$f'(\alpha)^2 \notin \mathfrak{p}$. Akibatnya $\langle \gamma^2 \rangle$ tidak dapat dibagi dengan $\mathfrak{p}$, demikian juga $\langle \gamma \rangle$, jadi $\chi_{\mathfrak{p}}$ berlaku untuk γ^2 dan γ. Menggunakan simbol Legendre, kita dapatkan

$$\chi_{\mathfrak{p}}(\gamma^2) = \left(\frac{\varphi(\gamma^2)}{p}\right) = \left(\frac{\varphi(\gamma)\varphi(\gamma)}{p}\right) = \left(\frac{\varphi(\gamma)}{p}\right)^2 = 1.$$

Demikian juga kita dapatkan $\chi_{\mathfrak{p}}(f'(\alpha)^2) = 1$. Jadi, menggunakan simbol Legendre, kita dapatkan

$$\begin{aligned}
1 &= \chi_{\mathfrak{p}}(\gamma^2) \\
&= \chi_{\mathfrak{p}}\left(f'(\alpha)^2 \prod_{(a,b)\in U}(a+\alpha b)\right) \\
&= \left(\frac{\varphi(f'(\alpha)^2 \prod_{(a,b)\in U}(a+\alpha b))}{p}\right) \\
&= \left(\frac{\varphi(f'(\alpha)^2)\prod_{(a,b)\in U}\varphi(a+\alpha b)}{p}\right) \\
&= \left(\frac{\varphi(f'(\alpha)^2)}{p}\right)\left(\frac{\prod_{(a,b)\in U}\varphi(a+\alpha b)}{p}\right) \\
&= \chi_{\mathfrak{p}}(\varphi(f'(\alpha)^2))\left(\frac{\prod_{(a,b)\in U}(a+rb)}{p}\right) \\
&= \prod_{(a,b)\in U}\left(\frac{a+rb}{p}\right).
\end{aligned}$$

Selesailah pembuktian teorema 111. Test kuadrat sesuai dengan teorema 111 dapat digunakan untuk meningkatkan jaminan bahwa produk menghasilkan kuadrat. Tentunya jika ternyata kuadrat tidak ditemukan, produk lain harus dicari.

Mari kita ringkas bagaimana apa yang sudah dijelaskan mengenai metode *number field sieve* digunakan untuk menguraikan suatu bilangan n:

1. Berdasarkan nilai n, suatu *irreducible polynomial* dengan *degree* d dipilih.

2. Penyaringan pertama menggunakan *rational factor base* yaitu sekumpulan bilangan prima.

3. Penyaringan kedua menggunakan *algebraic factor base* yaitu sekumpulan *first degree prime ideals* yang direpresentasikan menggunakan pasangan-pasangan (r, p).

4. Penyaringan ketiga dengan *quadratic characters* sesuai dengan teorema 111 menggunakan sekumpulan *first degree prime ideals*.

5. Setelah kuadrat-kuadrat ditemukan, kedua faktor dari n didapat menggunakan *difference of squares*.

Kompleksitas metode *number field sieve*, diestimasikan adalah

$$O(e^{C(\log n)^{1/3}(\log\log n)^{2/3}}).$$

Implementasi dari metode *number field sieve* tidak dijelaskan disini. Untuk pembaca yang ingin mengetahui lebih rinci mengenai implementasi dan estimasi kompleksitas dari metode ini dipersilahkan untuk membaca [buh93].

14.7 Ringkasan

Bab ini telah membahas penguraian bilangan bulat, topik yang sangat penting untuk kriptografi *public key*. Metode untuk menguraikan bilangan bulat dapat digolongkan menjadi dua kategori: metode yang bersifat Las Vegas dan metode yang menggunakan *factor base*. Contoh metode yang bersifat Las Vegas adalah metode Rho, sedangkan untuk yang menggunakan *factor base* telah dibahas metode Dixon, metode *continued fraction*, metode *quadratic sieve* dan metode *number field sieve*. Untuk menguraikan bilangan hingga 100 digit, metode *quadratic sieve* adalah yang tercepat, sedangkan untuk menguraikan bilangan lebih dari 100 digit hingga lebih dari 150 digit, metode *number field sieve* adalah yang tercepat. Secara umum, penguraian bilangan yang lebih besar dari 200 digit masih belum terjangkau. Akan tetapi pada tanggal 12 Desember 2009, sekelompok peneliti berhasil menguraikan kunci RSA sebesar 768 bit (232 digit) menggunakan metode *number field sieve* [kle10]. Penguraian tersebut memakan waktu $2\frac{1}{2}$ tahun menggunakan ratusan komputer yang tersebar di beberapa negara. Pencarian *polynomial* memakan waktu 6 bulan, *sieving* memakan waktu 2 tahun dan lainnya sekitar 2 minggu.

Banyak teknik-teknik implementasi yang tidak dibahas dalam bab ini. Untuk mengimplementasi metode penguraian bilangan bulat tentunya pembaca perlu mempelajari teknik-teknik tersebut. Misalnya untuk metode yang menggunakan *factor base*, algoritma *Block Lanczos* dapat digunakan untuk komputasi matrik.

Bab 15

Matematika VIII - Logaritma Diskrit

Jika $\mathbf{GF}(q)$ merupakan finite field dan $u, g \in \mathbf{GF}(q)^*$ (g biasanya *generator* untuk $\mathbf{GF}(q)^*$), maka logaritma diskrit dari u dengan basis g dalam $\mathbf{GF}(q)^*$ adalah bilangan bulat k, $0 \leq k < q-1$, dimana

$$u = g^k.$$

Sukarnya mengkalkulasi logaritma diskrit jika *finite field* sangat besar merupakan kunci dari keamanan berbagai algoritma kriptografi *public key* seperti Diffie-Hellman, DSA dan ElGamal. Sukarnya mengkalkulasi logaritma diskrit mirip dengan sukarnya menguraikan bilangan bulat yang sangat besar, bahkan perkembangan metode untuk mengkalkulasi logaritma diskrit saling mempengaruhi dengan perkembangan metode penguraian. Kita akan bahas tiga metode untuk kalkulasi logaritma diskrit yaitu metode Silver-Pohlig-Hellman, metode *baby steps - giant steps* dan metode *index calculus*.

15.1 Metode Silver-Pohlig-Hellman

Untuk bilangan prima p, $\mathbf{GF}(p)$ merupakan suatu *finite field*. Jika $p-1$ dapat diuraikan menjadi produk bilangan-bilangan prima yang kecil, maka logaritma diskrit dalam $\mathbf{GF}(p)^*$ dapat dikalkulasi dengan cepat menggunakan metode Silver-Pohlig-Hellman. Metode ini menggunakan fakta bahwa karena $\mathbf{GF}(p)^*$ mempunyai $p-1$ elemen, maka aritmatika *multiplicative group* untuk pangkat suatu elemen dari $\mathbf{GF}(p)^*$ dapat dilakukan modulo $p-1$. Berikut adalah bagaimana metode Silver-Pohlig-Hellman mencari solusi untuk k, $0 \leq k < p-1$, jika g *generator* untuk $\mathbf{GF}(p)^*$ dan $u = g^k$.

Langkah pertama dalam metode Silver-Pohlig-Hellman adalah mengkomputasi, untuk setiap bilangan prima p_i yang membagi $p-1$, akar-akar pangkat p_i dari 1,

$$r_{p_i,j} = g^{j(p-1)/p_i},$$

dimana g adalah *generator* untuk $\mathbf{GF}(p)^*$ dan $j = 0, 1, \ldots, p_i - 1$. Kita simpan $\{r_{p_i,j}\}_{0 \le j < p_i}$ dalam sebuah tabel.

Langkah berikutnya adalah mencari solusi untuk k modulo $p_i^{\alpha_i}$ untuk setiap bilangan prima p_i yang membagi $p-1$, dimana α_i adalah banyaknya p_i (pangkat dari p_i) dalam uraian $p-1$. Kita dapat tuliskan

$$k \equiv k_0 + k_1 p_i + k_2 p_i^2 + \ldots + k_{\alpha_i - 1} p_i^{\alpha_i - 1} \pmod{p_i^{\alpha_i}}.$$

Jika kita bisa temukan $k_0, k_1, \ldots, k_{\alpha_i - 1}$ maka kita akan dapatkan nilai untuk $k \pmod{p_i^{\alpha_i - 1}}$. Kita akan gunakan aritmatika $\mathbf{GF}(p)$ dalam mencari k_0. Pertama, kita kalkulasi $u^{(p-1)/p_i}$ untuk mendapatkan akar pangkat p_i dari 1 (karena $u^{p-1} = 1$). Karena $u = g^k$ dan $(k_1 p_i + k_2 p_i^2 + \ldots + k_{\alpha_i - 1} p_i^{\alpha_i - 1})/p_i$ adalah suatu bilangan bulat, kita dapatkan

$$\begin{aligned} u^{(p-1)/p_i} &= g^{k(p-1)/p_i} \\ &= g^{(k_0 + k_1 p_i + k_2 p_i^2 + \ldots + k_{\alpha_i - 1} p_i^{\alpha_i - 1})(p-1)/p_i} \\ &= g^{k_0 (p-1)/p_i} \\ &= r_{p_i, k_0}. \end{aligned}$$

Jadi kita dapat periksa tabel $\{r_{p_i,j}\}_{0 \le j < p_i}$ untuk mencari m dimana $r_{p_i,m} = r_{p_i,k_0}$ dan mendapatkan $k_0 = m$. Untuk mencari k_1, kita buat $u_1 = u/g^{k_0}$, jadi logaritma diskrit dari u_1 adalah

$$k - k_0 \equiv k_1 p_i + k_2 p_i^2 + \ldots + k_{\alpha_i - 1} p_i^{\alpha_i - 1} \pmod{p_i^{\alpha_i}}.$$

Karena u_1 adalah pemangkatan dengan p_i, $u_1^{(p-1)/p_i} = 1$ maka kita dapatkan

$$\begin{aligned} u_1^{(p-1)/p_i} &= g^{(k-k_0)(p-1)/p_i} \\ &= g^{(k_1 + k_2 p_i + \ldots + k_{\alpha_i - 1} p_i^{\alpha_i - 2})(p-1)/p_i} \\ &= g^{k_1 (p-1)/p_i} \\ &= r_{p_i, k_1}. \end{aligned}$$

Lagi, kita gunakan tabel $\{r_{p_i,j}\}_{0 \le j < p_i}$ untuk mendapatkan k_1. Jadi pola untuk mencari $k_0, k_1, \ldots, k_{\alpha_i - 1}$ sudah jelas. Untuk setiap $j = 1, 2, \ldots, \alpha_i - 1$, kita buat

$$u_j = u/g^{k_0 + k_1 p_i + \ldots + k_{j-1} p_i^{j-1}},$$

yang mempunyai logaritma diskrit ekuivalen dengan $k_j p_i^j + \ldots + k_{\alpha_i - 1} p_i^{\alpha_i - 1} \pmod{p_i^{\alpha_i}}$. Karena u_j merupakan pemangkatan dengan p_i^j, maka $u_j^{(p-1)/p_i^j} = 1$ dan kita dapatkan

$$\begin{aligned} u_j^{(p-1)/p_i^{j+1}} &= g^{(k_j + k_{j+1} p_i + \ldots + k_{\alpha_i - 1} p_i^{\alpha_i - j - 1})(p-1)/p_i} \\ &= g^{k_j (p-1)/p_i} \\ &= r_{p_i, k_j}. \end{aligned}$$

Kita cari k_j menggunakan tabel. Kita gabungkan hasilnya untuk mendapatkan

$$k \equiv k_0 + k_1 p_i + k_2 p_i^2 + \ldots + k_{\alpha_i - 1} p_i^{\alpha_i - 1} \pmod{p_i^{\alpha_i}}.$$

Langkah terahir setelah mendapatkan k modulo $p_i^{\alpha_i}$ untuk setiap p_i yang membagi p adalah menggabungkan hasil menggunakan *Chinese Remainder Theorem* (lihat bagian 10.2) untuk mendapatkan k modulo p.

Sebagai contoh penggunaan metode Silver-Pohlig-Hellman mari kita cari logaritma diskrit dari 28 dengan basis 2 dalam $\mathbf{GF}(37)^*$ (2 adalah *generator* untuk $\mathbf{GF}(37)^*$). Jadi kita ingin solusi untuk k dimana

$$28 \equiv 2^k \pmod{37}.$$

Karena $p = 37$ maka

$$p - 1 = 37 - 1 = 2^2 \cdot 3^2.$$

Jadi $p_1 = 2$, $\alpha_1 = 2$, $p_2 = 3$ dan $\alpha_2 = 2$. Untuk langkah pertama, kita dapatkan

$$\begin{aligned} 2^0 &\equiv 1 \pmod{37}, \\ 2^{36/2} &\equiv 36 \pmod{37}, \\ 2^0 &\equiv 1 \pmod{37}, \\ 2^{36/3} &\equiv 26 \pmod{37}, \\ 2^{36 \cdot 2/3} &\equiv 10 \pmod{37}. \end{aligned}$$

Jadi langkah pertama menghasilkan tabel

$$\begin{aligned} r_{2,0} &= 1, \\ r_{2,1} &= 36, \\ r_{3,0} &= 1, \\ r_{3,1} &= 26, \\ r_{3,2} &= 10. \end{aligned}$$

Berikutnya untuk $p_1 = 2$ kita kalkulasi $28^{36/2} \equiv 1 \pmod{37}$. Dari tabel kita dapatkan $r_{2,0}$, jadi $k_0 = 0$. Kemudian kita kalkulasi $28^{36/4} \equiv 36 \pmod{37}$, dan dari tabel kita dapatkan $k_1 = 1$. Jadi $k = 0 + 1 \cdot 2 = 2$ dan kita dapatkan

$$k \equiv 2 \pmod{4}.$$

Berikutnya untuk $p_2 = 3$ kita kalkulasi $28^{36/3} \equiv 26 \pmod{37}$. Dari tabel kita dapatkan $k_0 = 1$. Kemudian kita kalkulasi $14^{36/9} \equiv 10 \pmod{37}$, dan dari tabel kita dapatkan $k_1 = 2$. Jadi $k = 1 + 2 \cdot 3 = 7$ dan kita dapatkan

$$k \equiv 7 \pmod{9}.$$

Menggunakan *Chinese Remainder Theorem* kita dapatkan $k = 34$, jadi

$$28 \equiv 2^{34} \pmod{37}.$$

Tentunya contoh diatas tidak mencerminkan potensi metode karena hanya melibatkan bilangan-bilangan yang kecil. Metode Silver-Pohlig-Hellman cukup efisien jika setiap bilangan prima yang membagi $p-1$ relatif kecil, meskipun p sendiri sangat besar. Akan tetapi jika ada bilangan prima yang besar (contohnya 50 digit) yang membagi $p-1$ maka metode Silver-Pohlig-Hellman tidak akan berdaya karena perlu membuat tabel yang sangat besar untuk langkah pertama.

Ada varian metode Silver-Pohlig-Hellman yang menggunakan metode *baby steps - giant steps* untuk mencari $k_0, k_1, \ldots, k_{\alpha_i - 1}$. Untuk mencari k_0, kita buat

$$v = u^{p-1/p_i} = (g^{p-1/p_i})^{k_0} = h^{k_0}$$

dimana $h = g^{p-1/p_i}$. Kita pecahkan $v = h^{k_0}$ menggunakan metode *baby steps - giant steps* dimana *order* dari h adalah $n = (p-1)/p_i$.

15.2 Metode Baby Steps - Giant Steps

Jika $v, h \in \mathbf{GF}(p)^*$, h mempunyai *order* n dan $v \equiv h^a \pmod{p}$, maka metode *baby steps - giant steps* mencari a dimana $0 \leq a < n$ sebagai berikut.

- Kita buat $m = \lceil \sqrt{n} \rceil$. Maka terdapat bilangan-bilangan bulat a_0 dan a_1 yang menjadikan $a = a_0 + ma_1$ dimana $0 \leq a_0, a_1 < m$ (a_0 adalah *remainder* dan a_1 adalah *quotient*, lihat teorema 4).
- Komputasi *baby steps* $h^i \pmod{p}$ untuk $i = 0, 1, \ldots, m-1$. Hasil komputasi disimpan dalam struktur yang memudahkan pencarian nilai.
- Komputasi $u \equiv h^{-m} \pmod{p}$.
- Komputasi *giant steps* $vu^j \pmod{p}$ untuk $j = 0, 1, \ldots, m-1$. Periksa nilai apakah ada dalam hasil *baby steps*. Jika ada maka kita selesai.

Jika nilai *giant step* ditemukan dalam *baby steps* maka kita dapatkan

$$h^i \equiv v/h^{mj} \pmod{p}$$

jadi kita dapatkan nilai a yaitu $a = i + mj$.

Sebagai contoh penggunaan metode *baby steps - giant steps*, ambil contoh dari bagian 15.1 dan gunakan metode ini dalam mencari k_1 untuk $p_2 = 3$, dimana $v = 14^4 \equiv 10 \pmod{37}$, $h = 2^{12} \equiv 26 \pmod{37}$ dan $n = 2$ (jadi $m = 2$). Kita harus mencari solusi k_1 untuk

$$10 \equiv 26^{k_1} \pmod{37}.$$

Komputasi *baby steps* $h^i \pmod{37}$ menghasilkan

$$\begin{aligned} h^0 &\equiv 1 \pmod{37}, \\ h^1 &\equiv 26 \pmod{37}. \end{aligned}$$

Komputasi $u \equiv h^{-m} \pmod{37}$ menghasilkan

$$u \equiv 10^{-1} \equiv 26 \pmod{37}.$$

Komputasi *giant steps* vu^j menghasilkan

$$\begin{aligned} vu^0 &\equiv 10 \pmod{37}, \\ vu^1 &\equiv 1 \pmod{37}. \end{aligned}$$

Dari *baby step* pertama dan *giant step* terahir kita dapatkan $i = 0$ dan $j = 1$. Jadi

$$k_1 = 0 + 2 \cdot 1 = 2.$$

Hasil ini sesuai dengan yang kita dapatkan di bagian 15.1.

Metode *baby steps - giant steps* tidak akan efektif jika *order* dari basis (n) sangat besar. Contohnya jika n besarnya melebihi 200 digit (jadi $\sqrt{n}$ melebihi 100 digit), maka meskipun setiap partikel fundamental dapat digunakan untuk menyimpan data, jumlah partikel fundamental di seantero jagat raya (estimasi antara 10^{72} sampai dengan 10^{87}) tidak akan cukup untuk menyimpan *baby steps*.

15.3 Metode Index Calculus

Metode *index calculus* untuk mengkalkulasi logaritma diskrit sangat mirip dengan metode-metode untuk penguraian bilangan bulat yang menggunakan *factor base*. Karena kemiripan ini, kita hanya akan bahas "kerangka" dari metode ini dan fokus pada perbedaannya. Perbedaan utamanya adalah jika dalam penguraian menggunakan *factor base* kita menggunakan matrik dimana setiap baris merepresentasikan produk pemangkatan elemen-elemen *factor base*, dalam metode *index calculus* setiap baris dalam matrik merepresentasikan relasi

antara produk pemangkatan elemen-elemen *factor base* dengan pemangkatan basis. Sebagai contoh, baris

$$[1 \quad 0 \quad 1 \quad 2 \quad 17]$$

merepresentasikan relasi

$$p_1 p_3 p_4^2 \equiv g^{17},$$

dimana p_1, p_2, p_3, p_4 adalah elemen-elemen *factor base* dan g adalah basis untuk logaritma diskrit.

Untuk memudahkan pembahasan, kita akan fokus pada $\mathbf{GF}(p)$ dengan p bilangan prima, sehingga setiap elemen *factor base* adalah bilangan prima (kadang -1 diperbolehkan). Untuk $\mathbf{GF}(p^n)$, kita dapat menggunakan *irreducible polynomials* sebagai elemen-elemen *factor base*.

Kita mulai penjelasan metode *index calculus*, dimana

$$y \equiv g^x \pmod{p}$$

dan x harus dicari untuk suatu y. Langkah pertama yang harus dilakukan adalah menentukan *factor base*. Contohnya, kita dapat memilih himpunan n bilangan prima pertama $\{p_1, p_2, \ldots, p_n\}$ sebagai *factor base*. Komputasi logaritma diskrit terdiri dari dua tahap yaitu tahap prekomputasi dan tahap komputasi.

Pada tahap pertama, data logaritma diskrit untuk elemen-elemen *factor base* dihimpun dalam sebuah matrik. Ini mirip dengan metode Silver-Pohlig-Hellman dimana data dihimpun dalam suatu tabel. Tahap ini diawali dengan matrik yang kosong. Untuk beberapa k, dengan $1 \leq k < p$ (k dapat dipilih secara sembarang atau kita dapat menggunakan $k = 1, 2, \ldots$), kita ulang langkah-langkah berikut hingga banyaknya baris dalam matrik sama dengan banyaknya bilangan prima dalam *factor base* (r):

- Coba uraikan $g^k \pmod{p}$ menggunakan *factor base* menjadi $p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$.
- Jika berhasil diuraikan dan hasilnya *linearly independent* dengan matrik, maka tambahkan hasil sebagai baris dalam matrik.
- Jika banyaknya baris sama dengan r, kita selesai.

Penguraian menggunakan *factor base* adalah hal yang mudah karena hanya perlu membagi dengan elemen-elemen *factor base*. Setelah itu matrik dibuat menjadi *reduced echelon form*. Sebagai contoh, berikut adalah suatu matrik dalam *reduced echelon form*:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 6 \end{bmatrix}.$$

Elemen pertama pada kolom terahir adalah $\log_g(p_1)$ (logaritma diskrit untuk p_1), elemen kedua adalah $\log_g(p_2)$, dan seterusnya. Selesailah tahap pertama. Tahap ini hanya perlu dilakukan satu kali untuk berbagai logaritma diskrit dengan basis dan modulus yang sama.

Pada tahap kedua, logaritma diskrit dari y yaitu x dicari. Kita cari menggunakan beberapa s, dengan $0 \leq s < p$ (s dapat dipilih secara sembarang atau kita dapat menggunakan $s = 0, 1, 2, \ldots$) sehingga $g^s y$ dapat diuraikan sebagai berikut:

$$g^s y \equiv p_1^{f_1} p_2^{f_2} \cdots p_r^{f_r}.$$

Kita dapatkan x:

$$x = f_1 \log_g(p_1) + f_2 \log_g(p_2) + \ldots f_r \log_g(p_r) - s.$$

Sebagaimana halnya dengan penguraian menggunakan *factor base*, tidak ada jaminan secara umum bahwa logaritma diskrit dapat dicari dengan efisien menggunakan metode *index calculus*.

15.4 Ringkasan

Bab ini telah membahas logaritma diskrit, topik yang sangat penting untuk kriptografi *public key*. Sukarnya mengkalkulasi logaritma diskrit mirip dengan sukarnya menguraikan bilangan bulat. Bahkan teknik-teknik yang digunakan untuk mengkalkulasi logaritma diskrit diadaptasi untuk penguraian bilangan bulat dan sebaliknya. Logaritma diskrit untuk $\mathbf{GF}(2^n)$ lebih mudah untuk dikalkulasi dibandingkan $\mathbf{GF}(p^n)$ dimana p adalah bilangan prima ganjil (lihat [cop84]). Oleh sebab itu penggunaan $\mathbf{GF}(2^n)$ dalam kriptografi *public key* tidak direkomendasikan kecuali n sangat besar (> 2000).

Banyak teknik-teknik implementasi yang tidak dibahas dalam bab ini. Untuk mengimplementasi metode logaritma diskrit tentunya pembaca perlu mempelajari teknik-teknik tersebut. Misalnya untuk metode *index calculus*, algoritma *Block Lanczos* dapat digunakan untuk komputasi matrik.

Bab 16

Kriptografi Public Key

Dalam kriptografi klasik (simetris), jika seseorang mengetahui cara mengenkripsi naskah asli menjadi naskah acak, maka orang tersebut juga mengetahui cara mendekripsi naskah acak yang dihasilkan. Demikian juga jika seseorang mengetahui cara mendekripsi naskah acak, maka orang tersebut juga mengetahui cara mengenkripsi naskah asli untuk menghasilkan naskah acak.

Sekitar pertengahan tahun 1970an, muncul konsep baru dalam kriptografi yaitu kriptografi *public key* (asimetris). Seseorang yang mengetahui cara mengenkripsi naskah asli belum tentu mengetahui juga cara mendekripsi naskah acak yang dihasilkan. Demikian juga seseorang yang mengetahui cara mendekripsi naskah acak belum tentu mengetahui juga cara mengenkripsi naskah asli untuk menghasilkan naskah acak tersebut. Enkripsi dan dekripsi dalam kriptografi *public key* menggunakan sepasang kunci yaitu kunci publik (*public key*) dan kunci privat (*private key*). Naskah yang telah dienkripsi menggunakan kunci privat hanya dapat didekripsi menggunakan kunci publik dan naskah yang dapat didekripsi menggunakan kunci publik dapat dipastikan telah dienkripsi menggunakan kunci privat. Sebaliknya, naskah yang telah dienkripsi menggunakan kunci publik hanya dapat didekripsi menggunakan kunci privat. Mekanisme ini memungkinkan berbagai aplikasi, dua yang terpenting diantaranya adalah distribusi kunci sesi dan tanda tangan digital (*digital signature*).

Jika A ingin mengirim kunci sesi atau rahasia lainnya ke B dan hanya ingin B yang dapat membacanya, maka A mengenkripsi rahasia tersebut menggunakan kunci publik milik B. Dengan asumsi hanya B yang memiliki kunci privat B, maka hanya B yang dapat mendekripsi rahasia yang telah dienkripsi tersebut. Cara inilah yang kerap digunakan untuk mendistribusikan kunci sesi menggunakan kriptografi *public key*.

Jika A ingin menanda-tangan suatu naskah secara digital, maka A mengen-

kripsi naskah tersebut menggunakan kunci privat miliknya dan hasil enkripsi merupakan "tanda tangan," Jika seseorang (sebut saja B) ingin memeriksa apakah naskah tersebut telah ditanda-tangan oleh A, maka B mendekripsi "tanda tangan" tersebut dengan kunci publik milik A dan membandingkan hasil dekripsi dengan naskah yang ditanda-tangan. Jika sama maka B dapat meyakinkan dirinya sendiri bahwa A telah menanda-tangan naskah tersebut karena hanya A yang memiliki kunci privat yang digunakan untuk mengenkripsi naskah untuk menghasilkan "tanda-tangan." Dalam prakteknya, yang dienkripsi bukan naskah penuh melainkan *digest* dari naskah tersebut (lihat bab 9).

Jadi dalam kriptografi *public key*, kunci publik dapat disebar-luaskan kepada umum dan sebaiknya disebar luaskan. Sebaliknya, kunci privat harus dirahasiakan oleh pemiliknya.

Dalam bab ini akan dibahas empat sistem kriptografi *public key* yang di buku ini dianggap sebagai empat yang terpenting yaitu RSA, Diffie-Hellman, DSA dan ElGamal. Keamanan RSA mengandalkan sukarnya menguraikan bilangan yang sangat besar, sedangkan keamanan tiga sistem lainnya mengandalkan sukarnya mencari logaritma diskrit. Selain itu kriptografi *knapsack* juga dibahas karena merupakan bagian dari sejarah. Protokol *zero-knowledge* dibahas karena potensinya untuk aplikasi identifikasi.

16.1 RSA

Tahun 1978, Len Adleman, Ron Rivest dan Adi Shamir mempublikasikan sistem RSA (lihat [adl78]). Semula sistem ini dipatenkan di Amerika Serikat dan seharusnya masa paten habis tahun 2003, akan tetapi RSA Security melepaskan hak paten setelah 20 September 2000. Sebetulnya sistem serupa telah dilaporkan oleh Clifford Cocks tahun 1973 meskipun informasi mengenai ini baru dipublikasi tahun 1997 karena merupakan hasil riset yang diklasifikasikan sangat rahasia oleh pemerintah Britania Raya (Clifford Cocks bekerja untuk GCHQ, suatu badan di Britania Raya yang fungsinya serupa dengan fungsi NSA di Amerika Serikat), jadi validitas paten patut dipertanyakan karena adanya *prior art*.

Kita jelaskan secara garis besar bagaimana cara kerja RSA. Setiap pengguna memilih, menggunakan *random number generator*, dua bilangan prima yang sangat besar p dan q (masing-masing lebih dari 200 digit). Untuk produk $n = pq$, jika p dan q diketahui, fungsi Euler dapat dengan mudah dikomputasi yaitu

$$\phi(n) = (p-1)(q-1).$$

Kemudian pengguna memilih, menggunakan *random number generator*, suatu bilangan e antara 1 dan $\phi(n)$ yang koprima dengan $\phi(n)$. Berikutnya pengguna

mengkomputasi *inverse* dari e modulo $\phi(n)$:

$$d \equiv e^{-1} \pmod{\phi(n)}.$$

Ini dapat dilakukan menggunakan *extended Euclidean algorithm* (lihat pembahasan kalkulasi *inverse* modulo bilangan yang koprima menggunakan *extended Euclidean algorithm* di bagian 3.5). Pengguna kemudian mempublikasi kunci publiknya

$$K_E = (n, e),$$

dan merahasiakan kunci privatnya

$$K_D = (n, d).$$

Rumus untuk mengenkripsi atau mendekripsi menggunakan kunci publik adalah

$$M^e \bmod n$$

dimana M adalah representasi naskah asli (menggunakan bilangan bulat) jika mengenkripsi, atau representasi naskah acak jika mendekripsi. Rumus untuk mengenkripsi atau mendekripsi menggunakan kunci privat adalah

$$M^d \bmod n$$

dimana M adalah representasi naskah asli jika mengenkripsi, atau representasi naskah acak jika mendekripsi. Tidak terlalu sulit untuk melihat bahwa naskah yang dienkripsi menggunakan kunci publik dapat didekripsi menggunakan kunci privat:

$$(M^e)^d \equiv M^{ed} \equiv M \pmod{n}.$$

Jika $\gcd(M, n) = 1$, maka menggunakan teorema 32 kita dapatkan

$$\begin{aligned} (M^e)^d &\equiv M^{ed} \\ &\equiv M^{\phi(n)+1} \\ &\equiv M^{\phi(n)} M \\ &\equiv M \pmod{n}. \end{aligned}$$

Untuk $\gcd(M, n) > 1$ dimana M bukan kelipatan n, ini hanya bisa terjadi jika M merupakan kelipatan p atau kelipatan q, tetapi bukan kelipatan keduanya. Jika M merupakan kelipatan p, maka

$$M^{ed} \equiv 0 \pmod{p}$$

dan

$$M^{ed} \equiv M^{\phi(p)\phi(q)+1} \equiv M^{\phi(p)\phi(q)} M \equiv M \pmod{q}.$$

Menggunakan *Chinese Remainder Theorem*, dengan $n = pq$, kita dapatkan

$$M^{ed} \equiv M \pmod{n}.$$

Untuk M kelipatan q, hal serupa dapat ditunjukkan. Untuk M kelipatan n, kita dapatkan

$$(M^e)^d \equiv 0 \equiv M \pmod{n}$$

Jadi

$$(M^e)^d \equiv M \pmod{n}$$

untuk sembarang M.

Naskah yang dienkripsi menggunakan kunci privat dapat didekripsi menggunakan kunci publik:

$$(M^d)^e \equiv M^{de} \equiv M \pmod{n}.$$

Secara umum, jika $f \not\equiv d \pmod{\phi(n)}$ maka

$$(M^e)^f \not\equiv M \pmod{n},$$

yang berarti sesuatu yang dienkripsi menggunakan kunci publik tidak dapat didekripsi selain menggunakan kunci privat. Juga, jika $f \not\equiv e \pmod{\phi(n)}$ maka secara umum

$$(M^d)^f \not\equiv M \pmod{n},$$

yang berarti sesuatu yang dienkripsi menggunakan kunci privat tidak dapat didekripsi selain menggunakan kunci publik.

Keamanan dari RSA bersandar pada fakta bahwa mengetahui n dan d secara umum tidak membantu untuk mencari e yaitu *inverse* modulo $\phi(n)$ dari d. Hal ini karena mengetahui n tidak membantu mencari $\phi(n)$ jika n tidak bisa diuraikan menjadi

$$n = pq.$$

Untuk menjaga keamanan tersebut, ada beberapa hal yang perlu diperhatikan dalam memilih p dan q:

- Nilai p harus cukup jauh dari nilai q. Sebaiknya panjang dari p harus berbeda beberapa digit dari q. Jika nilai p terlalu dekat dengan nilai q, maka *Fermat factorization* dapat digunakan untuk menguraikan $n = pq$ (lihat bagian 14.2).

- Sebaiknya $\gcd(p-1, q-1)$ tidak terlalu besar.

- Sebaiknya $p-1$ dan $q-1$ mempunyai faktor prima yang besar.

Karena perkembangan yang pesat dalam teknik penguraian, kunci RSA sebaiknya minimal 2048 bit.

RSA dapat digunakan, baik untuk *key distribution* (termasuk *key exchange*), maupun untuk *digital signature*. Karena merupakan sistem pertama yang dapat digunakan untuk *key distribution* dan *digital signature*, RSA menjadi sistem kriptografi *public key* yang terpopuler. Boleh dikatakan semua standard protokol kriptografi memperbolehkan penggunaan RSA, termasuk SSL/TLS (untuk pengamanan http) dan SSH (*secure shell*).

Pembahasan RSA diatas tidak menjelaskan standard implementasi secara rinci, termasuk format data dan kunci. Untuk yang ingin mengetahui standard RSA dengan lebih rinci dipersilahkan membaca berbagai publikasi RSA Laboratories (bagian dari RSA Security) dalam seri yang berjudul *Public-Key Cryptography Standards* (PKCS).

16.2 Diffie-Hellman

Walaupun Diffie-Hellman adalah sistem kriptografi *public key* yang pertama, Diffie-Hellman tidak sepopuler RSA karena hanya dapat digunakan untuk *key agreement*. Menggunakan Diffie-Hellman, dua pengguna, sebut saja A dan B, dapat membuat kunci rahasia yang hanya diketahui oleh A dan B, meskipun komunikasi antara A dan B dapat dilihat semua orang.

Diffie-Hellman menggunakan *finite field* $\mathbf{GF}(q)$ yang sangat besar. A dan B keduanya mengetahui $\mathbf{GF}(q)$ dan elemen $g \in \mathbf{GF}(q)^*$. $\mathbf{GF}(q)$ dan g tidak perlu dirahasiakan, jadi boleh saja diketahui semua orang. Meskipun tidak harus, g sebaiknya merupakan *generator* untuk $\mathbf{GF}(q)^*$, atau setidaknya memiliki *order* yang besar agar *range* untuk pembuatan kunci cukup besar. Diffie-Hellman bekerja sebagai berikut:

- A memilih, menggunakan *random number generator*, a, mengkomputasi $g^a \in \mathbf{GF}(q)^*$, dan mengirim g^a ke B.
- B melakukan hal yang serupa, yaitu memilih, menggunakan *random number generator*, b, mengkomputasi $g^b \in \mathbf{GF}(q)^*$, dan mengirim g^b ke A.
- Setelah menerima g^b, A mengkomputasi kunci rahasia $k = (g^b)^a = g^{ab} \in \mathbf{GF}(q)^*$.
- B, setelah menerima g^a, mengkomputasi kunci rahasia $k = (g^a)^b = g^{ab} \in \mathbf{GF}(q)^*$.

Setelah selesai, A dan B mengetahui kunci rahasia $k = g^{ab}$, akan tetapi orang lain tidak bisa mendapatkan $k = g^{ab}$ meskipun mengetahui g, g^a, dan g^b. Ini didasarkan pada asumsi bahwa untuk mendapatkan g^{ab}, orang lain harus

mengkomputasi logaritma diskrit dari g^a atau g^b untuk mendapatkan a atau b terlebih dahulu, dan komputasi logaritma diskrit terlalu sukar.

Satu catatan yang perlu diperhatikan adalah jika *finite field* yang digunakan adalah $\mathbf{GF}(p^n)$ dimana p tentunya adalah bilangan prima dan $n > 1$. Setiap bilangan bulat x dimana $0 \leq x < p^n$ perlu dipetakan ke $\mathbf{GF}(p^n)$. Karena setiap elemen dalam $\mathbf{GF}(p^n)$ dapat dianggap sebagai n-tuple, maka ada pemetaan yang *bijective* antara $\mathbf{GF}(p^n)$ dengan $\mathbf{Z}/p^n\mathbf{Z}$ karena setiap n-tuple dapat diinterpretasikan sebagai suatu bilangan dimana setiap koordinat merepresentasikan suatu digit dengan basis p. Akan tetapi $\mathbf{GF}(p^n)$ dan $\mathbf{Z}/p^n\mathbf{Z}$ mempunyai struktur yang berbeda karena yang pertama adalah suatu *finite field* sedangkan yang kedua adalah suatu *ring* dimana setiap kelipatan p tidak mempunyai *inverse*. Jadi kita tidak bisa begitu saja menggunakan aritmatika modular bilangan bulat untuk aritmatika $\mathbf{GF}(p^n)$.

$\mathbf{GF}(2^n)$ banyak digunakan sebagai *finite field* untuk Diffie-Hellman karena komputasi dalam $\mathbf{GF}(2^n)$ sangat elegan dan mudah diprogram. Akan tetapi, karena logaritma diskrit lebih mudah dikomputasi untuk $\mathbf{GF}(2^n)^*$ dibandingkan $\mathbf{GF}(p)^*$, dimana p merupakan bilangan prima ganjil yang besarnya hampir sama dengan 2^n, maka n harus dipilih sangat besar (lebih dari 2000). Buku ini merekomendasikan penggunaan $\mathbf{GF}(p)$.

16.3 DSA

DSA (*Digital Signature Algorithm*) adalah salah satu algoritma yang digunakan dalam DSS (*Digital Signature Standard*), standard untuk *digital signature* yang dibuat oleh FIPS. DSS juga memperbolehkan penggunaan RSA. Karena DSS mewajibkan penggunaan SHA-1 (lihat bagian 9.2), maka DSA atau RSA digunakan untuk mengenkripsi *digest* sebesar 160 bit.

Parameter yang digunakan oleh DSA adalah sebagai berikut:

- Suatu bilangan prima p yang dipilih menggunakan *random number generator* minimum 512 bit, sebaiknya 1024 bit.

- Suatu bilangan prima q yang dipilih menggunakan *random number generator* sebesar 160 bit dimana q membagi $p - 1$. Untuk implementasi, mungkin lebih mudah untuk memilih q terlebih dahulu kemudian memilih p dimana $p \equiv 1 \pmod{q}$.

- Suatu bilangan $g \in \mathbf{GF}(p)^*$ yang mempunyai *order* q yang dipilih sebagai berikut: Pilih bilangan g_0 menggunakan *random number generator* dimana $1 < g_0 < p - 1$ lalu komputasi $g = g_0^{(p-1)/q} \bmod p$. Jika $g > 1$ maka itulah g yang dipilih. Jika tidak maka pilih g_0 yang lain sampai kita dapatkan $g > 1$.

- Suatu bilangan x yang dipilih menggunakan *random number generator* dimana $0 < x < q$.
- Bilangan $y = g^x \bmod p$.
- Suatu bilangan k yang dipilih menggunakan *random number generator* dimana $0 < k < q$. Setiap kali menanda-tangan, k yang baru harus dipilih.

Parameter p, q, g, x dan y merupakan parameter semi-permanen, sedangkan k yang baru harus dipilih setiap kali menanda-tangan. Parameter p, q, g dan y merupakan parameter publik, jadi harus disebar-luaskan, sedangkan parameter x dan k harus dirahasiakan.

Berikut adalah cara DSA membuat *digital signature* untuk naskah M menggunakan parameter diatas. Hasil pembuatan *digital signature* adalah sepasang bilangan bulat (r, s) dimana $0 \leq r, s < q$. Rumus untuk r dan s adalah sebagai berikut:

$$\begin{aligned} r &= (g^k \bmod p) \bmod q, \\ s &= (k^{-1}(\text{SHA-1}(M) + xr) \bmod q, \end{aligned}$$

dimana SHA-1(M) adalah hasil komputasi *digest* menggunakan SHA-1 (lihat bagian 9.2). Jika ternyata $r = 0$ atau $s = 0$ (kemungkinannya sangat kecil), maka pembuatan *digital signature* dapat diulang menggunakan nilai k yang lain.

Seseorang yang ingin memeriksa *digital signature* akan mendapatkan M, r dan s. Pertama r dan s diperiksa nilainya apakah $0 < r < q$ dan $0 < s < q$. Jika tidak maka *digital signature* ditolak. Pemeriksa kemudian mengkomputasi:

$$\begin{aligned} w &= s^{-1} \bmod q, \\ u_1 &= \text{SHA-1}(M)w \bmod q, \\ u_2 &= rw \bmod q, \text{ dan} \\ v &= (g^{u_1} y^{u_2} \bmod p) \bmod q. \end{aligned}$$

Jika $v = r$ maka pemeriksa dapat cukup yakin bahwa M telah ditanda-tangan menggunakan kunci pasangan dari y. Mari kita buktikan bahwa jika semua komputasi dilakukan sesuai dengan aturan DSA maka $v = r$. Untuk itu kita buat $M' = \text{SHA-1}(M)$, jadi

$$\begin{aligned} s &= (k^{-1}(M' + xr) \bmod q, \\ u_1 &= M'w \bmod q. \end{aligned}$$

Semua *inverse* dalam pembuktian adalah dalam $\mathbf{GF}(q)$. Menggunakan rumus komputasi kita dapatkan

$$v = (g^{u_1} y^{u_2} \bmod p) \bmod q$$

$$\begin{aligned} &= (g^{M'w} y^{rw} \bmod p) \bmod q \\ &= (g^{M'w} g^{xrw} \bmod p) \bmod q \\ &= (g^{(M'+xr)w} \bmod p) \bmod q. \end{aligned}$$

Karena

$$s = k^{-1}(M' + xr) \bmod q,$$

maka

$$w = k(M' + xr)^{-1} \bmod q,$$

jadi

$$(M' + xr)w \bmod q = k \bmod q.$$

Kembali ke v:

$$\begin{aligned} v &= (g^k \bmod p) \bmod q \\ &= r. \end{aligned}$$

Selesailah pembuktian kita.

Suatu hal yang menarik dengan DSA adalah fungsi dari parameter k. Setiap kali menanda-tangan, nilai k yang baru dipilih secara acak. Walaupun naskah yang ditanda-tangan sama, jika ditanda-tangan dua kali oleh pemilik a, tanda tangannya berbeda. Jadi k berfungsi seperti *initialization vector*.

16.4 ElGamal

Sistem kriptografi ElGamal menjadi populer ketika pengembang *open source software* untuk kriptografi mencari alternatif dari RSA yang ketika itu patennya masih berlaku. Seperti halnya dengan Diffie-Hellman dan DSA, keamanan ElGamal didasarkan atas sukarnya mengkomputasi logaritma diskrit. Akan tetapi berbeda dengan Diffie-Hellman yang khusus dirancang untuk *key agreement* dan DSA yang khusus dirancang untuk *digital signature*, ElGamal lebih seperti RSA karena fungsinya untuk enkripsi umum. Kita akan bahas ElGamal untuk enkripsi terlebih dahulu, kemudian dilanjutkan dengan pembahasan versi ElGamal untuk *digital signature*.

ElGamal menggunakan *finite field* $\mathbf{GF}(q)$ yang besar. Untuk memudahkan pembahasan, $q = p$, suatu bilangan prima yang sangat besar. Jika $q = p^n$ dimana $n > 1$ maka dalam pembahasan p harus diganti dengan q dan aritmatika yang digunakan bukan sekedar aritmatika modular bilangan bulat. Jadi disini *finite field* yang digunakan adalah $\mathbf{GF}(p)$.

Suatu $g \in \mathbf{GF}(p)^*$ dipilih yang sebaiknya adalah suatu *generator* untuk $\mathbf{GF}(p)^*$. Jika bukan *generator* maka sebaiknya g mempunyai *order* yang sangat besar dalam $\mathbf{GF}(p)^*$. Untuk membuat kunci privat, pengguna memilih,

menggunakan *random number generator*, a, dimana $0 < a < p-1$. Kunci publik pasangan a adalah $g^a \in \mathbf{GF}(p)^*$. Tentunya kunci publik perlu disebar-luaskan. Seseorang yang mengetahui kunci publik tidak bisa mendapatkan kunci privat (a) tanpa mengkomputasi logaritma diskrit dari g^a.

Seseorang yang ingin mengenkripsi suatu naskah P (representasi P menggunakan bilangan bulat harus lebih kecil dari p) untuk hanya dapat didekripsi oleh pemilik a memilih, menggunakan suatu *random number generator*, k, dan mengirim pasangan

$$(g^k, Pg^{ak})$$

kepada pemilik a. Ini dapat dilakukan karena pengirim mengetahui g, g^a, P dan k. Seperti halnya dengan DSA, k disini berfungsi sebagai *initialization vector*. Pemilik a dapat mendekripsi kiriman sebagai berikut:

- Mengkomputasi $(g^k)^{(p-1-a)} = g^{-ak}$. Ini dapat dilakukan karena pemilik a mengetahui g^k, p dan a.
- Mengalikan Pg^{ak} dengan g^{-ak} untuk mendapatkan P.

Jadi Pg^{ak} seolah naskah P menggunakan topeng atau *mask*, dan mengalikannya dengan g^{-ak} membuka topeng tersebut.

Satu hal yang menarik dengan ElGamal adalah kemampuan untuk *reencryption*. Jika g dan g^a diketahui, seseorang dapat membuat r secara acak dan mentransformasi pasangan (g^k, Pg^{ak}) menjadi $(g^{(k+r)}, Pg^{a(k+r)})$ (kalikan g^k dengan g^r dan Pg^{ak} dengan g^{ar}). Jadi *initialization vector* telah diganti dari k menjadi $k+r$, tanpa harus mengetahui k. Ini dapat digunakan untuk anonimitas, karena tanpa mengetahui a atau r, pasangan $(g^{(k+r)}, Pg^{a(k+r)})$ tidak dapat dikaitkan dengan (g^k, Pg^{ak}). Aplikasi anonimitas contohnya untuk *electronic voting*.

Seperti halnya dengan RSA, ElGamal dapat digunakan untuk *digital signature*. Berikut adalah cara pemilik a menanda-tangan suatu naskah S dimana representasi S menggunakan bilangan bulat lebih kecil dari $p-1$:

- Pilih, menggunakan *random number generator*, k yang koprima dengan $p-1$ ($\gcd(k, p-1) = 1$). Lakukan komputasi $r = g^k \bmod p$.
- Cari solusi untuk x dimana $g^S \equiv g^{ar} r^x \pmod{p}$.
- *Digital signature* untuk S adalah pasangan (r, x).

Solusi untuk x dapat dicari oleh pemilik a karena

$$\begin{aligned} g^S &\equiv g^{ar} r^x \pmod{p} \\ &\equiv g^{ar} g^{kx} \pmod{p} \\ &\equiv g^{ar+kx} \pmod{p}. \end{aligned}$$

Ini berarti

$$S \equiv ar + kx \pmod{p-1},$$

jadi

$$x = k^{-1}(S - ar) \bmod p - 1,$$

dimana *inverse* dikalkulasi dalam aritmatika modulo $p-1$. Seseorang yang ingin memeriksa tanda tangan diatas cukup memeriksa bahwa

$$g^S \equiv g^{ar} r^x \pmod{p}.$$

Ini dapat dilakukan oleh pemeriksa karena ia mengetahui g, S, g^a, r dan x.

16.5 Knapsack

Tahun 1970an banyak riset yang dilakukan guna menemukan mekanisme untuk kriptografi *public key*. Selain kriptografi yang berbasis pada sukarnya penguraian dan kriptografi yang berbasis pada sukarnya logaritma diskrit, ada juga yang mencoba membuat sistem kriptografi *public key* yang berbasis pada sukarnya mencari solusi untuk *knapsack problem*, dipelopori oleh Merkle dan Hellman.

Secara informal, *knapsack problem* adalah masalah bagaimana mengisi suatu *knapsack* yang mempunyai kapasitas tertentu dengan benda-benda dari sekumpulan yang mempunyai ukuran berbeda-beda sehingga *knapsack* terisi penuh sesuai kapasitas. Tidak semua benda dari kumpulan perlu dimasukkan kedalam *knapsack*. *Knapsack problem* dapat didefinisikan secara formal sebagai berikut.

Definisi 50 (Knapsack Problem) *Jika* $\{v_0, v_1, \ldots, v_{k-1}\}$ *adalah himpunan dengan* k *elemen, dimana setiap elemen* $v_i \in \mathbf{Z}, v_i > 0$*, dan* $V \in \mathbf{Z}, V > 0$*, cari solusi* $\{d_0, d_1, \ldots, d_{k-1}\}$ *dimana setiap* $d_i \in \{0, 1\}$ *dan*

$$\sum_{i=0}^{k-1} d_i v_i = V.$$

Bisa saja suatu *knapsack problem* tidak mempunyai solusi, atau ada solusi yang unik, atau ada lebih dari satu solusi. Secara umum *knapsack problem* adalah masalah yang tergolong *NP-complete* (bersama dengan *travelling salesman problem*), jadi terlalu sukar untuk dikomputasi. Namun ada jenis *knapsack problem* yang dapat dipecahkan dengan efisien yaitu *super-increasing knapsack problem* dimana, deretan $v_0, v_1, \ldots, v_{k-1}$ dapat diurutkan sehingga untuk setiap v_i:

$$v_i > \sum_{j=0}^{i-1} v_j.$$

Sebagai contoh, $2, 3, 7, 15, 31$ adalah deretan yang *super-increasing*. Untuk deretan yang *super-increasing* ada algoritma berikut untuk memecahkan *knapsack problem* dengan efisien.

1. $W \leftarrow V, j \leftarrow k$.
2. $j \leftarrow j - 1, d_j \leftarrow 0$.
3. Jika $v_j \leq W$ maka $d_j \leftarrow 1, W \leftarrow W - v_j$.
4. Jika $j > 0$ dan $W > 0$ kembali ke langkah 2.
5. Sukses jika $W = 0$ dan gagal jika $W > 0$.

Untuk deretan $2, 3, 7, 15, 31$ dan $V = 24$ kita dapatkan sukses dengan $d_0 = 1, d_1 = 0, d_2 = 1, d_3 = 1, d_4 = 0$.

Dasar dari kriptografi *knapsack* seperti Merkle-Hellman adalah

- untuk yang mengetahui kunci privat, esensi dekripsi adalah memecahkan *super-increasing knapsack problem*, sesuatu yang mudah; sedangkan
- untuk yang tidak mengetahui kunci privat, esensi dekripsi adalah memecahkan *knapsack problem* secara umum, sesuatu yang sangat sukar.

Berikut dijelaskan sistem Merkle-Hellman (lihat juga [hel78]). Dalam sistem Merkle-Hellman, nilai parameter k sama dengan banyaknya bit dalam unit naskah.

Untuk keperluan pasangan kunci, dipilih secara acak suatu deretan yang *super-increasing* $v_0, v_1, \ldots, v_{k-1}$ dan suatu bilangan bulat m dimana

$$\sum_{i=0}^{k-1} v_i < m.$$

Ini dapat dilakukan misalnya dengan memilih, menggunakan *random number generator*, sederetan bilangan bulat positif $z_0, z_1, \ldots, z_k$, lalu membuat

$$\begin{aligned} v_0 &\leftarrow z_0, \\ v_i &\leftarrow z_i + \sum_{j=0}^{i-1} v_j, \text{ untuk } 1 \leq i < k, \\ m &\leftarrow z_k + \sum_{i=0}^{k-1} v_i. \end{aligned}$$

Langkah selanjutnya adalah memilih suatu bilangan bulat a secara acak, dimana $0 < a < m$ dan $\gcd(a, m) = 1$. Ini dapat dilakukan dengan memilih,

menggunakan *random number generator*, suatu bilangan bulat a', dimana $0 < a' < m$ dan kemudian membuat a sebagai bilangan terkecil yang mematuhi $a' \leq a$ dan $\gcd(a, m) = 1$. Setelah itu dibuat b:

$$b = a^{-1} \bmod m$$

dimana *inverse* adalah dalam aritmatika modulo m. Kunci publik K_E adalah $(w_0, w_1, \ldots, w_{k-1})$ dimana

$$w_i = av_i \bmod m.$$

Kunci privat K_D adalah $(b, m, v_0, v_1, \ldots, v_{k-1})$.

Untuk mengenkripsi naskah $P = (p_0, p_1, \ldots, p_{k-1})$ dimana $p_i \in \{0, 1\}$, seseorang yang mengetahui kunci publik mengkomputasi

$$C = \sum_{i=0}^{k-1} p_i w_i.$$

Untuk mendekripsi C, seseorang yang mengetahui kunci privat melakukan komputasi

$$V = bC \bmod m.$$

Kita dapatkan

$$\begin{aligned} V &= bC \bmod m \\ &= (b \sum_{i=0}^{k-1} p_i w_i) \bmod m \\ &= (\sum_{i=0}^{k-1} p_i b_i w_i) \bmod m \\ &= (\sum_{i=0}^{k-1} p_i v_i) \bmod m \text{ (karena } b_i w_i \equiv v_i \pmod m) \\ &= \sum_{i=0}^{k-1} p_i v_i \text{ (karena } \sum_{i=0}^{k-1} p_i v_i < m). \end{aligned}$$

Karena $V = \sum_{i=0}^{k-1} p_i v_i$ maka $(p_0, p_1, \ldots, p_{k-1})$ dapat dicari secara efisien menggunakan algoritma untuk *super-increasing knapsack problem*.

Untuk yang hanya mengetahui kunci publik dan tidak mengetahui kunci privat, mencari $(p_0, p_1, \ldots, p_{k-1})$ dari $C = \sum_{i=0}^{k-1} p_i w_i$ adalah memecahkan secara umum *knapsack problem*, sesuatu yang sangat sukar. Itulah asumsi semula. Namun karena *knapsack problem* ini merupakan transformasi yang relatif sederhana dari suatu *super-increasing knapsack problem*, Adi Shamir berhasil

menemukan cara memecahkan Merkle-Hellman dalam *polynomial-time* (lihat [sha82]). Meskipun ada usaha memperbaiki Merkle-Hellman untuk mendapatkan sesuatu yang benar-benar *knapsack problem* umum, minat pada kriptografi *knapsack* surut setelah itu.

16.6 Zero-Knowledge Protocol

Zero-knowledge protocol agak berbeda dengan jenis-jenis kriptografi yang sudah dibahas. Meskipun semua jenis kriptografi mengandung unsur probabilitas, untuk *zero-knowledge protocol*, probabilitas berperan langsung dalam mekanismenya.

Secara garis besar, *zero-knowledge protocol* adalah mekanisme dimana seorang dengan rahasia tertentu, sebut saja Peggy, dapat meyakinkan pengujinya, sebut saja Victor, bahwa Peggy mengetahui rahasia itu, tanpa membuka rahasia tersebut kepada Victor atau orang lain. Tentu saja rahasia itu tidak bisa sembarang rahasia, tetapi rahasia yang mempunyai konsekuensi yang dapat diperiksa oleh Victor.

Contoh yang sering digunakan untuk menjelaskan konsep *zero-knowledge protocol* biasanya menggunakan gua atau terowongan yang bercabang. Versi disini, panjang terowongan sekitar 300m, dan 100m setelah pintu masuk, terowongan bercabang dua. Kedua cabang berjalan paralel dan pintu keluar keduanya berdekatan. Sekitar 5m dari pintu keluar terdapat suatu pintu yang menghubungkan kedua cabang yang hanya dapat dibuka oleh seseorang yang mengetahui suatu kode rahasia. Peggy ingin meyakinkan Victor bahwa ia (Peggy) mengetahui kode rahasia tersebut, tanpa memberi petunjuk kepada Victor apa kode rahasia tersebut. Victor sendiri tidak mengetahui kode rahasia tersebut. Peggy masuk kedalam terowongan lalu memilih satu cabang. Saat Peggy tiba di pintu antara kedua cabang, yaitu 5m sebelum pintu keluar, ia memberi kabar kepada Victor menggunakan telpon genggam. Victor kemudian menyatakan dari cabang mana Peggy harus keluar. Karena Peggy hanya diberi waktu 10 detik untuk keluar, jika ia berada di cabang yang salah maka tidak mungkin ia kembali ketempat dimana terowongan bercabang lalu masuk cabang yang benar. Ia harus membuka pintu dengan kode rahasia dan melewatinya. Sebelum 10 detik habis, Peggy keluar dari cabang yang diminta oleh Victor. Tentunya jika ini hanya dilakukan satu kali Victor belum tentu puas karena ada kemungkinan Peggy sebetulnya tidak mengetahui kode rahasia tetapi beruntung memilih cabang yang benar. Victor dapat meminta eksperimen ini diulang beberapa kali, setiap kali ia pastikan sebelumnya bahwa pintu antara cabang benar-benar tertutup, dan memilih secara acak lewat cabang mana Peggy harus keluar. Jika eksperimen ini diulang n kali dan Peggy selalu keluar dari cabang yang diminta, maka probabilitas bahwa Peggy tidak mengetahui kode rahasia adalah $(1/2)^n$. Contohnya, jika $n = 10$ probabilitas ini adalah

1/1024, jadi Victor dapat 99.9 persen yakin bahwa Peggy mengetahui kode rahasia, dan Victor sendiri tetap tidak mengetahui kode rahasia.

Konsep *zero-knowledge protocol* digunakan dalam beberapa protokol untuk identifikasi (*zero-knowledge identification protocol*). Protokol pertama jenis ini adalah protokol Fiat-Shamir. Kita akan bahas protokol Fiat-Shamir dan dua protokol yang merupakan derivatif dari Fiat-Shamir yaitu protokol Feige-Fiat-Shamir dan protokol Guillou-Quisquater. Ketiga protokol yang akan dibahas tergolong apa yang dinamakan *challenge-response protocol*.

Ada tiga aktor yang berperan dalam protokol Fiat-Shamir yaitu *trusted center* (sebut saja Tim), *prover* (Peggy) dan *verifier* (Victor). Tim membuat suatu modulus seperti RSA $n = pq$, mengumumkan n tetapi merahasiakan p dan q. Peggy membuat secara acak (menggunakan *random number generator*) kunci privat s, dimana $0 < s < n$ dan $\gcd(s, n) = 1$. Kunci publik Peggy adalah $v = s^2 \bmod n$ dan v diregistrasi ke Tim. Victor dapat memperoleh kunci publik Peggy v yang telah diregistrasi, dari Tim. Langkah-langkah berikut diulang t kali, setiap kali dengan nilai-nilai acak yang baru, agar Peggy dapat diidentifikasi oleh Victor.

1. Peggy memilih secara acak, menggunakan *random number generator*, r, $0 < r < n$, dan mengirim $x = r^2 \bmod n$ kepada Victor.

2. Victor memilih secara acak, menggunakan *random number generator*, e, $e \in \{0, 1\}$, dan mengirimnya ke Peggy.

3. Peggy mengkomputasi $y = rs^e \bmod n$ dan mengirim y ke Victor.

4. Jika $y = 0$ atau $y^2 \not\equiv xv^e \pmod n$ maka Victor menolak dan proses identifikasi gagal.

Jika langkah-langkah diatas telah diulang t kali tanpa penolakan maka identifikasi Peggy diterima oleh Victor. Probabilitas bahwa Peggy telah berhasil menipu Victor adalah 1 dalam 2^t.

Keamanan dari Fiat-Shamir berdasarkan pada sukarnya mengkalkulasi akar kuadrat modulo pq jika p dan q tidak diketahui (hanya produknya $n = pq$ yang diketahui). Jika p dan q diketahui, kita dapat menggunakan teknik diahir bagian 11.2 untuk mengkalkulasi akar kuadrat modulo pq, jadi p dan q harus dirahasiakan oleh Tim. Fungsi pengacakan menggunakan parameter e adalah agar Peggy tidak curang. Jika Victor selalu meminta $y = r$ ($e = 0$) maka jelas Peggy tidak perlu mengetahui s untuk menjawabnya. Jika Victor selalu meminta $y = rs \bmod n$, Peggy juga dapat mengelabui Victor tanpa mengetahui s sebagai berikut. Pada langkah 1 Peggy mengirim

$$x = r^2 v^{-1} \bmod n$$

kepada Victor. Ketika diminta untuk mengirim $y = rs \bmod n$ maka Peggy mengirim $y = r$. Jadi Victor terkelabui karena

$$\begin{aligned} xv &\equiv r^2 v^{-1} v \pmod{n} \\ &\equiv r^2 \pmod{n} \\ &\equiv y^2 \pmod{n}. \end{aligned}$$

Dengan pengacakan, jika Peggy mengirim $x = r^2 \bmod n$ maka ia kadang harus menggunakan s, sedangkan jika ia mengirim $x = r^2 v^{-1} \bmod n$ maka ia kadang harus mencari akar kuadrat modulo n karena ia harus mengirim

$$y = (r^2 v^{-1})^{-2} \bmod n.$$

Untuk menunjukkan bahwa tidak ada rahasia yang bocor ke Victor, kita gunakan cara standard yaitu dengan simulasi. Seseorang yang tidak mengetahui p, q dan s akan tetapi mengetahui apa yang akan diminta Victor untuk e tentunya akan dapat berperan sebagai Peggy dengan mengirim $x = r^2 \bmod n$ atau $x = r^2 v^{-1} \bmod n$ tergantung pada nilai e. Informasi yang dikeluarkan oleh Peggy dapat dikeluarkan oleh siapa saja tanpa mengetahui p, q dan s, jadi tidak memberi tahu nilai p, q dan s.

Sebetulnya Fiat-Shamir membocorkan 1 bit dari nilai s, yaitu *sign* ($+$ atau $-$) dari s. Protokol Feige-Fiat-Shamir menutup kebocoran ini. Selain itu Feige-Fiat-Shamir melakukan k "pembuktian" secara paralel yang mengurangi interaksi antara Peggy dan Victor karena langkah-langkah tidak perlu diulang sebanyak pada Fiat-Shamir, bahkan langkah-langkah tidak perlu diulang jika k cukup besar.

Dalam Feige-Fiat-Shamir, Peggy membuat k kunci privat $s_1, s_2, \ldots, s_k$ dimana $\gcd(s_i, n) = 1$ untuk setiap i, dan mempublikasikan k kunci publik $v_1, v_2, \ldots, v_k$ dimana

$$v_i = s_i^2 \bmod n$$

untuk setiap i. Langkah-langkah Feige-Fiat-Shamir adalah sebagai berikut.

1. Peggy memilih secara acak, menggunakan *random number generator*, r, $0 < r < n$, dan $s \in \{-1, 1\}$, dan mengirim $x = sr^2 \bmod n$ kepada Victor.

2. Victor memilih secara acak $e_1, e_2, \ldots, e_k$, $e_i \in \{0, 1\}$ untuk setiap i, dan mengirimnya ke Peggy.

3. Peggy mengkomputasi $y = rs_1^{e_1} s_2^{e_2} \cdots s_k^{e_k} \bmod n$ dan mengirim y ke Victor.

4. Jika $y^2 \not\equiv \pm x v_1^{e_1} v_2^{e_2} \cdots v_k^{e_k} \pmod{n}$ maka Victor menolak dan proses identifikasi gagal.

Jika $k = 20$, maka probabilitas bahwa Peggy berhasil mengelabui Victor kurang dari 1 dalam sejuta, dengan hanya 1 putaran langkah-langkah diatas.

Untuk Feige-Fiat-Shamir ada yang menggunakan $v_i = 1/s_i^2 \bmod n$ untuk kunci publik. Jika demikian, maka pada langkah 4, Victor harus memeriksa

$$x \equiv \pm y^2 v_1^{e_1} v_2^{e_2} \cdots v_k^{e_k} \pmod{n}.$$

Protokol Guillou-Quisquater kita bahas disini karena sangat populer untuk aplikasi identifikasi smartcard. Untuk Guillou-Quisquater, Peggy berperan sebagai smartcard, Victor adalah komputer yang sedang mengidentifikasi smartcard, dan Tim adalah pengelola smartcard. Seperti Fiat-Shamir, $n = pq$ dipilih oleh Tim, n dipublikasikan, sedangkan p dan q dirahasiakan. Selain n, parameter v yang merupakan eksponen dengan $\gcd(v, \phi(n)) = 1$ juga dipublikasikan. Untuk Peggy, terdapat parameter B yang merupakan bilangan yang merepresentasikan *credentials* dari smartcard (terdiri misalnya dari *card issuer*, *serial number* dan *expiry date*). Peggy juga diberi Tim kunci privat K, dimana

$$B \cdot K^v \equiv 1 \pmod{n}.$$

Peggy mengirim *credentials* miliknya (yang direpresentasikan menggunakan B) kepada Victor. Langkah-langkah *challenge-response* kemudian berlangsung mirip dengan Fiat-Shamir dan Feige-Fiat-Shamir:

1. Peggy memilih secara acak, menggunakan *random number generator*, r, $0 < r < n$, dan mengirim $T = r^v \bmod n$ kepada Victor.

2. Victor memilih secara acak, menggunakan *random number generator*, d, dimana $0 \leq d < v$, dan mengirimnya ke Peggy.

3. Peggy mengkomputasi $D = rK^d \bmod n$ dan mengirim D ke Victor.

4. Victor memeriksa $D^v B^d \equiv T \pmod{n}$. Jika tidak cocok maka Victor menolak dan proses identifikasi gagal.

Pada langkah 4, Victor memeriksa $D^v B^d \equiv T \pmod{n}$ karena menggunakan fakta bahwa $B \cdot K^v \equiv 1 \pmod{n}$ kita dapatkan:

$$\begin{aligned} D^v B^d &\equiv (rK^d)^v B^d \pmod{n} \\ &\equiv r^v K^{dv} B^d \pmod{n} \\ &\equiv r^v (BK^v)^d \pmod{n} \\ &\equiv r^v \pmod{n} \\ &\equiv T \pmod{n}. \end{aligned}$$

Karena komunikasi antara Peggy dan Victor tidak diamankan, ketiga protokol identifikasi yang telah dibahas mempunyai kelemahan yaitu rentan terhadap *man-in-middle attack*. Jadi dalam aplikasinya, peluang untuk seseorang menempatkan dirinya diantara Peggy dan Victor dalam media komunikasi

harus dihilangkan atau dibuat sekecil mungkin. Tentunya jika komunikasi diamankan menggunakan, misalnya RSA, maka tidak ada kelemahan ini. Namun jika sudah menggunakan RSA maka protokol identifikasi seperti diatas tidak diperlukan.

16.7 Penggunaan Kriptografi Public Key

Hampir semua sistem kriptografi *public key* yang telah dibahas dirancang untuk keperluan khusus. Diffie-Hellman dirancang untuk keperluan *key agreement*, DSA untuk *digital signature*, dan sistem yang berbasis pada *zero-knowledge protocol* untuk identifikasi. RSA dan ElGamal adalah dua sistem kriptografi *public key* yang dapat digunakan untuk enkripsi. Namun jika dibandingkan dengan sistem enkripsi klasik seperti AES, 3DES dan CAST, maka RSA dan ElGamal sangat lambat. Jadi penggunaan RSA dan ElGamal dalam enkripsi juga agak khusus yaitu mengenkripsi kunci enkripsi klasik atau mengenkripsi *digest* untuk keperluan *digital signature*. Enkripsi kunci enkripsi klasik contohnya dalam suatu sesi SSL atau SSH (akan dibahas di bab 20), kunci sesi yang biasanya merupakan kunci AES, 3DES atau CAST, dienkripsi menggunakan kunci publik jenis RSA atau ElGamal. Demikian juga jika mengenkripsi *file* yang ukurannya bisa cukup besar, kriptografi klasik digunakan untuk mengenkripsi *file*, lalu kunci klasik dienkripsi menggunakan kunci publik.

Kriptografi *public key* tentunya juga berperan sangat besar dalam suatu *public key infrastructure* (akan dibahas di bab 23). Fungsi utama *public key infrastructure* adalah manajemen *digital signature* dan kunci publik untuk dekripsi, termasuk manajemen *certificate*, untuk kunci publik *digital signature* dan untuk kunci publik enkripsi. Aplikasi untuk kunci publik yang dikelolapun beragam, dari *certificate* untuk *website*, kunci publik untuk keperluan *secure email*, sampai dengan *certificate* untuk keperluan IPsec (akan dibahas di bagian 20.3).

16.8 Ringkasan

Berbagai sistem kriptografi *public key* telah dibahas di bab ini, yaitu RSA, Diffie-Hellman, DSA, ElGamal, *knapsack*, Fiat-Shamir, Feige-Fiat-Shamir dan Guillou-Quisquater. Kecuali *knapsack*, mekanisme transformasi untuk semua sistem yang telah dibahas berandalkan pada rumus

$$a^{\phi(q)} \equiv 1 \pmod{q}$$

dimana $a \in \mathbf{N}$ dan $\gcd(a, q) = 1$, atau

$$a^{q-1} = 1$$

dimana $a \in \mathbf{GF}(q)^*$. Keamanan dari RSA, Fiat-Shamir, Feige-Fiat-Shamir dan Guillou-Quisquater berbasis pada sukarnya menguraikan q, sedangkan keamanan dari Diffie-Hellman, DSA dan ElGamal berbasis pada sukarnya mengkomputasi logaritma diskrit. Sistem kriptografi yang berbasis pada *knapsack problem* tidak populer karena hilangnya kepercayaan pada keamanannya sejak sistem pertama yang berbasis pada *knapsack problem*, yaitu Merkle-Hellman, dapat dipecahkan.

Dibandingkan kriptografi klasik (simetris), kriptografi *public key* sangat tidak efisien untuk enkripsi umum. Namun banyak keperluan enkripsi khusus yang dapat dipenuhi oleh kriptografi *public key* dan tidak dapat dipenuhi oleh kriptografi klasik. Oleh sebab itu kriptografi *public key* digunakan untuk berbagai keperluan khusus seperti *key agreement*, *key distribution*, *digital signature* dan *identification protocol*.

Bab 17

Kriptografi Elliptic Curve

Sistem kriptografi *public key* yang berbasis pada sukarnya mengkomputasi logaritma diskrit seperti Diffie-Hellman, DSA dan ElGamal bekerja menggunakan suatu *multiplicative group* $\mathbf{GF}(q)^*$. Suatu *elliptic curve over a finite field* juga memberikan *Abelian group* yang dapat digunakan untuk mekanisme kriptografi yang serupa dengan sistem berbasis logaritma diskrit. Lebih menarik lagi, *elliptic curve over a finite field* memberikan lebih banyak fleksibilitas dibandingkan *finite field* yang terbatas pada $\mathbf{GF}(p)$ dan $\mathbf{GF}(p^n)$.

Sebelum membahas penggunaan *elliptic curve* untuk kriptografi, tentunya kita perlu mengetahui apa itu *elliptic curve*, khususnya *elliptic curve over a finite field*. Suatu *elliptic curve* bukan merupakan suatu elips, tetapi merupakan suatu "kurva" untuk persamaan sebagai berikut:

$$Ax^3 + Bx^2y + Cxy^2 + Dy^3 + Ex^2 + Fxy + Gx + Iy = 0.$$

Field apa saja dapat digunakan untuk membuat *elliptic curve*, termasuk $\mathbf{R}$, $\mathbf{Q}$ (masing-masing mempunyai *characteristic* 0), dan *finite field* (yang tentunya mempunyai *characteristic* suatu bilangan prima). Jika *field* yang digunakan mempunyai *characteristic* k dimana $k \neq 2$ dan $k \neq 3$, maka persamaan diatas dapat disederhanakan menjadi:

$$y^2 = x^3 + ax + b. \tag{17.1}$$

Jika $k = 2$ maka persamaan hanya dapat disederhanakan menjadi

$$y^2 + cy = x^3 + ax + b \tag{17.2}$$

atau

$$y^2 + xy = x^3 + ax + b, \tag{17.3}$$

sedangkan jika $k = 3$ persamaan hanya dapat disederhanakan menjadi

$$y^2 = x^3 ax^2 + bx + c. \tag{17.4}$$

Suatu *elliptic curve over field* F adalah titik-titik (a, b) yang merupakan solusi untuk x dan y dalam persamaan, dimana $a, b \in F$, ditambah dengan titik di ∞ (*point at infinity*) yang diberi simbol 0. Suatu *Abelian group* dapat dibentuk menggunakan titik-titik tersebut dan operasi $+$. Untuk $F = \mathbf{R}$ (jadi $k = 0$), kita definisikan $+$ dan $-$ (*inverse*) sebagai berikut:

1. $-0 = 0$ dan $0 + Q = Q$ (jadi 0 merupakan *identity*).

2. Untuk $P \neq 0$, $-P$ adalah titik dengan koordinat x yang sama dan negatif koordinat y. Jadi $-(x, y) = (x, -y)$.

3. Jika P dan Q adalah dua titik yang berbeda, maka tidak terlalu sulit untuk melihat bahwa garis $l = \overline{PQ}$ melewati titik ketiga R. Kita definisikan $P + Q = -R$ (jadi $P + Q + R = 0$).

4. $P + (-P) = 0$.

5. Yang terahir adalah untuk $P + P$. Jika l adalah garis tangen untuk P, maka l akan bertemu dengan satu titik lagi yaitu R, kecuali jika P merupakan titik infleksi. Kita definisikan $P + P = -R$, kecuali jika P adalah titik infleksi dimana kita definisikan $P + P = -P$.

Untuk bagian 3, kita bisa dapatkan rumus yang lebih rinci lagi berdasarkan persamaan 17.1. Kita gunakan koordinat (x_P, y_P) untuk P, (x_Q, y_Q) untuk Q dan (x_R, y_R) untuk R. Jika

$$y = \alpha x + \beta$$

adalah rumus untuk garis l yang melalui P dan Q, maka

$$\alpha = (y_Q - y_P)/(x_Q - x_P)$$

dan

$$\beta = y_P - \alpha x_P.$$

Suatu titik pertemuan antara garis l dan kurva akan mematuhi persamaan

$$(\alpha x + \beta)^2 = x^3 + ax + b$$

yang, dalam bentuk *polynomial* menjadi

$$x^3 - \alpha^2 x^2 + (a - 2\beta)x + (b - \beta^2) = 0.$$

Tentunya P, Q dan R merupakan akar dari persamaan ini. Karena penjumlahan akar merupakan negatif koefisien x^2 dalam persamaan (lihat pembahasan *trace* di bagian 12.3), maka $x_R = \alpha^2 - x_P - x_Q$, jadi

$$\begin{aligned} x_R &= \left(\frac{y_Q - y_P}{x_Q - x_P}\right)^2 - x_P - x_Q, \\ y_R &= -y_P + \left(\frac{y_Q - y_P}{x_Q - x_P}\right)(x_P - x_R). \end{aligned} \tag{17.5}$$

Untuk bagian 5, kita juga bisa dapatkan rumus yang rinci, tetapi α merupakan derivatif dy/dx di P. Diferensiasi implisit persamaan 17.1 menghasilkan $\alpha = (3x_P^2 + a)/2y_P$, jadi

$$\begin{aligned} x_R &= \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P, \\ y_R &= -y_P + \left(\frac{3x_P^2 + a}{2y_P}\right)(x_P - x_R). \end{aligned} \tag{17.6}$$

Rumus 17.5 dan 17.6 dapat digunakan untuk menunjukkan bahwa definisi diatas membentuk suatu *Abelian group*.

Rumus 17.5 dan 17.6 juga berlaku untuk *elliptical curve over a finite field* jika *finite field* mempunyai *characteristic* $k > 3$. Jika $k = 2$, maka rumus 17.5 dan 17.6 harus disesuaikan dengan persamaan 17.2 atau 17.3, sedangkan untuk $k = 3$, rumus 17.5 dan 17.6 harus disesuaikan dengan persamaan 17.4.

Tentunya jumlah titik-titik pada *elliptical curve* dengan *finite field* $\mathbf{GF}(q)$, n, tidak melebihi $2q + 1$, karena untuk setiap $x \in \mathbf{GF}(q)$ (ada q nilai x yang berbeda) terdapat maksimal dua nilai untuk $y \in \mathbf{GF}(q)$ dimana (x, y) merupakan titik pada kurva, ditambah 1 titik di ∞. Akan tetapi, karena hanya setengah dari elemen $\mathbf{GF}(q)^*$ mempunyai akar kuadrat, maka jumlah titik-titik pada kurva kira-kira hanya setengah dari itu. Untuk tepatnya, kita gunakan fungsi *quadratic character* χ. Fungsi χ memetakan setiap $x \in \mathbf{GF}(q)^*$ ke ± 1 tergantung apakah x merupakan kuadrat dalam $\mathbf{GF}(q)$ (1 jika kuadrat dan -1 jika bukan kuadrat). Jika $q = p$, suatu bilangan prima, tentunya $\chi(x) = \left(\frac{x}{p}\right)$ (lihat bagian 11.1). Untuk setiap $x \in \mathbf{GF}(q)$, banyaknya solusi y yang menjadikan (x, y) suatu titik pada kurva (jika $k > 3$) adalah

$$1 + \chi(x^3 + ax + b)$$

yang menghasilkan 0 jika $\chi(x^3 + ax + b) = -1$ dan 2 jika $\chi(x^3 + ax + b) = 1$. Jadi banyaknya titik, termasuk titik di ∞ adalah

$$1 + \sum_{x \in \mathbf{GF}(q)} (1 + \chi(x^3 + ax + b)) = 1 + q + \sum_{x \in \mathbf{GF}(q)} \chi(x^3 + ax + b).$$

Karena peluang $\chi(x^3 + ax + b)$ untuk menghasilkan 1 sama dengan peluangnya untuk menghasilkan -1, maka penjumlahan diatas ibarat melakukan *random*

walk dan dibatasi oleh $2\sqrt{q}$. Inilah yang menjadi dasar dari teorema Hasse sebagai berikut:

Teorema 112 (Hasse's Theorem) *Jika n adalah banyaknya titik-titik pada elliptic curve yang didefinisikan menggunakan* $\mathbf{GF}(q)$*, maka*

$$|n - (q + 1)| \leq 2\sqrt{q}.$$

Jika ingin mengetahui dengan tepat banyaknya titik-titik pada *elliptic curve* yang didefinisikan menggunakan $\mathbf{GF}(q)$, ada algoritma yang pertama ditemukan oleh Schoof (lihat [sch85]) dan dikembangkan lebih lanjut oleh peneliti lainnya. Nilai n ini perlu diketahui untuk mengetahui keamanan enkripsi, karena jika n dapat diuraikan menjadi produk dari bilangan-bilangan prima yang kecil, maka metode Pohlig-Silver-Hellman dapat digunakan untuk mengkomputasi logaritma diskrit.

Sistem kriptografi *public key* yang berdasarkan pada logaritma diskrit mempunyai versi yang menggunakan *elliptic curve*. Disini kita akan bahas dua diantaranya yaitu Diffie-Hellman dan ElGamal. Untuk yang ingin mempelajari versi *elliptic curve* dari DSA disarankan untuk membaca [nis00].

Untuk Diffie-Hellman, suatu *finite field* $\mathbf{GF}(q)$ digunakan untuk membuat suatu *elliptic curve*, E. Jika dalam versi asli suatu g, yang sebaiknya merupakan *generator*, dipilih sebagai basis, maka di versi *elliptic curve*, dipilih suatu titik $B \in E$. Alice dan Bob melakukan *key agreement* menggunakan Diffie-Hellman versi *elliptic curve* sebagai berikut:

- Alice memilih, secara acak, suatu bilangan a yang besarnya sekitar q, mengkomputasi aB, dan mengirimkan aB ke Bob.
- Bob memilih, secara acak, suatu bilangan b yang besarnya sekitar q, mengkomputasi bB, dan mengirimkan bB ke Alice.
- Setelah menerima bB dari Bob, Alice mengkomputasi abB yang menjadi kunci bersama dengan Bob.
- Bob, setelah menerima aB dari Alice, mengkomputasi $abB = baB$.

Proses *key agreement* menghasilkan kunci bersama abB antara Alice dan Bob. Seseorang yang tidak mengetahui a dan tidak mengetahui b tidak dapat mengkomputasi abB dari aB dan bB, tanpa menemukan cara efisien untuk mengkomputasi logaritma diskrit (asumsi Diffie-Hellman). Tabel 17.1 membandingkan Diffie-Hellman versi *finite field* (DH) dengan Diffie-Hellman versi *elliptic curve* (ECDH).

Serupa dengan Diffie-Hellman, untuk ElGamal, suatu *finite field* $\mathbf{GF}(q)$ digunakan untuk membuat suatu *elliptic curve*, E. Suatu titik $B \in E$ digunakan sebagai basis. Untuk membuat pasangan kunci, Alice memilih secara acak suatu bilangan a yang ia jadikan kunci privat. Alice kemudian mengkomputasi

Komponen	DH	ECDH
Basis	$g \in \mathbf{GF}(q)$	$B \in E$
Potongan kunci A	g^a	aB
Potongan kunci B	g^b	bB
Kunci bersama	g^{ab}	abB

Tabel 17.1: Perbedaan Diffie-Hellman dengan versi *elliptic curve*

aB dan mempublikasikannya sebagai kunci publiknya. Untuk mengenkripsi suatu naskah yang telah dikodifikasi sebagai M dalam E, agar hanya dapat dibaca oleh Alice, seseorang melakukan langkah-langkah berikut:

- Pilih bilangan bulat k secara acak.
- Kirim pasangan $(kB, M + k(aB))$ ke Alice.

Alice dapat mendekripsi $(kB, M + k(aB))$ dengan, pertama, mengkomputasi $a(kB)$, lalu mengkomputasi

$$M + k(aB) - a(kB) = M.$$

Seseorang yang tidak mengetahui k atau kunci privat a tentunya tidak dapat mengkomputasi $k(aB)$ tanpa menemukan cara efisien untuk mengkomputasi logaritma diskrit. Tabel 17.2 membandingkan ElGamal versi *finite field* (EG) dengan ElGamal versi *elliptic curve* (ECEG).

Komponen	EG	ECEG
Basis	$g \in \mathbf{GF}(q)$	$B \in E$
Kunci privat	a	a
Kunci publik	g^a	aB
IV	k	k
Naskah	M	M
Enkripsi	(g^k, Mg^{ak})	$(kB, M + k(aB))$

Tabel 17.2: Perbedaan ElGamal dengan versi *elliptic curve*

Ada dua hal mengenai implementasi *elliptic curve* suatu sistem kriptografi berbasis logaritma diskrit yang akan kita bahas disini:

- Pemetaan antara bilangan-bilangan yang merepresentasikan naskah dengan titik-titik dalam *elliptic curve*.
- Operasi perkalian dengan bilangan bulat dalam *elliptic curve*.

Sayangnya belum ada algoritma *polynomial time* yang dapat secara deterministik memetakan bilangan-bilangan menjadi titik-titik dalam *elliptic curve*. Akan tetapi terdapat algoritma probabilisitik yang cukup efisien dan kemungkinan untuk gagal dapat dibuat sangat kecil. Kita beri contoh untuk $q = p^r$ sangat besar dan ganjil. Kita pilih suatu bilangan bulat κ berdasarkan seberapa kecil kita ingin probabilitas kegagalan, yaitu

$$1/2^\kappa.$$

Biasanya nilai κ antara 30 sampai dengan 50. Kita harus membatasi suatu unit naskah sehingga mempunyai nilai bilangan bulat m dimana $0 \leq m < M$ dan $M\kappa < q$. Kita dapat menulis setiap bilangan bulat dari 1 sampai dengan $M\kappa$ dalam bentuk

$$m\kappa + j$$

dimana $1 \leq j \leq \kappa$. Maka terdapat suatu *injection* dari bilangan-bilangan tersebut ke $\mathbf{GF}(q)$ sebagai berikut:

- Representasikan setiap bilangan sebagai bilangan dengan basis p, jadi maksimal terdapat r digit.
- Bilangan dengan basis p tersebut dapat diinterpretasikan sebagai elemen dari $\mathbf{GF}(q)$.

Setiap m kita petakan ke elemen dari E sebagai berikut:

1. $j \leftarrow 1$.
2. Dapatkan $x \in \mathbf{GF}(q)$ sebagai nilai $m\kappa + j$ berdasarkan *injection* diatas.
3. Komputasi $f(x)$ sebagai sisi kanan dari persamaan $y^2 = x^3 + ax + b$.
4. Coba cari akar kuadrat dari $f(x)$ menggunakan metode di ahir bagian 11.2.
5. Jika akar kuadrat $f(x)$ ditemukan, maka kita gunakan akar tersebut sebagai nilai y, dan kita selesai dengan memetakan m ke (x, y).
6. Jika belum berhasil maka $j \leftarrow j + 1$ dan kembali ke langkah 2.

Algoritma diatas akan gagal jika kita tidak menemukan (x, y) sebelum j melebihi κ, dan probabilitas kegagalan adalah $1/2^\kappa$.

Jika dalam aritmatika $\mathbf{GF}(q)$ untuk kriptografi operasi pemangkatan elemen dengan bilangan bulat diperlukan, maka dalam kriptografi *elliptic curve* operasi perkalian elemen dengan bilangan bulat diperlukan. Jika pemangkatan dalam aritmatika $\mathbf{GF}(q)$ dapat dilakukan secara efisien menggunakan teknik *repeated squaring* (pengkuadratan berulang), maka dalam aritmatika *elliptic*

curve perkalian elemen dengan bilangan bulat dapat dilakukan secara efisien menggunakan *repeated doubling* (penggandaan berulang). Sebagai contoh, $100P$ dapat dikomputasi secara efisien sebagai

$$100P = 2(2(P + 2(2(2(P + 2P))))).$$

Ini dapat dilakukan menggunakan langkah-langkah berikut secara berulang, dimana $M > 0$ adalah pengali:

1. Jika $M = 1$ kita selesai dengan hasil P.

2. Jika M genap maka $M \leftarrow M/2$, kita cari hasil dengan M yang baru, lalu kalikan 2 ke hasil tersebut.

3. Jika M ganjil maka $M \leftarrow M - 1$, kita cari hasil dengan M yang baru, lalu tambahkan P ke hasil tersebut.

Kita ahiri bab ini dengan pembahasan secara ringkas mengapa kriptografi *public key* menggunakan *elliptic curve* diminati. Logaritma diskrit untuk *elliptic curve* jauh lebih sukar dibandingkan logaritma diskrit untuk $\mathbf{GF}(q)$ (kecuali untuk *supersingular elliptic curve* yang sangat jarang). Untuk *elliptic curve*, secara umum logaritma diskrit hanya dapat dikomputasi menggunakan algoritma Shanks atau algoritma Pollard, dan kedua algoritma mempunyai kompleksitas *full exponential*. Untuk *finite field*, kompleksitas logaritma diskrit mirip dengan kompleksitas penguraian yaitu *sub-exponential*. Jadi kompleksitas untuk *elliptic curve* tumbuh lebih cepat dibandingkan kompleksitas untuk *finite field*. Berdasarkan perkiraan oleh Alfred Menezes (lihat [men95]), penggunaan kriptografi *finite field* seperti DSA atau RSA dengan kunci sebesar 1024 bit sama kuatnya dengan penggunaan kriptografi *elliptic curve* dengan kunci sebesar 160 bit. Jadi cukup menyolok perbedaan besar kunci untuk kekuatan yang sama. Untuk kekuatan yang lebih besar, perbedaan semakin menyolok karena kekuatan *elliptic curve* tumbuh lebih cepat (*exponential* dibandingkan *sub-exponential* untuk *finite field*), seperti terlihat pada tabel 17.3 (data berdasarkan [rob97]). ECDSA adalah DSA versi *elliptic curve* sedang-

DSA/ElGamal	**RSA**	**ECDSA/ECES**
1024	1024	160
2048	2048	224
3072	3072	256
7680	7680	384
15360	15360	512

Tabel 17.3: Besar kunci untuk kekuatan yang sama

kan ECES adalah ElGamal versi *elliptic curve*. Tabel 17.4 menunjukkan relatif waktu yang dibutuhkan untuk berbagai operasi (data berdasarkan [rob97]). Untuk *key generation*, RSA jauh lebih lambat dibandingkan DSA/ElGamal

	DSA/ElGamal	**RSA**	**ECDSA/ECES**
	1024 bit	1024 bit	160 bit
encryption	480	17	120
decryption	240	384	60
signing	240	384	60
verification	480	17	120

Tabel 17.4: Waktu untuk berbagai operasi

dan ECDSA/ECES. Jadi cukup jelas mengapa sistem kriptografi *public key* dengan *elliptic curve* sangat menarik, terutama untuk aplikasi di perangkat kecil dengan kemampuan terbatas seperti *smartcard* atau perangkat *Bluetooth*. Kriptografi *elliptic curve* juga semakin menarik untuk penggunaan masa depan karena pertumbuhan kunci yang dibutuhkan tidak sebesar kriptografi *finite field*. Sebetulnya versi *elliptic curve* untuk RSA juga ada, namun berbeda dengan logaritma diskrit dimana versi *elliptic curve* lebih sukar untuk dipecahkan dibandingkan versi *finite field*, penguraian bilangan bulat tetap merupakan penguraian bilangan bulat, jadi tidak ada keuntungan dengan menggunakan *elliptic curve* untuk RSA.

17.1 Ringkasan

Di bab ini telah didiskusikan secara garis besar konsep *elliptic curve*, bagaimana sistem kriptografi yang berbasis pada logaritma diskrit seperti Diffie-Hellman, DSA dan ElGamal dapat diadaptasi untuk menggunakan *elliptic curve*, dan beberapa masalah implementasi kriptografi *elliptic curve*. Pembahasan ringkas mengenai mengapa kriptografi *elliptic curve* diminati terdapat pada ahir bab ini, termasuk mengapa versi *elliptic curve* untuk RSA tidak diminati.

Bab 18

Quantum Key Distribution

Quantum key distribution adalah metode untuk *key agreement* yang didasarkan pada fisika kuantum. Metode ini bukan suatu sistem kriptografi yang menggunakan *quantum computer*, melainkan suatu sistem yang didasarkan pada kenyataan bahwa jika pengukuran dilakukan terhadap suatu partikel yang terisolir (contohnya foton), maka pengukuran tersebut mempengaruhi kelakuan dari partikel tersebut. Fisika kuantum tidak akan dibahas panjang lebar disini, hanya sifat dasar polarisasi foton saja dan asumsi bahwa suatu *quantum state* tidak dapat di*clone*. Kita juga tidak akan bahas *no cloning theorem*. Pembahasan disini lebih pada logika dari protokol untuk *quantum key distribution* (dengan asumsi *quantum state* tidak dapat di*clone*), jadi bukan pada aspek fisika kuantum dari *quantum key distribution*. Bahkan kita akan lebih fokus lagi pada jenis protokol berbasis pengukuran, bukan pada protokol berbasis *quantum entanglement*. Sebagai contoh akan kita bahas protokol Bennett-Brassard. Penggunaan *Heisenberg uncertainty principle* di buku ini adalah sebagai limitasi teknik pengukuran, bukan sebagai suatu limitasi dari apa yang dapat kita ketahui, jadi bukan penggunaan yang kontroversial.

Sinar dapat dipolarisir misalnya dengan filter Polaroid atau dengan kristal *calcite*. Foton yang terpolarisir bisa diambil dari sinar yang telah dipolarisir dengan cara yang serupa dengan yang dilakukan dalam eksperimen oleh Aspect, Grangier dan Roger (lihat [asp82]). Karena *Heisenberg uncertainty principle*, pengukuran hanya dapat memberikan informasi 1 bit mengenai polarisasi foton. Jika sinar dipolarisir dengan orientasi α dan ditujukan ke filter B yang mempunyai orientasi β, maka setiap foton dalam sinar berperilaku dikotomis dan probabilistik saat bertemu filter B yaitu:

- foton ditransmisi oleh B dengan probabilitas $\cos^2(\alpha - \beta)$, atau
- foton diabsorbsi oleh B dengan probabilitas $\sin^2(\alpha - \beta)$.

Jadi foton bersifat deterministik hanya jika α dan β paralel (foton dipastikan ditransmisi oleh B) atau α dan β tegak lurus (foton dipastikan diabsorbsi oleh B). Jika α dan β tidak tegak lurus, tidak ada informasi tambahan yang bisa didapatkan jika foton ditransmisi karena foton ditransmisi dengan polarisasi β, jadi orientasi α sudah "dilupakan" oleh foton.

Dalam ilmu fisika kuantum, konsep probabilitas yang digunakan adalah *probability amplitude*, yang harus dikuadratkan untuk mendapatkan probabilitas. Itulah sebabnya probabilitas untuk foton ditransmisi adalah $\cos^2(\alpha - \beta)$, dimana $\cos(\alpha-\beta)$ adalah *probability amplitude*. *Internal state* dari sistem kuantum (contohnya polarisasi foton) direpresentasikan menggunakan vektor yang panjangnya 1 dalam ruang linear menggunakan *complex field* (ruang Hilbert). Untuk polarisasi foton, ruang Hilbert yang digunakan adalah ruang Hilbert dua dimensi, jadi *state* dari foton dapat direpresentasikan menggunakan kombinasi linear dari dua vektor yang *linearly independent*, contohnya

$$r_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad r_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

dimana r_1 merepresentasikan polarisasi horisontal dan r_2 merepresentasikan polarisasi vertikal. Jadi suatu foton yang terpolarisir dengan orientasi α direpresentasikan oleh vektor $(\cos\alpha, \sin\alpha)$. Jika foton tersebut diukur polarisasi horisontal dan vertikalnya, maka foton tersebut "memilih" untuk menjadi horisontal dengan probabilitas $\cos^2\alpha$, atau menjadi vertikal dengan probabilitas $\sin^2\alpha$. Basis $\{r_1, r_2\}$ disebut basis *rectilinear*. Sebagai alternatif, kita dapat menggunakan

$$d_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \qquad d_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix},$$

dimana d_1 merepresentasikan polarisasi 45 derajat dan d_2 merepresentasikan polarisasi 135 derajat. Basis $\{d_1, d_2\}$ disebut basis *diagonal*. Dua basis (contohnya *rectilinear* dan *diagonal*) disebut *conjugate* jika setiap vektor dalam satu basis mempunyai proyeksi yang sama panjang ke dua vektor dalam basis pasangannya. Sebagai contoh, d_1 jika diproyeksikan ke r_1 dan r_2 menjadi

$$d_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} r_1 + \frac{1}{\sqrt{2}} r_2,$$

dimana $\frac{1}{\sqrt{2}}$ merupakan *probability amplitude* d_1 akan memilih r_1 atau r_2 jika diukur menggunakan basis *rectilinear*, jadi probabilitas untuk memilih r_1 atau r_2 sama yaitu $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$. Ini berarti suatu foton yang berada pada *state* tertentu dalam suatu basis (d_1 atau d_2 dalam basis *diagonal*, r_1 atau r_2 dalam basis *rectilinear*) akan berperilaku acak (semua informasi hilang) jika diukur

menggunakan basis *conjugate*. Sebetulnya masih ada satu lagi basis yang *conjugate* dengan basis *rectilinear* dan basis *diagonal* dalam ruang Hilbert dua dimensi yaitu basis *circular*, akan tetapi basis tersebut tidak diperlukan disini.

Secara garis besar, dalam *quantum key distribution*, saluran kuantum tidak digunakan untuk transmisi naskah, baik naskah asli maupun naskah yang telah dienkripsi, melainkan digunakan untuk transmisi bit secara acak antara dua pengguna yang pada awalnya tidak memiliki rahasia bersama. Dengan probabilitas yang sangat tinggi, keduanya dapat mendeteksi apabila ada pihak ketiga yang telah menyadap transmisi bit melalui saluran kuantum, karena pihak ketiga harus melakukan pengukuran yang menyebabkan transmisi kemungkinan besar terganggu (karena pengukuran bisa merubah polarisasi foton). Jika kedua pengguna berpendapat bahwa transmisi tidak terganggu, maka deretan bit yang telah ditransmisi dapat digunakan untuk membuat kunci rahasia bersama. Jika tidak, maka hasil tranmisi dibuang, dan transmisi dicoba lagi (dengan urutan bit acak yang lain).

Secara lebih rinci, pengguna pertama (sebut saja Alice) membuat deretan bit acak dan deretan basis acak. Untuk setiap bit, Alice mentransmisi bit tersebut kepada pengguna kedua (sebut saja Bob) melalui saluran kuantum, menggunakan basis untuk bit tersebut (yang telah dibuat secara acak *rectilinear* atau *diagonal*). Bob "membaca" transmisi dari Alice dengan mengukur deretan bit menggunakan deretan basis yang ia buat acak, independen dari deretan basis yang digunakan Alice. Secara rerata, Bob akan menggunakan basis yang benar untuk setengah dari semua bit yang ditransmisi. Alice dan Bob mencocokkan basis yang mereka pergunakan, dengan menggunakan jalur umum untuk komunikasi. Dari semua pengukuran yang cocok, Bob memilih kira-kira sepertiga bit secara acak, lalu Alice dan Bob mencocokkan bit-bit yang terpilih melalui jalur umum. Jika semua bit cocok, maka Alice dan Bob dapat cukup yakin bahwa transmisi telah berlangsung secara benar tanpa penyadapan. Alice dan Bob dapat mengambil sisa dua pertiga dari bit yang diukur secara benar, dan menggunakannya untuk membuat kunci rahasia bersama. Tentunya banyaknya bit yang digunakan tergantung pada apa yang diperlukan untuk membuat kunci rahasia bersama. Jika kunci rahasia merupakan *one-time pad* (biasanya sistem *quantum key distribution* digunakan untuk membuat *one-time pad*), maka diperlukan deretan bit yang panjang. Gambar 18.1 memperlihatkan contoh suatu sesi protokol Bennett-Brassard dimana hasil yang dapat digunakan untuk membuat kunci rahasia bersama adalah $1, 0, 1, 1$.

Jika ada pihak ketiga yang menyadap transmisi melalui jalur kuantum, maka besar kemungkinan ada ketidak-cocokan dalam sepertiga bit yang dicocokkan, kecuali jika jumlah bit yang disadap tidak terlalu banyak. Komunikasi melalui jalur publik juga harus diamankan dari penyusupan. Ini dapat dilakukan misalnya menggunakan *tag* Wegman-Carter (lihat [weg81]) untuk *secure hashing* (lihat bab 9). Menggunakan *secure hashing* sebagai *message*

authentication code merupakan sesuatu yang cukup umum untuk mekanisme *authentication*.

Meskipun beberapa produk komersial *quantum key distribution* telah dipasarkan, keamanan dari *quantum key distribution* masih dipertanyakan. Sebagai contoh, Jörgen Cederlöf dan Jan-Åke Larsson membahas kelemahan *tagging* Wegman-Carter sebagai mekanisme *authentication* (lihat [ced08]). Kelemahan pada *detector module* yang digunakan dalam produk komersial *quantum key distribution* juga telah dipublikasi (lihat [mak09]). Dengan menimbang berbagai kelemahan yang telah dipublikasi, dan kenyataan bahwa *quantum key distribution* belum merupakan *proven technology*, maka buku ini belum bisa merekomendasikan penggunaan *quantum key distribution*.

18.1 Ringkasan

Di bab ini telah dibahas secara ringkas cara melakukan *key agreement* menggunakan *quantum key distribution*. Protokol yang telah dibahas adalah protokol Bennett-Brassard. Pembahasan fisika kuantum hanya sebatas sifat dasar polarisasi, sekedar cukup untuk menjelaskan protokol Bennet-Brassard. Karena beberapa kelemahan *quantum key distribution* dan kenyataan bahwa *quantum key distribution* belum merupakan *proven technology*, penggunaan *quantum key distribution* belum bisa direkomendasikan.

Jika *quantum key distribution* kelak menjadi sesuatu yang praktis, maka enkripsi *one-time pad* juga akan menjadi praktis, meskipun aplikasinya terbatas karena sifat *quantum key distribution* yang *point-to-point* secara fisik.

TRANSMISI JALUR KUANTUM															
A	0	1	1	0	1	1	0	0	1	0	1	1	0	0	1
A	D	R	D	R	R	R	R	R	D	D	R	D	D	D	R
A	↗	↑	↘	→	↑	↑	→	→	↘	↗	↑	↘	↗	↗	↑
B	R	D	D	R	R	D	D	R	D	R	D	D	D	D	R
B	1		1		1	0	0	0		1	1	1		0	1
TRANSMISI JALUR PUBLIK															
B	R		D		R	D	D	R		R	D	D		D	R
A			√		√			√				√		√	√
√			1		1			0				1		0	1
B					1									0	
A					√									√	
HASIL															
√			1					0				1			1

Legenda

A	-	Alice
B	-	Bob
R	-	*Rectilinear* (basis)
D	-	*Diagonal* (basis)
→	-	Polarisasi untuk 0 (basis *rectilinear*)
↑	-	Polarisasi untuk 1 (basis *rectilinear*)
↗	-	Polarisasi untuk 0 (basis *diagonal*)
↘	-	Polarisasi untuk 1 (basis *diagonal*)
√	-	ok

Gambar 18.1: Contoh Sesi Protokol Bennett-Brassard

Bab 19

Kebutuhan Akan Kriptografi

Penggunaan kriptografi mempunyai sejarah yang cukup panjang. Julius Caesar diketahui menggunakan *substitution cipher* untuk mengenkripsi pesan rahasia. Di jaman yang lebih modern, kriptografi digunakan oleh militer untuk merahasiakan perintah strategis melalui jalur komunikasi radio. Pesan rahasia antara pemerintahan suatu negara dengan misi diplomatiknya juga diamankan menggunakan kriptografi.

Dewasa ini, kriptografi tidak hanya dibutuhkan oleh militer dan misi diplomatik. Kemajuan teknologi komunikasi dan komputer membuat kriptografi dibutuhkan oleh kalangan yang lebih luas, bahkan boleh dikatakan bahwa semua orang yang menggunakan internet menggunakan kriptografi, meski sering tanpa disadari. Di bab ini, kita akan bahas beberapa situasi dimana kriptografi dibutuhkan.

19.1 Informasi Sensitif

Setiap orang, perusahaan, institusi dan instansi pemerintahan mempunyai informasi sensitif yang sebaiknya tidak jatuh ketangan orang yang tidak berhak untuk mendapatkannya. Contoh dari informasi sensitif seseorang secara pribadi antara lain:

- Informasi kesehatan pribadi.
- Informasi keuangan pribadi dan pola belanja.

Dari sisi kepentingan umum, memang ada informasi kesehatan pribadi yang dibutuhkan oleh instansi tertentu, misalnya untuk pencegahan penularan pe-

nyakit. Namun dari sisi pribadi, informasi kesehatan adalah sesuatu yang sensitif. Tentunya ada informasi yang perlu diketahui oleh dokter dan rumah sakit, misalnya kondisi jantung atau masalah alergi terhadap obat-obat tertentu. Tetapi jika informasi kesehatan pribadi disebar-luaskan, ada kemungkinan timbul tindakan atau reaksi diskriminatif terhadap yang bersangkutan. Informasi kesehatan pribadi juga bisa menjadi bahan untuk gosip. Demikian juga dengan informasi keuangan, instansi tertentu seperti instansi perpajakan mungkin perlu mengetahui informasi keuangan pribadi. Namun elemen-elemen kriminal dapat menggunakan informasi keuangan dan pola belanja seseorang untuk melakukan perbuatan kriminal.

Suatu perusahaan juga mempunyai informasi sensitif atau rahasia yang sebaiknya tidak disebar-luaskan. Contoh informasi sensitif yang perlu dirahasiakan oleh suatu perusahaan termasuk:

- Cara pembuatan suatu produk.
- Rencana strategis yang rinci.

Meskipun dari sisi pemegang saham, perusahaan diinginkan agar transparan, tentunya ada rahasia perusahaan seperti resep pembuatan suatu produk, yang jika jatuh ke perusahaan lain, akan menguntungkan perusahaan lain dan merugikan perusahaan pemilik resep. Kadang, rencana strategis yang rinci juga perlu dirahasiakan.

Untuk suatu institusi, contoh informasi sensitif termasuk:

- Alamat, nomor telpon dan email anggota institusi.
- Nilai akademis.

Kerap seorang anggota institusi tidak ingin informasi pribadinya diberikan kepada pihak ketiga. Informasi pribadinya bisa berupa alamat, nomor telpon dan email. Suatu institusi pendidikan seperti universitas juga menyimpan informasi sensitif. Sebagai contoh, nilai akademis mahasiswa sebaiknya tidak bisa begitu saja diberikan ke pihak ketiga tanpa persetujuan mahasiswa.

Suatu instansi pemerintahan juga memiliki informasi yang sensitif, sebagai contoh antara lain:

- Informasi pribadi pembayar pajak.
- Data mengenai persenjataan militer.

Meskipun publik menginginkan instansi pemerintahan yang transparan, ada informasi tertentu yang sensitif dan perlu dirahasiakan. Sebagai contoh, instansi perpajakan sebaiknya merahasiakan informasi pribadi seorang pembayar pajak, kecuali jika informasi tersebut diperlukan untuk kepentingan hukum. Contoh

lain adalah instansi militer dimana informasi tertentu mengenai persenjataan atau pergerakan pasukan dimasa perang perlu dirahasiakan.

Seseorang atau suatu organisasi tentunya bertanggung jawab atas kerahasiaan informasi sensitif yang dimilikinya, dan kriptografi dapat membantu menjaga kerahasiaan informasi sensitif yang disimpan secara elektronik. Berbagai langkah menggunakan kriptografi dapat diambil untuk menjaga kerahasiaan informasi sensitif yang disimpan secara elektronik termasuk:

- *Access control* terhadap informasi.
- Enkripsi data dalam transit.
- Enkripsi data dalam media penyimpanan.

Di jaman sekarang dimana komputer saling terhubung, *access control* bukan semata kontrol secara fisik, namun juga harus meliputi kontrol *access* secara *online*. Ini biasanya dilakukan menggunakan *password* atau *passphrase* dan diamankan menggunakan kriptografi sebagai berikut:

- menggunakan *secure hashing* untuk penyimpanan di *server*, dan
- menggunakan enkripsi untuk transmisi.

Data sensitif dalam transit juga perlu diamankan menggunakan enkripsi. Definisi data dalam transit juga mungkin perlu diperluas, bukan saja data yang sedang ditransmisi melalui jalur komunikasi, tetapi meliputi juga data dalam *notebook computer* dan *flash disk* yang keduanya dapat saja dicuri atau hilang. Dalam prakteknya, pengamanan data dalam transit dapat dilakukan dengan:

- Menggunakan *secure session* seperti SSL/TLS, SSH dan IPsec (akan dibahas di bab 20).
- Mengenkripsi *file* atau *file system* dalam *flash disk* dan *hard drive notebook computer*.

Untuk tingkat pengamanan yang lebih tinggi lagi, bukan hanya data sensitif yang disimpan dalam *flash disk* dan *hard drive notebook computer* saja yang perlu dienkripsi, tetapi juga data sensitif yang disimpan di media penyimpanan lain seperti *hard drive* untuk *desktop computer*. Ini terutama jika media dapat diakses oleh orang yang tidak diinginkan mengakses data sensitif tersebut.

19.2 Mencegah Penyadapan

Jika mendengar kata "penyadapan" maka yang terbayang di pikiran pembaca mungkin penyadapan oleh agen asing atau penyadapan oleh penegak hukum. Namun dewasa ini penyadapan komunikasi dapat dilakukan oleh siapa saja,

termasuk elemen kriminal, dengan peralatan yang relatif murah. Secara umum, komunikasi nirkabel rentan terhadap penyadapan karena penyadap tidak perlu akses fisik ke kabel komunikasi. Berikut adalah beberapa macam penyadapan yang sebagian diantaranya dapat dicegah menggunakan enkripsi:

- Penyadapan komunikasi nirkabel.
- Penyadapan komunikasi dengan kabel (tembaga maupun optik).
- Penyadapan radiasi elektromagnetik.
- Penyadapan akustik.

Dua standard komunikasi lokal nirkabel yang populer adalah Wi-Fi (IEEE 802.11) dan Bluetooth. Kedua protokol sebenarnya sudah menyediakan enkripsi, Wi-Fi melalui WEP dan WPA, dan Bluetooth melalui *security mode* 2, 3 dan 4 dan *encryption mode* 2 dan 3. Untuk Wi-Fi, sebaiknya WPA digunakan jika ada karena WEP terlalu mudah untuk dipecahkan (lihat 6.1). Akan tetapi ini tidak cukup jika kunci WPA atau WEP yang digunakan adalah kunci bersama, jadi sebaiknya gunakan enkripsi tambahan untuk data yang sensitif, contohnya menggunakan SSH (lihat bagian 20.2). Jika pembaca ingin rekomendasi yang lebih rinci mengenai pengamanan Wi-Fi yang sudah mendukung WPA, silahkan membaca [nis07]. Untuk Wi-Fi yang belum mendukung WPA, silahkan membaca [nis08a]. Bluetooth lebih rentan terhadap penyadapan karena limitasi perangkat, baik limitasi fitur keamanan yang ada dalam perangkat, maupun limitasi yang diakibatkan kesalahan implementasi fitur keamanan dalam perangkat. Jadi untuk Bluetooth, sebaiknya gunakan enkripsi tambahan untuk data sensitif. Untuk informasi yang lebih rinci mengenai pengamanan Bluetooth, silahkan membaca [nis08b]. Untuk komunikasi data sensitif melalui jaringan selular, jelas enkripsi tambahan diperlukan.

Hampir semua komunikasi melalui kabel menggunakan protokol internet yaitu TCP/IP. Sebetulnya, pada *layer* IP sudah tersedia pengamanan menggunakan enkripsi yaitu IPsec, yang akan dibahas di bagian 20.3. Namun untuk orang awam, *deployment* IPsec agak lebih sulit dibandingkan pengamanan sesi pada *layer* atas seperti SSL/TLS (akan dibahas di bagian 20.1) dan SSH (akan dibahas di bagian 20.2).

Untuk mencegah informasi bocor lewat radiasi elektromagnetik dari perangkat, biasanya perangkat dilindungi dengan *Faraday's cage*, yaitu "sangkar" dari bahan logam. Tentunya enkripsi sebaiknya tetap digunakan untuk komunikasi data sensitif dari/ke perangkat.

Penyadapan akustik bukan hanya penyadapan percakapan. Suara dari penggunaan *keyboard* dapat disadap, dan dengan analisa statistik frekuensi penggunaan setiap kunci di *keyboard*, apa yang diketik di *keyboard* dapat diketahui. Kemungkinan penyadapan akustik mungkin tidak terlalu besar dibandingkan kemungkinan adanya *trojan* yang melakukan *keyboard logging*. Namun

jika pembaca paranoid atau sedang menginput suatu rahasia negara yang sangat sensitif, maka pembaca sebaiknya menggunakan *keyboard* dalam ruangan kedap suara. Enkripsi secara tradisional tidak akan membantu untuk masalah ini. Yang mungkin dapat dilakukan adalah memancarkan juga secara *acak* bunyi berbagai kunci bersamaan dengan penggunaan *keyboard*. Jadi bunyi kunci yang ditekan bercampur dengan bunyi kunci secara acak. Konsep ini mirip dengan *steganography*, yaitu penyembunyian pesan dalam sesuatu yang lebih besar (contohnya dalam *file* gambar atau *file* audio). Bedanya, dalam *steganography*, orang yang kepada siapa pesan ditujukan, diharapkan dapat membaca pesan yang terpendam.

19.3 Mencegah Penyamaran

Di era *online* dimana komunikasi dilakukan melalui jaringan internet, identitas orang atau komputer yang hendak berkomunikasi dengan kita kadang perlu dipastikan. Sebagai contoh, jika akses ke suatu sistem informasi hanya diperbolehkan untuk pengguna yang telah terdaftar, maka sistem harus memastikan bahwa seorang yang ingin mengakses sistem adalah pengguna yang telah terdaftar. Berbagai contoh situasi dimana identitas perlu dipastikan antara lain:

- Seorang yang mengaku pengguna dan ingin mengakses sistem memang benar pengguna yang telah terdaftar.
- *Website* yang sedang kita kunjungi dan kepada siapa kita hendak mengirim nomor kartu kredit memang benar *website* yang kita kehendaki.
- Perangkat yang sedang mencoba untuk bergabung dalam jaringan komunikasi lokal nirkabel memang perangkat yang diperbolehkan untuk bergabung.

Memastikan identitas pengguna adalah bagian dari *access control* yang telah sedikit dibahas di bagian 19.1. Memastikan identitas kerap disebut *authentication*. Tren saat ini adalah menggunakan *multiple-factor authentication* yaitu menggunakan beberapa atribut unik pengguna sistem untuk identifikasi. Sifat atribut dapat berupa:

- Atribut langsung pengguna (*what you are*) misalnya sidik jari.
- Benda yang dimiliki pengguna (*what you have*) misalnya suatu *token*.
- Apa yang diketahui pengguna (*what you know*) misalnya *password* atau kunci privat.

Enkripsi berperan dalam mengamankan komunikasi mengenai apa yang diketahui pengguna, baik *password* maupun *authentication* menggunakan *public*

key cryptography. Tentunya *access control* bukan hanya memastikan identitas, tetapi juga meliputi kontrol terhadap apa yang dapat diakses oleh pengguna sistem. Kerap apa yang dapat diakses oleh satu pengguna berbeda dengan apa yang dapat diakses pengguna lain. Kontrol terhadap apa yang dapat diakses berbagai pengguna sistem dapat diamankan menggunakan kriptografi misalnya dengan mengenkripsi *file*.

Memastikan bahwa suatu *website* bukan merupakan penyamaran biasanya dilakukan menggunakan protokol SSL/TLS (lihat bagian 20.1) dimana identitas *website* didukung penggunaan *certificate* yang dikeluarkan suatu *certificate authority*. Akan tetapi kerap ini sebenarnya bukan apa yang diperlukan pengguna (lihat pembahasan di bagian 26.3).

Memastikan bahwa perangkat yang akan bergabung dalam jaringan komunikasi lokal nirkabel adalah perangkat yang diperbolehkan untuk bergabung sudah didukung oleh standard Wi-Fi dan Bluetooth. Untuk Wi-Fi, biasanya perangkat yang ingin bergabung (melalui *access point*) harus mengetahui kunci WPA untuk jaringan (tentunya ini hanya bisa jika jaringan menggunakan WPA). Untuk Bluetooth versi sebelum 2.1, dengan *security mode* 2 atau 3, *authentication* dilakukan menggunakan PIN. Untuk versi 2.1, dengan *security mode* 4, *authentication* dilakukan menggunakan Elliptic Curve Diffie-Hellman (lihat bab 17). Untuk informasi lebih rinci mengenai pengamanan Bluetooth, silahkan membaca [nis08b].

19.4 Ringkasan

Di bab ini telah dibahas berbagai situasi dimana kriptografi dibutuhkan. Telah dibahas kebutuhan pengamanan informasi sensitif, pencegahan penyadapan dan pencegahan penyamaran. Kriptografi berperan besar dalam melayani tiga kebutuhan tersebut. Bab-bab selanjutnya akan menjelaskan secara lebih rinci penggunaan kriptografi dalam melayani tiga kebutuhan tersebut.

Bab 20

Aplikasi - Pengamanan Sesi

Yang dimaksud dengan sesi (*session*) disini adalah sesi komunikasi. Ditingkat dasar seperti *layer* IP, sesi komunikasi biasanya bersifat umum. Pada *layer* atas, sesi komunikasi biasanya bersifat lebih khusus, misalnya untuk berkomunikasi dengan *shell*[1]. Kita akan bahas 3 macam pengamanan sesi yaitu:

- SSL/TLS untuk pengamanan sesi komunikasi antar proses.
- SSH untuk pengamanan sesi *shell*, yaitu sesi dimana antarmuka pengguna dengan sistem menggunakan *command line interface*.
- IPsec untuk pengamanan sesi koneksi internet umum (*layer* IP).

20.1 SSL/TLS

SSL adalah singkatan dari *Secure Socket Layer*, suatu *defacto standard* yang dibuat oleh Netscape, perusahaan pembuat *web browser* terpopuler tahun 1995. Ada dua versi SSL yang digunakan secara umum, yaitu SSL2 (lihat [hic95]) dan SSL3 (lihat [fri96]). Tahun 1999, standard SSL diambil alih oleh Internet Engineering Task Force (IETF) dan namanya diubah menjadi TLS (lihat [die99]), yang merupakan singkatan dari *Transport Layer Security* (SSL3 menjadi TLS versi 1). Perubahan nama ini mungkin agar nama menjadi lebih netral karena *socket* adalah istilah Unix. Versi terbaru dari TLS adalah versi 1.2 (lihat [die08]). Karena nama SSL sudah sangat melekat, meskipun nama sudah berganti, disini kita menyebutnya sebagai SSL/TLS.

[1] Ini adalah istilah Unix. Dalam dunia Microsoft Windows, yang mirip dengan *shell* adalah *command prompt*.

SSL/TLS dimaksudkan untuk mengamankan sesi komunikasi antar proses. Dalam suatu *operating system* yang *multi-tasking* seperti Unix, Microsoft Windows, Linux atau MacOS, berbagai proses berjalan secara bersamaan. (Untuk melihat berbagai proses yang aktif, jika menggunakan Microsoft Windows, pembaca dapat menggunakan Windows Task Manager. Jika menggunakan Unix, Linux atau MacOS 10, pembaca dapat menggunakan *command line program* `ps` untuk melihat berbagai proses yang aktif.) Pengamanan komunikasi tidak terbatas pada pengamanan komunikasi antar proses dalam satu komputer, akan tetapi juga pengamanan komunikasi antar proses di dua komputer yang berbeda. Bahkan pengamanan komunikasi antar proses di dua komputer yang berbeda jauh lebih penting karena terdapat lebih banyak ancaman. Sebagai contoh, penggunaan terbesar dari SSL/TLS adalah untuk "sesi aman" antara proses *web browser* dengan proses *web server* yang biasanya berada pada dua komputer yang berbeda.

20.1.1 Standard SSL/TLS

Dalam sesi menggunakan SSL/TLS, proses yang disebut *client* berkomunikasi dengan proses yang disebut *server*. Secara garis besar, sesi antara *client* dan *server* diamankan dengan, pertama melakukan *handshake*, lalu mengenkripsi komunikasi antara *client* dan *server* selama sesi berlangsung. Tujuan dari *handshake* adalah:

- *Server authentication* (*optional*).
- Menentukan parameter enkripsi.
- *Client authentication* (*optional*).

Bagian *handshake* mungkin merupakan yang terpenting dalam sesi SSL/TLS. Yang jelas *handshake* merupakan bagian paling rumit dari segi protokol. Secara garis besar, protokol *handshake* adalah sebagai berikut:

1. *Client* mengirim ke *server*: nomor versi SSL/TLS yang digunakan *client*, parameter enkripsi, data yang dibuat secara acak, dan informasi lain yang dibutuhkan oleh *server* untuk berkomunikasi dengan *client*. Jika dibutuhkan, *client* juga meminta *server certificate*.
2. *Server* mengirim ke *client*: nomor versi SSL/TLS yang digunakan *server*, parameter enkripsi, data yang dibuat secara acak, dan informasi lain yang dibutuhkan oleh *client* untuk berkomunikasi dengan *server*. Jika diminta dalam langkah 1, *server* juga mengirim *server certificate*. Jika dibutuhkan, *server* juga meminta *client* untuk mengirim *client certificate*.
3. Jika *client* meminta *server certificate* dalam langkah 1, *client* melakukan *server authentication* (akan dijelaskan secara lebih rinci) menggunakan

server certificate dan informasi lain yang didapat. Jika *authentication* sukses, *server certificate* tidak diminta, atau pengguna mengizinkan, *client* meneruskan ke langkah 4. Jika tidak, sesi dihentikan.

4. Menggunakan data yang telah didapat, *client* membuat suatu *premaster secret* untuk sesi. Tergantung jenis enkripsi yang digunakan, ini dapat dilakukan dengan partisipasi *server*. *Premaster secret* dienkripsi menggunakan kunci publik *server* (diambil dari *server certificate*), lalu dikirim ke *server*.

5. Jika *server* meminta *client certificate* pada langkah 2, *client* menandatangan secara *digital* data yang unik untuk sesi yang diketahui oleh *client* dan *server*. Data berikut *digital signature* dan *client certificate* dikirim oleh *client* ke *server*.

6. Jika *server* meminta *client certificate* pada langkah 2, *server* melakukan *client authentication*. Jika *authentication* diminta dan gagal, maka sesi dihentikan.

7. *Client* dan *server* membuat *master secret* menggunakan *premaster secret*. *Master secret* digunakan oleh *client* dan *server* untuk membuat kunci sesi yang merupakan kunci enkripsi simetris.

8. *Client* memberi tahu *server* bahwa kunci sesi akan digunakan untuk mengenkripsi komunikasi lebih lanjut. *Client* kemudian mengirim pesan yang dienkripsi ke *server* yang mengatakan bahwa ia selesai dengan *handshake*.

9. *Server* memberi tahu *client* bahwa kunci sesi akan digunakan untuk mengenkripsi komunikasi lebih lanjut. *Server* kemudian mengirim pesan yang dienkripsi ke *client* yang mengatakan bahwa ia selesai dengan *handshake*.

10. *Handshake* selesai.

Server authentication dilakukan dengan memeriksa *server certificate*. Dalam *server certificate* terdapat informasi antara lain:

- kunci publik *server*,
- masa berlaku *certificate*,
- *domain name* untuk *server*, dan
- *domain name* untuk pembuat *certificate* (biasanya *certificate* dibuat oleh suatu *certificate authority*).

Server certificate ditanda-tangan secara *digital* oleh pembuatnya. Pemeriksaan *certificate* dilakukan dengan menjawab berbagai pertanyaan sebagai berikut:

- Apakah tanggal pemeriksaan berada dalam masa berlaku *certificate*?
- Apakah pembuat *certificate* (teridentifikasi dengan *domain name* pembuat) dapat kita percayai? Pembuat bisa merupakan suatu *certificate authority* atau sumber lain.
- Apakah *certificate* ditanda-tangan secara *digital* dengan benar oleh pembuatnya?
- Apakah *domain name* untuk *server* yang tertera dalam *certificate* sesuai dengan *domain name* untuk *server* yang sebenarnya?

Jika semua pertanyaan terjawab secara memuaskan, maka *authentication* sukses. Jika ada pertanyaan yang jawabannya tidak memuaskan, maka *authentication* gagal. *Client authentication* juga dilakukan serupa. Setelah *handshake* selesai, komunikasi antara *client* dan *server* dienkripsi menggunakan kunci sesi.

Standard SSL/TLS tentunya juga menentukan format yang harus digunakan dalam komunikasi antara *client* dan *server*. Standard juga menentukan *cipher suite* yang dapat digunakan. *Cipher suite* adalah kombinasi metode enkripsi yang terdiri dari:

- Metode enkripsi untuk *key exchange* atau *key agreement*.
- Metode enkripsi untuk *certificate*.
- Metode enkripsi menggunakan kunci sesi.
- Metode enkripsi untuk *message authentication code* (MAC).

Untuk *key exchange* atau *key agreement*, metode enkripsi yang dapat digunakan antara lain:

- RSA, dan
- Diffie-Hellman (DH).

Untuk *certificate*, metode enkripsi yang dapat digunakan antara lain:

- RSA, dan
- DSA/DSS.

Untuk enkripsi menggunakan kunci sesi, metode enkripsi yang dapat digunakan antara lain:

- RC4,
- 3DES,

- AES 128 bit, dan
- AES 256 bit.

Jika menggunakan *block cipher* seperti 3DES atau AES, maka metode enkripsi harus menggunakan mode *cipher block chaining* (CBC). Untuk *message authentication code*, metode enkripsi yang dapat digunakan antara lain:

- HMAC-MD5,
- HMAC-SHA-1,
- HMAC-SHA256.

Standard TLS versi 1.2 mengharuskan implementasi sedikitnya bisa mendukung *cipher suite* yang terdiri dari RSA untuk *key exchange* dan *certificate*, AES 128 bit mode CBC untuk enkripsi menggunakan kunci sesi, dan HMAC-SHA-1 untuk *message authentication code*. Untuk mendapatkan informasi teknis secara rinci mengenai TLS versi 1.2, silahkan membaca [die08].

20.1.2 Penggunaan SSL/TLS

Seperti dikatakan sebelumnya, penggunaan terbesar SSL/TLS adalah untuk *secure web browsing*. Semua *web browser* yang populer mendukung *secure web browsing* menggunakan SSL/TLS. Biasanya pengguna *web browser* tidak perlu mengetahui SSL/TLS. Saat *web browser* akan menampilkan *web page* dengan *prefix* `https` (jadi bukan `http`), maka *web browser* secara otomatis akan memulai sesi SSL/TLS. Dalam melakukan *handshake* SSL/TLS, *web browser* akan melakukan *server authentication* dengan memeriksa *certificate* untuk *web server*. *Web browser* biasanya sudah memiliki daftar *certificate authority* yang dapat dipercaya, dan ada *web browser* yang memperbolehkan pengguna untuk menambah pembuat *certificate* yang dipercaya kedalam daftar. Jika ada masalah dalam *authentication* maka *web browser* biasanya memberi tahu pengguna dan menanyakan pengguna apakah sesi diteruskan atau tidak. Biasanya ada juga opsi untuk menambah *certificate* yang bermasalah ke daftar pengecualian dimana *certificate* yang ada dalam daftar tidak perlu diperiksa. Jika *handshake* SSL/TLS berhasil maka *web page* dapat ditampilkan setelah terlebih dahulu di*download* dengan proteksi sesi SSL/TLS.

Selain *web browser*, perangkat lunak untuk *web server* seperti Apache juga sudah mendukung SSL/TLS. Jadi pembaca dapat membuat *web server* yang dapat melayani pengguna internet dengan *secure web page*. Apache memberi opsi untuk menggunakan *server certificate* yang dibuat sendiri atau dibuat oleh suatu *certificate authority*. Tentunya ada kemungkinan *web browser* pengguna akan berpendapat bahwa *certificate* bermasalah, akan tetapi ini dapat diatasi seperti dibahas di paragraf sebelum ini.

Karena *web browser* dan *web server* mendukung SSL/TLS, maka semua sistem informasi *client-server* yang berbasis *web* dapat menggunakan fasilitas SSL/TLS dengan mudah. Buku ini sangat merekomendasikan pendekatan *client-server* yang berbasis *web* untuk suatu sistem informasi. Penggunaan sistem informasi akan sangat fleksibel karena pengguna dapat berada dimana saja asalkan ada koneksi TCP/IP dengan *server*, contohnya:

- di komputer yang sama dengan *server*,
- di komputer lain yang terhubung dengen *server* melalui *local area network*, atau
- di lokasi lain, bahkan di negara yang berbeda waktu 12 jam dengan *server*, asalkan ada koneksi internet ke *server*.

Jika sistem informasi dibuat menggunakan sesuatu yang *platform independent* seperti Apache-MySQL-PHP, maka sistem informasi akan lebih fleksibel lagi karena *server* dapat ditempatkan di komputer dengan *operating system* apa saja, baik Microsoft Windows, Linux, MacOS 10, FreeBSD, OpenBSD, atau varian Unix lainnya, dan dapat dengan mudah dipindahkan.

Tentunya penggunaan SSL/TLS tidak harus melalui *web*. Pada prinsipnya apa saja yang memerlukan pengamanan komunikasi antar proses dapat menggunakan SSL/TLS. Ada beberapa alat yang dapat membantu pembuatan sistem yang menggunakan SSL/TLS. Contohnya adalah *cryptographic library* seperti RSA BSafe (lihat bagian 24.2) atau Cryptlib (lihat bagian 24.3). Contoh lain adalah OpenSSL yang juga memberikan fasilitas *cryptographic library* selain memberikan fasilitas SSL/TLS. Cryptlib dan OpenSSH keduanya menggunakan OpenSSL. OpenSSL adalah implementasi *open source* dari SSL/TLS yang didasarkan pada SSLeay, implementasi SSL oleh Eric Young. Meskipun *open source*, OpenSSL menggunakan *licensing* yang tidak kompatibel dengan GPL (*GNU Public License*). Untuk pembaca yang menginginkan implementasi SSL yang lebih kompatibel dengan GPL, GnuTLS adalah implementasi serupa yang menggunakan LGPL (*Lesser GNU Public License*).

20.2 SSH

Asal mula dari *secure shell* (SSH) adalah *software* yang dibuat oleh Tatu Ylönen, ssh, yang dimaksudkan sebagai program Unix untuk *secure remote shell access*, menggantikan telnet dan program *remote shell access* lainnya yang tidak aman. Protokol SSH kemudian dijadikan standard IETF.

Secara tradisional, dalam mengakses sistem jenis Unix, *shell* kerap digunakan sebagai antarmuka. Jika sistem yang diakses berada di komputer lain, maka program Unix untuk *remote shell access* seperti telnet atau rlogin perlu

digunakan. Namun penggunaan telnet atau rlogin tidak aman karena dapat disadap. Lebih parah dari itu, *password* untuk mengakses sistem dapat dicuri, misalnya menggunakan *password sniffer*. SSH berfungsi seperti telnet dan rlogin, tetapi komunikasi antara komputer pengguna dan komputer yang diakses diamankan. Pengamanan bukan hanya untuk saat yang penting seperti *login*, tetapi juga untuk komunikasi selanjutnya. SSH menggunakan konsep *client-server* dimana *client* terdapat di komputer pengguna dan *server* terdapat di komputer dimana *shell* berjalan.

Ada perbedaan pendapat antara Ylönen dan para pembuat OpenSSH mengenai status nama ssh. Ylönen berpendapat bahwa ssh adalah *trademark* yang menjadi hak miliknya, sedangkan pembuat OpenSSH berpendapat bahwa nama tersebut sudah menjadi nama generik sehingga tidak dapat digunakan sebagai *trademark*. Buku ini tidak berpihak dalam hal tersebut, tetapi sekedar mengungkapkan adanya perbedaan pendapat.

20.2.1 Standard SSH

Standard IETF untuk SSH (ada yang menyebutnya SSH-2) dipublikasikan pada tahun 2006 dalam bentuk sekumpulan RFC ([leh06], [ylo06a], [ylo06b], [ylo06c], [ylo06d], [sch06] dan [cus06]). Standard mendefinisikan arsitektur dari protokol SSH terdiri dari berbagai *layer* sebagai berikut:

- *Transport layer* (lihat [ylo06c]), *layer* terbawah untuk SSH. *Layer* ini fungsinya mirip dengan SSL/TLS. Jadi *handshake* dilakukan di *layer* ini, dan juga komunikasi pada tingkat paket, yang diamankan menggunakan enkripsi. *Authentication* yang dilakukan pada *handshake* adalah *server authentication*. *Layer* ini juga melakukan pergantian kunci sesi, misalnya setiap 1 GB data dikomunikasikan atau setiap 1 jam. *Layer* ini memberikan layanan ke *layer* atas melalui antarmuka yang telah ditetapkan.

- *User authentication layer* (lihat [ylo06b]), *layer* untuk *login* ke *server*. SSH memberikan fleksibilitas kepada *server* dan *client* mengenai cara *user authentication*. *Server* dapat menawarkan beberapa alternatif untuk cara *user authentication*. *Client* kemudian mencoba satu per satu satu diantaranya dengan urutan yang sembarang. Berikut adalah beberapa cara yang dapat digunakan untuk *user authentication*:

  - *Public key infrastructure*. Hanya cara ini yang harus didukung suatu implementasi untuk *user authentication*. Dengan cara ini, *server* mengetahui kunci publik dari suatu pasangan kunci. Pengguna harus memiliki kunci privat dari pasangan kunci, yang digunakan untuk membuat *digital signature* sesuatu yang dikirim *server*. *Server* mengecek *digital signature* tersebut untuk menentukan apakah benar

pengguna memiliki kunci privat dari pasangan kunci. Algoritma *digital signature* yang dapat digunakan antara lain RSA dan DSA/DSS.

- *Password*. Ini adalah cara yang sederhana untuk *user authentication*. *Layer* ini juga memberi fasilitas untuk pergantian *password*.
- *Host-based*. Dengan cara ini siapapun dapat mengakses *shell* asalkan *client* berada pada komputer tertentu berdasarkan *domain name* atau nomor IP. *Authentication* dilakukan dengan mengecek *certificate* untuk komputer dimana *client* berada.

Ada mekanisme untuk menambah cara *user authentication* tanpa merubah standard yaitu *generic message exchange authentication* [cus06].

- *Connection layer* (lihat [ylo06d]). Di *layer* ini konsep *global request*, *channel*, sesi interaktif, dan TCP/IP *port forwarding* didefinisikan. *Global request* adalah permintaan yang dapat merubah *state* di sisi *server* independen dari semua *channel*. Setiap sesi dan *forwarded port connection* merupakan *channel*. Jenis *channel* yang standard termasuk:
  - *shell*,
  - *direct* TCP/IP, untuk *forwarded connection* yang tidak diminta, jadi *port forwarding* atas inisiatif sisi pengirim sendiri (pengirim bisa *client* bisa *server*), dan
  - *forwarded* TCP/IP, untuk *forwarded connection* yang diminta oleh penerima (penerima bisa *client* bisa *server*).

  Satu *connection* dapat terdiri dari beberapa *channel*.

Untuk informasi teknis yang rinci mengenai arsitektur protokol SSH, silahkan membaca [ylo06a].

Kadang, sewaktu melakukan *server authentication*, *client* tidak mengenal kunci publik *server*. Jika demikian, *fingerprint* dari kunci publik dapat diperlihatkan ke pengguna. Jika pengguna cukup yakin bahwa kunci publik memang milik *server* yang ia ingin akses, maka sesi dapat diteruskan. Ada metode alternatif untuk mengecek *fingerprint* yang ditawarkan oleh standard SSH dengan menggunakan fasilitas DNSSEC (lihat [sch06]). *Fingerprint* dari *server* dapat dipublikasikan secara aman sebagai bagian dari DNS *record*.

20.2.2 Penggunaan SSH

Penggunaan SSH secara tradisional memang untuk mengakses *shell* di komputer lain (*server*). Biasanya *server* menggunakan *operating system* jenis Unix. Beberapa *operating system* jenis Unix yang populer saat ini termasuk:

- Linux,

- Mac OS 10,
- Solaris,
- FreeBSD, dan
- OpenBSD.

Dua implementasi SSH yang cukup dikenal adalah ssh (suatu produk komersial) dan OpenSSH (*open source*). Untuk OpenSSH, *client* dapat juga ditempatkan di komputer dengan *operating system* Microsoft Windows.

Penggunaan lain SSH adalah sebagai jalur aman bagi suatu program *client-server*. Istilah penggunaan ini adalah *tunneling*. Dua program yang termasuk dalam paket OpenSSH:

- SFTP dan
- SCP,

menggunakan SSH sebagai *tunnel*. SFTP berfungsi seperti program ftp, meskipun fiturnya diluar pengamanan jauh lebih sedikit dibandingkan ftp. SCP adalah program untuk *secure remote copy*. Menggunakan SCP *files* bisa di*copy* antar komputer secara aman, biasanya antara komputer *client* dan komputer *server*. Beberapa aplikasi dari pihak ketiga juga menggunakan OpenSSH untuk *tunneling*.

Sejak versi 4.3, OpenSSH dapat digunakan untuk membuat suatu *virtual private network* berdasarkan OSI *layer* 2/3 TUN. Suatu *virtual private network* (VPN) adalah jaringan komputer didalam jaringan yang lebih besar dimana koneksi antar komputer dalam VPN diamankan sehingga VPN seolah membuat jaringan sendiri. Menggunakan OpenSSH, VPN *tunnel* antara dua komputer dapat dibuat dimana di setiap komputer ujungnya adalah suatu *virtual device*, sebut saja `tun0`. Tentunya `tun0` harus dikonfigurasi untuk memiliki IP *address*, contohnya menggunakan `ipconfig` untuk Microsoft Windows atau `ifconfig` untuk Linux atau jenis Unix lainnya. Meskipun OpenSSH dapat digunakan untuk membuat VPN, karena *overhead* yang tinggi, solusi ini tidak se-efisien solusi menggunakan IPsec (lihat bagian 20.3). Kecuali untuk penggunaan darurat atau sementara, penggunaan OpenSSH untuk implementasi VPN tidak direkomendasikan. Perlu ditekankan disini bahwa VPN *tunneling* yang dilakukan oleh OpenSSH tidak ada dalam standard SSH.

20.3 IPsec

IPsec adalah protokol pengamanan komunikasi pada *layer* IP (Internet Protocol). Secara garis besar, IPsec mengamankan komunikasi dengan memberikan fasilitas *authentication* dan enkripsi untuk setiap paket IP. Dalam protokol

IPsec ditentukan juga cara untuk *mutual authentication* di permulaan sesi dan cara untuk negosiasi kunci sesi.

20.3.1 Standard IPsec

Ada 3 tempat dimana standard IPsec dapat dimplementasi:

- terintegrasi dalam IP *stack*,
- diluar komputer (*bump in the wire*), atau
- dibawah IP *stack* diatas *device driver* (*bump in the stack*).

IP *stack* untuk berbagai *operating system* sudah mengintegrasikan *standard* IPsec. Solusi kedua secara tradisional merupakan solusi militer menggunakan perangkat keras khusus. Solusi ketiga bisa digunakan jika IP *stack* belum mengintegrasikan *standard* IPsec.

Dari segi arsitektur protokol (lihat [ken05a]), IPSec terdiri dari 4 komponen:

- protokol untuk *authentication header* (AH),
- protokol untuk *encapsulating security payload* (ESP),
- konsep *security association*, terdiri dari beberapa parameter pengamanan yang menentukan bagaimana suatu sambungan satu arah menggunakan AH atau ESP diamankan, dan
- protokol untuk manajemen *security association*, termasuk *key management*, secara otomatis menggunakan *internet key exchange* (IKE atau IKEv2).

Authentication header (AH) memberi jaminan integritas data, *authentication* terhadap asal dari paket IP, dan pencegahan *replay attack*. AH mengamankan integritas IP *payload* dan setiap *field* dalam *header* kecuali jika *field* adalah *mutable field* (*field* yang dapat berubah dalam perjalanan paket, contohnya TTL). Integritas IP *payload* diamankan menggunakan *integrity check value* (ICV). *Authentication* juga dijamin oleh ICV secara tidak langsung dengan penggunaan kunci rahasia bersama dalam kalkulasi ICV. Ada dua mode penggunaan AH:

- *transport mode* untuk mengamankan *layer* atas termasuk TCP, dan
- *tunnel mode* untuk mengamankan "*inner*" IP.

Tabel 20.1 memperlihatkan format paket AH.

- *Next header* adalah *field* sebesar 8 bit yang mengidentifikasi jenis *payload*. Isinya merupakan IP *protocol number* sesuai dengan yang telah ditentukan oleh *Internet Assigned Numbers Authority* (IANA). Sebagai contoh, *protocol number* 4 mengidentifikasi IPv4, sedangkan 6 mengidentifikasi TCP.

Bit 0-7	Bit 8-15	Bit 16-23	Bit 24-31
Next header	Payload length	Reserved	
Security parameters index (SPI)			
Sequence number field			
Integrity check value - ICV (variable)			

Tabel 20.1: Format paket AH

- *Payload length* adalah *field* sebesar 8 bit yang memberi tahu besarnya AH. Nilai *field* ini ditambah 2 lalu dikalikan dengan 32 bit merupakan besarnya AH.

- *Reserved* adalah *field* sebesar 16 bit untuk penggunaan masa depan. Pengirim paket harus mengisi *field* ini dengan 0. Penerima paket menggunakan nilai *field* ini dalam komputasi ICV, dan setelah itu dapat mengabaikannya.

- SPI adalah *field* sebesar 32 bit yang digunakan oleh penerima paket untuk menentukan *security association* yang berlaku.

- *Sequence number* adalah field sebesar 32 bit yang digunakan untuk mencegah *replay attack*. Untuk setiap *security association*, paket pertama yang dikirim mempunyai *sequence number* 1, paket kedua mempunyai *sequence number* 2 dan seterusnya. Penerima mencegah *replay attack* menggunakan *sliding window*. AH dengan *sequence number* diluar *sliding window* otomatis ditolak. Duplikat AH juga ditolak. Jika AH dengan *sequence number* sama dengan nilai terkecil *sliding window* sudah diterima, maka *sliding window* dapat digeser. Besar *sliding window* minimal 32, direkomendasikan 64, tetapi implementasi dapat menggunakan *sliding window* yang lebih besar.

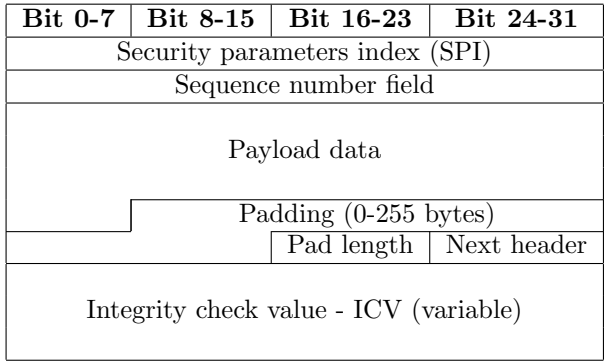
- ICV adalah *field* yang besarnya adalah kelipatan dari 32 bit. Pengirim paket mengkalkulasi ICV dan menempatkannya di *field* ini. Penerima paket juga mengkalkulasi ICV dan membandingkannya dengan *field* ini. Jika cocok maka penerima menganggap integritas paket tidak terganggu.

Untuk mendapatkan informasi teknis yang lebih rinci mengenai *authentication header*, termasuk *transport mode* dan *tunnel mode*, penggunaan SPI untuk menentukan *security association*, kalkulasi ICV, dan *packet fragmentation* dan *reassembly*, silahkan membaca [ken05b].

Encapsulating security payload (ESP) memberi fasilitas pengamanan kerahasiaan data, *authentication* terhadap asal data, dan integritas. ESP dapat digunakan sendiri, bersama dengan AH, atau secara berlapis. Pengamanan kerahasiaan data dilakukan menggunakan enkripsi, sedangkan integritas dan *authentication* terhadap asal data dilakukan menggunakan ICV seperti halnya dengan AH. Juga seperti halnya dengan AH, ada dua mode penggunaan ESP:

- *transport mode* untuk mengamankan *layer* atas termasuk TCP, dan
- *tunnel mode* untuk mengamankan "*inner*" IP.

Tabel 20.2 memperlihatkan format paket ESP.



Tabel 20.2: Format paket ESP

- *Payload data* adalah data yang bisa dienkripsi bisa tidak. Jika dienkripsi maka *payload data* termasuk data sinkronisasi enkripsi seperti *initialization vector*.
- *Padding* diperlukan untuk dua macam keperluan yaitu agar besar naskah yang dienkripsi merupakan kelipatan dari ukuran blok dan agar ruang antara ahir dari naskah acak dan permulaan dari *pad length* terisi. Oleh sebab itu bisa jadi sebagian dari *padding* dienkripsi dan sebagian tidak dienkripsi.
- *Pad length* adalah *field* yang memberi tahu besarnya *padding* dalam byte.

Security parameter index, *sequence number*, *next header* dan *integrity check value*, isinya dan fungsinya sama seperti dalam AH. Meskipun enkripsi dan integritas (termasuk *authentication*) dapat digunakan secara independen, ESP biasanya digunakan untuk keduanya. Jadi 3 kombinasi enkripsi dan integritas yang dapat digunakan adalah:

- *confidentiality only* (BOLEH didukung implementasi),
- *integrity only* (HARUS didukung implementasi), dan
- *confidentiality and integrity* (HARUS didukung implementasi).

Untuk mendapatkan informasi teknis yang lebih rinci mengenai *encapsulating security payload*, termasuk *transport mode* dan *tunnel mode*, penggunaan SPI untuk menentukan *security association*, enkripsi, *padding*, kalkulasi ICV, dan *packet fragmentation* dan *reassembly*, silahkan membaca [ken05c].

Security association (SA) terdiri dari beberapa parameter yang menentukan bagaimana komunikasi paket satu arah menggunakan AH atau ESP diamankan. Jadi untuk komunikasi dua arah diperlukan sepasang SA, satu untuk setiap arah. SA yang digunakan ditentukan oleh SPI dan jenis pengamanan (AH atau ESP). Komputer yang terlibat dalam komunikasi AH atau ESP menyimpan setiap SA dalam suatu *database*. Parameter yang ada dalam SA termasuk:

- *security parameter index* (SPI),
- *counter* untuk *sequence number* (untuk paket keluar),
- *sequence counter overflow* yaitu *flag* yang mengindikasikan terjadinya *overflow* pada *counter*,
- *anti-replay window* (untuk paket yang diterima),
- algoritma dan kunci untuk *authentication* (untuk AH dan ESP),
- algoritma, kunci dan IV untuk enkripsi (untuk ESP),
- *lifetime* dari SA, bisa berupa jangka waktu atau jumlah byte,
- *mode*: *transport* atau *tunnel*,
- kebijakan untuk IP *filtering*,
- dan lainnya.

Setiap *device* TCP/IP yang menggunakan IPsec mempunyai *security policy database* (SPD) dan *security association database* (SAD). Suatu *security policy* dalam SPD adalah kebijakan bagaimana komunikasi TCP/IP lewat *device* tersebut harus diproses, termasuk apakah paket tertentu harus diproses menggunakan IPsec. Jadi *security policy* bersifat lebih umum, sedangkan *security association* bersifat lebih khusus.

Manajemen *security association* antar komputer dilakukan menggunakan protokol *internet key exchange* (versi kini IKEv2), terutama untuk *key management*. Komunikasi menggunakan IKEv2 selalu terdiri dari pasangan *request*

dan *response*. Pasangan *request* dan *response* disebut *exchange*. Komunikasi menggunakan IKEv2 selalu dimulai dengan *initial exchanges*, antara *initiator* dan *responder*, terdiri dari dua *exchange* yaitu IKE-SA-INIT dan IKE-AUTH. IKE-SA-INIT fungsinya adalah negosiasi algoritma kriptografi dan melakukan Diffie-Hellman *key agreement*. Untuk IKE-SA-INIT, pesan pertama adalah dari *initiator* ke *responder* sebagai berikut:

$$\begin{array}{ll} \textit{Initiator} & \textit{Responder} \\ HDR, SA_{i1}, KE_i, N_i & \rightarrow \end{array}$$

dimana HDR adalah *header* IKE terdiri dari SPI, berbagai nomor versi dan berbagai *flag*, SA_{i1} mendeklarasikan berbagai alternatif algoritma kriptografi yang didukung oleh *initiator*, KE_i adalah nilai Diffie-Hellman dari *initiator*, dan N_i adalah *nonce*[2] dari *initiator*. Pesan kedua dalam IKE-SA-INIT adalah dari *responder* ke *initiator* sebagai berikut:

$$\begin{array}{lll} \textit{Initiator} & & \textit{Responder} \\ & \leftarrow & HDR, SA_{r1}, KE_r, N_r[, CERTREQ] \end{array}$$

dimana HDR adalah *header* IKE, SA_{r1} menyatakan pilihan *cryptographic suite* berdasarkan apa yang ditawarkan SA_{i1}, KE_r adalah nilai Diffie-Hellman dari *responder*, dan N_r adalah *nonce* dari *responder*. *Responder* dapat meminta *certificate* dari *initiator* menggunakan CERTREQ (*optional*). Setelah pesan ini, kedua pihak (*initiator* dan *responder*) dapat menyepakati suatu *seed* yang dinamakan SKEYSEED untuk membuat berbagai kunci (*seed* disepakati menggunakan Diffie-Hellman *key agreement*), termasuk SK_a yaitu kunci untuk *authentication* dan SK_e yaitu kunci untuk enkripsi. Setiap pesan setelah IKE-SA-INIT diamankan menggunakan SK_a dan SK_e (kecuali *field* HDR). *Exchange* kedua dalam IKEv2 adalah IKE-AUTH, yang fungsinya adalah *authentication* pesan-pesan dalam IKE-SA-INIT dan negosiasi CHILD-SA. Pesan pertama dalam IKE-AUTH adalah dari *initiator* ke *responder* sebagai berikut:

$$\begin{array}{ll} \textit{Initiator} & \textit{Responder} \\ HDR, ID_i, [CERT,][CERTREQ,][ID_r,] & \rightarrow \\ AUTH, SA_{i2}, TS_i, TS_r & \end{array}$$

dimana pesan (kecuali *field* HDR) diamankan menggunakan SK_a dan SK_e. ID_i adalah identitas dari *initiator*. Jika *certificate* diminta oleh *responder* maka $CERT$ berisi *certificate*. *Initiator* juga dapat meminta *certificate responder* menggunakan $CERTREQ$ (*optional*). ID_r bersifat *optional* dan berisi pilihan *initiator* dari identitas *responder*. $AUTH$ adalah data *authentication* untuk pesan pertama dalam IKE-SA-INIT. SA_{i2}, TS_i dan TS_r digunakan untuk negosiasi CHILD-SA, dimana SA_{i2} adalah SA yang ditawarkan, TS_i dan

[2] *Nonce* adalah suatu nilai acak yang setiap kali dibuat, diharapkan berbeda dari *nonce* lainnya. Fungsi dari *nonce* adalah untuk mencegah *replay attack*.

TS_r adalah *traffic selectors* yang ditawarkan. Pesan kedua dalam IKE-AUTH adalah dari *responder* ke *initiator* sebagai berikut:

$$\begin{array}{lll} \textit{Initiator} & & \textit{Responder} \\ & \leftarrow & HDR, ID_r, [CERT,] \\ & & AUTH, SA_{r2}, TS_i, TS_r. \end{array}$$

Seperti halnya dengan pesan pertama, pesan kecuali *field* HDR diamankan menggunakan SK_a dan SK_e. ID_r adalah identitas *responder*. Jika *certificate* diminta oleh *initiator*, maka $CERT$ berisi *certificate*. $AUTH$ adalah data *authentication* untuk pesan kedua dalam IKE-SA-INIT, SA_{r2} adalah SA yang disetujui *responder* untuk CHILD-SA dan TS_i dan TS_r adalah *traffic selectors* yang disetujui. Setelah IKE-AUTH, CHILD-SA dapat dibuat atau CHILD-SA yang sudah ada dapat diganti kuncinya menggunakan *exchange* CREATE-CHILD-SA. Kedua pihak dapat memulai *exchange* ini, jadi *initiator* untuk CREATE-CHILD-SA tidak harus *initiator* untuk IKE-SA-INIT dan IKE-AUTH. Pesan pertama dalam CREATE-CHILD-SA adalah sebagai berikut:

$$\begin{array}{lll} \textit{Initiator} & & \textit{Responder} \\ HDR, [N,]SA, Ni[, KE_i][, TS_i, TS_r] & \rightarrow & \end{array}$$

Jika CREATE-CHILD-SA sedang digunakan untuk mengganti kunci, maka N mengidentifikasi SA yang hendak diganti kuncinya. SA adalah tawaran berbagai parameter untuk SA, dan N_i adalah *nonce*. KE_i (*optional*) adalah nilai Diffie-Hellman yang digunakan untuk memperkuat kunci SA. Jika SA menawarkan beberapa grup untuk Diffie-Hellman, maka KE_i merupakan elemen dari grup yang seharusnya dapat disetujui *responder* (jika tidak disetujui, maka *exchange* gagal dan harus diulang dengan nilai KE_i yang lain). Jika CREATE-CHILD-SA sedang membuat CHILD-SA yang baru (jadi bukan mengganti kunci), maka TS_i dan TS_r berisi *traffic selectors* yang ditawarkan. Jawaban untuk pesan pertama adalah pesan berikut:

$$\begin{array}{lll} \textit{Initiator} & & \textit{Responder} \\ & \leftarrow & HDR, SA, N_r, [KE_r,] \\ & & [TS_i, TS_r] \end{array}$$

dimana SA adalah berbagai parameter SA yang dipilih oleh *responder* dari yang ditawarkan *initiator*, dan N_r adalah *nonce*. Jika pesan pertama memuat KE_i dan berasal dari grup yang disetujui dalam SA, maka KE_r merupakan nilai Diffie-Hellman dari *responder*. Jika KE_i berasal dari grup yang tidak ada dalam SA, maka *exchange* harus ditolak, dan *initiator* harus mengulang *exchange* menggunakan KE_i yang berada dalam grup yang disetujui SA. Jika CREATE-CHILD-SA bukan sedang digunakan untuk mengganti kunci, maka TS_i dan TS_r adalah *traffic selectors* yang dipilih oleh *responder* dari yang

ditawarkan oleh *initiator*. Selain *exchange* jenis IKE-SA-INIT, IKE-AUTH dan CREATE-CHILD-SA, ada satu lagi jenis *exchange* dalam IKEv2 yaitu INFORMATIONAL. *Exchange* INFORMATIONAL digunakan untuk pesan *control* dan dapat berisi notifikasi, penghapusan dan konfigurasi. Untuk mendapatkan informasi yang rinci mengenai *exchange* INFORMATIONAL dan informasi lainnya mengenai IKEv2, silahkan membaca [kau05].

20.3.2 Penggunaan IPsec

Karena IPsec melakukan pengamanan komunikasi pada *level* IP, IPsec banyak digunakan untuk implementasi *virtual private network* (VPN), yang memang secara umum beroperasi di *level* IP. Ini jauh lebih efisien dibandingkan implementasi menggunakan OpenSSH karena IPsec lebih terintegrasi dengan IP dibandingkan OpenSSH.

Ada sedikit komplikasi jika IPsec digunakan melewati NAT (*network address translation*), yang digunakan oleh berbagai perangkat seperti:

- *consumer broadband modem/router*,
- *firewall*,

dimana *address* IP internet di *interface* eksternal perangkat diterjemahkan menjadi *address* IP privat yang digunakan jaringan internal seperti 192.168.1.5. (Situasinya sebetulnya lebih rumit lagi karena biasanya terdapat lebih dari satu *address* IP privat di jaringan internal, kadang paket dari luar harus diteruskan ke salah satu dari sekian *address* IP privat yang ada berdasarkan *port address*.) Protokol AH tidak kompatibel dengan NAT karena *address* IP dalam paket diganti, sedangkan *address* IP merupakan *field* yang diamankan AH dan bukan merupakan *mutable field*. Beruntung untuk ESP, *address* IP bukan merupakan *field* yang diamankan, jadi ESP dapat digunakan bersamaan dengan NAT. Mungkin itulah sebabnya ESP juga mendukung *authentication* disamping enkripsi, karena kita tidak dapat menggunakan AH untuk *authentication* yang melewati NAT. Akan tetapi meskipun pergantian *address* IP bukan merupakan masalah bagi ESP, persoalan *mapping* antara *address* IP eksternal dengan *address* IP privat perlu dipecahkan. Salah satu solusi untuk ini adalah menggunakan NAT *traversal* dalam IKEv2 dan UDP *encapsulation* untuk ESP. Untuk informasi yang lebih rinci mengenai hal ini, silahkan membaca [kau05].

20.4 Ringkasan

Di bab ini telah dibahas tiga standard pengamanan sesi yaitu SSL/TLS, SSH dan IPsec. SSL/TLS dimaksudkan untuk pengamanan komunikasi antar proses, dan banyak digunakan untuk *secure web pages*. SSH dimaksudkan untuk

pengamanan penggunaan *remote shell*. Satu implementasi SSH yaitu OpenSSH mendukung penggunaannya sebagai *virtual private network* (VPN), namun penggunaan ini tidak efisien dan hanya direkomendasikan untuk penggunaan darurat atau sementara. IPsec adalah standard pengamanan pada *level* IP, oleh sebab itu penggunaan IPsec dalam implementasi VPN sangat direkomendasikan, meskipun penggunaannya dengan *network address translation* (NAT) sedikit rumit.

Bab 21

Aplikasi - Pengamanan Email

Meskipun banyak orang yang tidak menyadari, pengamanan *email* merupakan sesuatu yang penting untuk berbagai situasi. *Email* biasanya tidak dienkripsi jadi rentan terhadap penyadapan. Ini terutama jika menggunakan fasilitas seperti Yahoo atau Gmail, jelas pihak Yahoo atau Google dapat membaca *email* yang dikirimkan dari atau ke fasilitas mereka. *Email* juga bisa dipalsukan, seorang dapat saja mengaku sebagai orang lain dalam mengirim email, bahkan menggunakan *email address* orang lain sebagai *address* pengirim. Jadi seperti halnya dengan pengamanan sesi (lihat bab 20), diperlukan dua macam pengamanan untuk *email* yaitu:

- *authentication* dan
- enkripsi.

Untuk memastikan bahwa *email* bukan sesuatu yang dipalsukan, mekanisme *authentication* biasanya menggunakan *digital signature*. *Email* ditanda-tangan secara *digital* menggunakan kunci privat pengirim, dan diperiksa penerima menggunakan kunci publik pengirim. Untuk memastikan bahwa hanya penerima yang diinginkan yang dapat membacanya, *email* dienkripsi menggunakan kunci publik penerima, jadi hanya bisa didekripsi oleh pemegang kunci privat penerima. (Sebetulnya yang dienkripsi menggunakan kunci publik penerima adalah kunci sesi, sedangkan *email* dienkripsi menggunakan kunci sesi. Penerima mendekripsi kunci sesi menggunakan kunci privat, lalu mendekripsi *email* menggunakan kunci sesi.) Jadi cukup jelas bahwa pengamanan *email* memerlukan kriptografi *public key*.

Ada dua standard pengamanan *email* yang populer yaitu:

- S/MIME (*Secure/Multipurpose Internet Mail Extensions*) dan
- OpenPGP.

S/MIME berbasis pada format CMS (*cryptographic message syntax*), yang berawal pada PKCS#7, jadi berorientasi pada X.509 (lihat bagian 23.2), sedangkan OpenPGP berbasis pada format PGP (lihat bagian 23.1). Kedua standard juga menggunakan format *multipurpose internet mail extensions* (MIME) untuk integrasi dengan *email*. Cara berfungsi kedua standard sebenarnya serupa, hanya format saja yang berbeda. Tabel 21.1 memperlihatkan beberapa perbedaan format antara S/MIME dan OpenPGP.

Feature	S/MIME	OpenPGP
Message Format	CMS	PGP
Certificate Format	X.509	PGP
Symmetric Encryption	3DES, AES, IDEA, CAST	3DES, AES, CAST, Blowfish
Key Exchange	Diffie-Hellman, RSA	ElGamal, RSA
Digital Signature	RSA, DSA/DSS	DSA/DSS, RSA
Hash	SHA	SHA
MIME Encapsulation Signed Data	multipart/signed or CMS	multipart/signed with ASCII armor
MIME Encapsulation Encrypted Data	application/pkcs7-mime	multipart/encrypted

Tabel 21.1: Perbedaan S/MIME dengan OpenPGP

Tabel 21.1 menunjukkan bahwa S/MIME dan OpenPGP menggunakan kumpulan algoritma kriptografi yang serupa. Perbedaan terletak pada berbagai format yang digunakan. (Untuk mendapatkan informasi yang rinci mengenai format S/MIME versi 3.1, lihat [ram04]. Untuk mendapatkan informasi yang rinci mengenai format OpenPGP, lihat [cal07].) S/MIME bersumber pada "*the establishment*" yang awalnya berorientasi pada RSA dan berbagai standard PKCS yang dikembangkan oleh RSA. Meskipun tidak bisa disebut "*anti-establishment*," PGP dikembangkan oleh Philip Zimmerman sebagai alternatif untuk RSA yang saat itu (tahun 1991) masih dilindungi hak paten, dan sebagai taktik gerilya melawan kontrol pemerintahan Amerika Serikat terhadap kriptografi saat itu. Dengan mengendurnya kontrol pemerintahan Amerika Serikat terhadap kriptografi dan habisnya masa berlaku paten untuk RSA, pembagian pengguna kriptografi menjadi "*the establishment*" versus "*outsider*" sudah tidak berlaku lagi. Tren saat ini mengarah pada standardisasi dengan S/MIME sebagai standard yang dipilih. Bahkan GnuPG, suatu program *open*

source untuk OpenPGP, kini juga mendukung standard S/MIME. Oleh sebab itu selanjutnya kita akan lebih fokus pada S/MIME.

Secara garis besar, pengamanan *email* dilakukan dengan apa yang disebut *data enveloping* atau memasukkan data kedalam amplop. Berbagai jenis *data enveloping* antara lain:

- *Basic data enveloping.*
- *Compressed data enveloping.*
- *Encryption enveloping.*
- *Authenticated enveloping.*

Basic enveloping hanya sekedar membungkus data menjadi suatu unit tanpa memproses data. *Compressed data enveloping*, selain membungkus juga melakukan kompresi data (jadi yang dibungkus adalah data yang sudah dikompresi). Demikian juga *encryption enveloping* melakukan enkripsi terhadap data sebelum dibungkus. Sedangkan *authenticated enveloping* melakukan *authentication* terhadap data yang dibungkus. Yang membuat konsep *enveloping* sangat berguna adalah kita dapat mengkombinasi beberapa jenis *enveloping* secara berlapis. Sebagai contoh, kita dapat melakukan *compressed enveloping* untuk mengkompres data, kemudian melakukan *encrypted enveloping* terhadap *compressed envelope* untuk mengenkripsinya, kemudian melakukan *authenticated enveloping* terhadap *encrypted envelope* dengan melakukan *digital signing.*

Sesuatu yang sangat penting dalam S/MIME adalah *certificate* untuk kunci publik, baik kunci publik untuk mengecek *digital signature* maupun kunci publik untuk enkripsi. *Certificate* merupakan sesuatu yang seharusnya dibuat oleh seorang atau badan yang dapat dipercaya. Biasanya pembuat adalah suatu *certificate authority*, namun seorang dapat juga membuat *self-signed certificate*. Tentunya siapa yang dapat dipercaya adalah sesuatu yang relatif. Sebagai contoh, pemerintah Perancis tentunya tidak akan mempercayai *certificate authority* komersial dari Amerika Serikat seperti Verisign untuk keperluan resmi negara Perancis. Jika agak ragu dengan kebenaran *certificate*, kita dapat mengecek *fingerprint* dari kunci publik melalui jalur lain (misalnya telpon) dengan pemilik kunci publik. Manajemen *certificate* akan dibahas lebih lanjut di bab 23.

Berbagai program *email* terkemuka seperti Outlook dan Mozilla Thunderbird sudah mendukung S/MIME. Menggunakan program *email* yang sudah mendukung S/MIME, untuk mengenkripsi atau menanda-tangan secara *digital email* yang akan dikirim, tinggal memilih opsi sewaktu membuat *email.* Menerima *email* yang dienkripsi juga seolah menerima *email* biasa, asalkan *email* dienkripsi menggunakan kunci publik penerima. Akan tetapi ada resiko dengan penerimaan *email* yang dienkripsi, karena *virus scanner* biasanya

tidak bisa mendeteksi virus yang telah dienkripsi. Untuk mengurangi resiko sebaiknya ikuti petunjuk sebagai berikut:

- Jangan publikasikan kunci publik kita kecuali kepada orang-orang yang kita kehendaki.
- Jangan buka *email* yang dienkripsi kecuali dari orang-orang yang telah kita berikan kunci publik.

Verifikasi *digital signature* juga bisa otomatis, mirip dengan verifikasi *secure web page* (lihat bagian 20.1).

Untuk program *email* yang belum mendukung S/MIME, biasanya kita dapat menambahkan S/MIME, misalnya menggunakan cryptlib (lihat bagian 24.3). Untuk *web-based email* ceritanya agak berbeda. Mayoritas *web-based email* tidak memberikan fasilitas untuk S/MIME dan agak lebih sukar untuk mengintegrasi S/MIME dengan *web-based email*. Jika mendapatkan *email* yang diamankan menggunakan S/MIME, *envelope* akan berupa *attachment* yang dapat di*save* ke *file* contohnya `smime.p7m`. *File* `smime.p7m` dapat dibuka menggunakan program seperti p7mviewer atau menggunakan cryptlib (lihat bagian 24.3). Sebaliknya, jika ingin mengirimkan *email* yang diamankan menggunakan S/MIME, *file attachment* dapat dibuat menggunakan cryptlib. Untuk yang menggunakan Gmail, *browser* Mozilla Firefox kini mendukung penggunaan S/MIME dengan Gmail, dengan meng*install* Gmail S/MIME *extension*.

21.1 Ringkasan

Di bab ini kita telah bahas keperluan pengamanan *email* dan dua standard pengamanan *email* yaitu S/MIME dan OpenPGP. Beberapa program *email* terkemuka sudah mendukung S/MIME dan penggunaannya sangat mudah (istilahnya *seemless*), namun penerimaan *email* yang dienkripsi harus dengan hati-hati.

Bab 22

Aplikasi - Authentication

Authentication adalah komponen yang sangat penting dalam pengamanan sistem informasi. Boleh dikatakan bahwa semua standard pengamanan sistem informasi, termasuk SSL/TLS, SSH, IPsec, S/MIME dan OpenPGP, mempunyai komponen *authentication*. Pengamanan non-standard biasanya juga mempunyai komponen *authentication*, contohnya menggunakan *password*. Penggunaan suatu komputer dapat mewajibkan pengguna untuk *login* menggunakan *password*. Namun di jaman yang serba terhubung seperti sekarang, fasilitas sistem informasi suatu organisasi yang cukup besar biasanya tidak terdapat pada hanya satu komputer tetapi terdistribusi pada beberapa komputer yang terhubung melalui suatu *local area network* (LAN). Sangat tidak efisien jika pengguna harus *login* ke setiap aplikasi yang tersedia dalam jaringan. Kerberos mencoba membantu dalam masalah ini dengan menggunakan *centralized authentication*, dimana *authentication* dipusatkan di *authentication server*.

22.1 Kerberos

Kerberos dikembangkan di Massachusetts Institute of Technology (MIT) sebagai bagian dari proyek Athena bersama dengan Digital Equipment Corporation (DEC) dan IBM. Proyek Athena berlangsung dari tahun 1983 sampai dengan tahun 1991. Saat Kerberos dikembangkan, satu masalah yang dihadapi dalam penggunaan LAN di MIT adalah tidak adanya *authentication* yang efektif ketika seorang hendak menggunakan aplikasi di komputer lain di jaringan. Contohnya, koneksi ke komputer menggunakan rlogin tidak melalui *authentication* yang aman, komputer yang melayani (*server*) hanya mengecek nama dari pengguna dalam daftar, jadi sangat mudah untuk mengelabui *server*. Salah satu tujuan Kerberos adalah untuk mengamankan berbagai aplikasi yang disediakan oleh proyek Athena, dengan melakukan *authentication* terhadap calon

pengguna.

Karena *authentication* dipusatkan di *authentication server*, protokol Kerberos tidak cukup hanya melakukan *client authentication*, tetapi juga harus memperkenalkan *client* ke *server* yang diminta layanannya oleh *client*. Ada empat aktor dalam protokol Kerberos:

- *client*,
- *authentication server* (AS),
- *ticket-granting server* (TGS), dan
- *server*.

Tugas AS adalah melakukan *authentication* terhadap *client* sedangkan tugas TGS adalah memperkenalkan *client* ke *server*. Meskipun AS dan TGS adalah dua entitas yang berbeda, keduanya bisa ditempatkan di satu komputer. Kombinasi AS dan TGS biasanya disebut sebagai *key distribution center* (KDC). Kunci simetris untuk setiap pengguna (*client*) harus diregistrasi di AS dan kunci simetris untuk setiap *server* harus diregistrasi di TGS. Biasanya kunci pengguna didapat dari *password* melalui *hashing* (lihat bab 9). Protokol Kerberos didasarkan pada protokol Needham-Schroeder dan secara garis besar berjalan sebagai berikut:

1. *Client* melaporkan ke AS bahwa ia ingin menggunakan layanan *server*.
2. AS mengecek daftar *client*. Jika *client* (berikut kuncinya) ada dalam daftar, AS membuat kunci sesi K_1 untuk komunikasi antara *client* dengan TGS. AS kemudian membuat *ticket-granting ticket* (TGT) yang dienkripsi menggunakan kunci TGS dan isinya termasuk identitas *client*, *client network address*, masa berlaku TGT dan K_1. K_1 yang dienkripsi menggunakan kunci *client* dan TGT keduanya dikirimkan ke *client*.
3. *Client* mendekripsi menggunakan kunci *client* untuk mendapatkan K_1 dan kemudian membuat permohonan yang dikirimkan ke TGS dan terdiri dari TGT, identitas *server* dan suatu *authenticator* A_1. A_1 itu sendiri dienkripsi menggunakan K_1 dan isinya termasuk identitas *client*, *client network address* dan *timestamp* (waktu saat A_1 dibuat).
4. TGS mendekripsi TGT menggunakan kunci TGS untuk mendapatkan K_1, identitas *client*, *client network address* dan masa berlaku TGT. Jika TGT masih berlaku maka TGS kemudian mendekripsi A_1 menggunakan K_1 untuk melakukan validasi. Jika validasi sukses maka TGS membuat kunci sesi K_2 untuk komunikasi antara *client* dan *server*. TGS kemudian membuat *client-to-server ticket* (CST) yang dienkripsi menggunakan kunci *server* dan isinya termasuk identitas *client*, *client network address*,

masa berlaku CST dan K_2. K_2 yang dienkripsi menggunakan K_1 dan CST dikirimkan ke *client*.

5. *Client* mendekripsi menggunakan K_1 untuk mendapatkan K_2 lalu membuat *authenticator* A_2 yang dienkripsi menggunakan K_2 dan isinya termasuk identitas *client*, *client network address* dan *timestamp* (waktu saat A_2 dibuat). CST dan A_2 dikirimkan ke *server*.

6. *Server* mendekripsi CST menggunakan kunci *server* untuk mendapatkan K_2, identitas *client*, *client network address* dan masa berlaku CST. Jika CST masih berlaku, *server* kemudian mendekripsi A_2 menggunakan K_2 untuk melakukan validasi. Jika validasi sukses maka *server* membuat konfirmasi yang isinya termasuk *timestamp* dan identitas *server*. Konfirmasi dienkripsi menggunakan K_2 dan dikirimkan ke *client*.

7. Menggunakan K_2, *client* mendekripsi dan mengecek konfirmasi. Jika tidak bermasalah maka *client* dapat memulai permintaan pelayanan dari *server*.

8. *Server* dapat melayani permintaan *client*.

Authentication yang dilakukan AS terhadap *client* bersifat tidak langsung, yaitu dengan mengenkripsi kunci sesi antara *client* dengan TGS menggunakan kunci *client*. Jika seorang yang bukan *client* mengaku sebagai *client* dan meminta TGT kepada AS, maka itu akan sia-sia karena orang tersebut tidak bisa mendapatkan kunci sesi (ia tidak memiliki kunci *client* dan tidak memiliki kunci TGS). Tanpa kunci sesi antara *client* dengan TGS, *authenticator* A_1 yang *valid* tidak bisa dibuat.

Protokol Needham-Schroeder versi awal tidak menggunakan *timestamp*. Akan tetapi, tanpa *timestamp*, protokol rentan terhadap *replay attack*. Oleh sebab itu Kerberos menggunakan *timestamp* untuk mencegah *replay attack*. Validasi *authenticator* yang dilakukan TGS dan *server*, selain mengecek identitas *client* dan *client network address*, juga mengecek *timestamp*. Jika umur *timestamp* melebihi batas yang ditentukan (biasanya 5 menit), maka *authenticator* dianggap tidak *valid*. TGS dan *server* juga mengelola daftar terdiri dari permintaan yang telah divalidasi dalam 5 menit terahir. Jika ada permintaan ulang dalam jangka waktu 5 menit (ini dapat dicek menggunakan daftar), maka permintaan ulang tersebut ditolak.

Kerberos dapat di*download* dalam bentuk *source code* dari MIT. *Web site* untuk Kerberos adalah `http://web.mit.edu/Kerberos/`. Saat bab ini ditulis versi terbaru Kerberos adalah versi 5-1.7. Berbagai dokumentasi mengenai Kerberos juga bisa didapat dari *web site* Kerberos, termasuk *Installation Guide*, *User's Guide* dan *Administrator's Guide*.

Untuk dapat menggunakan aplikasi dengan Kerberos, tentunya aplikasi harus mendukung protokol Kerberos (istilahnya aplikasi sudah *Kerberized*),

baik disisi *client* maupun disisi *server*. Ini dapat dilakukan dengan mengintegrasi protokol Kerberos langsung dalam aplikasi atau menggunakan suatu *wrapper* yang mendukung Kerberos. Aplikasi disisi *client* juga harus mengetahui kunci *client* yang diregistrasi dengan AS, sedangkan aplikasi disisi *server* juga harus mengetahui kunci *server* yang diregistrasi dengan TGS. Beberapa program Unix yang *Kerberized* sudah termasuk dalam Kerberos versi 5-1.7, antara lain telnet, rlogin, ftp, rsh, rcp dan ksu.

Setiap *administrative domain* (misalnya `athena.mit.edu`) mempunyai *key distribution center* (KDC) sendiri dan dinamakan *realm*. Jika *client* ingin menggunakan layanan *server* di *realm* yang lain, maka diperlukan *cross-realm authentication*. TGS untuk *server* (sebut saja STGS) harus diregistrasi di TGS untuk *client*, atau setidaknya harus ada rantai

client, AS, TGS, TGS1 , ... , TGSn, STGS, *server*

dimana STGS diregistrasi di TGSn, TGSn diregistrasi di TGSn-1, ... , TGS2 diregistrasi di TGS1 dan TGS1 diregistrasi di TGS. Untuk *authentication*, *client* harus melalui rantai tersebut:

- oleh AS, *client* diberikan TGT untuk TGS;
- jika TGT dapat divalidasi oleh TGS, *client* diberikan TGT1 untuk TGS1;
- jika TGT1 dapat divalidasi oleh TGS1, *client* diberikan TGT2 untuk TGS2;
- dan seterusnya.

Hanya STGS yang dapat memberikan CST kepada *client* untuk kemudian diberikan kepada *server*.

Kerberos banyak digunakan untuk *single-sign-on*. Sebagai contoh, menggunakan sistem Unix, pengguna hanya mengetik *password* satu kali. Setiap kali menggunakan program *client-server* yang sudah *Kerberized* seperti telnet, pengguna tidak perlu mengetik *password* lagi.

22.2 Ringkasan

Di bab ini kita telah bahas *client-server authentication* menggunakan Kerberos. Kerberos banyak digunakan untuk implementasi *single-sign-on* dan dapat di*download* dalam bentuk *source code* dari

`http://web.mit.edu/Kerberos/.`

Bab 23

Aplikasi - PKI

Hampir semua aplikasi kriptografi di bidang teknologi informasi menggunakan kriptografi *public key*. Aplikasi dasar kriptografi *public key* memang untuk *key management* dan *digital signature*. Namun bermula dari aplikasi dasar, kebutuhan berkembang dan *certificate management* kini menjadi bagian penting dari kriptografi *public key*. Suatu *public key infrastructure* (PKI) adalah suatu infrastruktur yang mendukung

- *key management* termasuk *key generation*, *key exchange*, dan *key agreement*,
- *digital signing* dan *digital signature checking*, dan
- *certificate management* termasuk *certificate generation*, *certificate publishing*, *certificate checking*, dan *certificate revocation*.

Key generation dalam hal ini adalah proses pembuatan pasangan kunci publik dan privat, baik untuk keperluan *digital signature* maupun keperluan *key management*. Pembuatan kunci simetris tidak masuk disini karena tidak melibatkan unsur kriptografi publik, kecuali jika menggunakan *key agreement*. *Key exchange* adalah proses pengiriman kunci simetris secara aman menggunakan kriptogafi *public key*. *Key agreement* adalah proses pembuatan kunci simetris secara bersama menggunakan kriptografi *public key*, contohnya menggunakan Diffie-Hellman (lihat bagian 16.2). Kunci simetris hasil *key agreement* bisa berupa *seed* yang digunakan untuk membuat kunci simetris lainnya.

Digital signing adalah proses pembuatan *digital signature* contohnya menggunakan RSA atau DSA, sedangkan *digital signature checking* adalah proses pengecekan *digital signature* apakah sesuai dengan naskah yang di*sign* dan kunci publik yang di*claim*.

Certificate generation adalah proses pembuatan *certificate* untuk kunci publik, baik untuk keperluan *digital signature* maupun untuk keperluan *key management*. Secara garis besar, ada dua standard untuk format *certificate* yaitu format X.509 dan format PGP. *Certificate publishing* bisa dilakukan menggunakan fasilitas seperti LDAP (*lightweight directory access protocol*) atau cara yang lebih sederhana. *Certificate checking* adalah proses pengecekan *certificate*, baik dari segi format, maupun dari segi isi. *Certificate revocation* adalah proses pembatalan suatu *certificate* yang telah dibuat. Semua aspek *certificate management* dapat melibatkan *certificate authority* yaitu seorang atau suatu badan yang fungsi utamanya adalah mengesahkan *certificate*.

Solusi PKI biasanya berbasis pada X.509 atau pada PGP. Pada awalnya, solusi berbasis X.509 adalah solusi komersial, sedangkan solusi PGP populer di kalangan *open source* saat RSA masih dilindungi hak paten dan pemerintah Amerika Serikat melakukan kontrol yang ketat[1] terhadap penggunaan kriptografi. Dengan habisnya masa berlaku paten untuk RSA dan mengendurnya kontrol pemerintah Amerika Serikat terhadap penggunaan kriptografi, tren saat ini adalah standardisasi kearah X.509.

23.1 PGP

Solusi berbasis PGP untuk PKI menggunakan format dan "cara" PGP. "Cara" PGP adalah pendekatan gerilya dimana struktur hirarkis formal tidak terlalu diperhatikan. Jadi konsep *certificate authority* tidak digunakan. Setiap pengguna dapat menentukan sendiri apakah suatu kunci publik *valid*:

- Sebagai tanda bahwa pengguna, sebut saja A, menganggap bahwa kunci publik B *valid*, A membuat *certificate* untuk kunci publik B.
- Kunci publik juga *valid* jika memiliki *certificate* yang dibuat oleh pemilik kunci publik *valid* dengan *complete trust*, asalkan rantai *certificate* dari pengguna panjangnya tidak lebih dari 5. Rantai *certificate* contohnya A membuat *certificate* untuk kunci B, B membuat *certificate* untuk kunci C, C membuat *certificate* untuk kunci D. Dengan contoh rantai, jika kunci publik C *valid* dengan *complete trust*, make kunci publik D *valid*.
- Kunci publik dengan tiga *certificate* yang masing-masing dibuat oleh pemilik kunci publik *valid* dengan *marginal trust* juga dianggap *valid*, asalkan rantai *certificate* dari pengguna panjangnya tidak lebih dari 5.

Nilai parameter untuk banyaknya *certificate* dengan *full trust* yang diperlukan, banyaknya *certificate* dengan *marginal trust* yang diperlukan, dan panjang ran-

[1]Ini hanya dalam teori. Dalam prakteknya pemerintah Amerika Serikat tidak dapat mengontrol penggunaan kriptografi.

tai, semua dapat diubah oleh pengguna. Nilai diatas adalah nilai *default*. Pengguna dapat menentukan *trust level* dari suatu kunci publik sebagai pembuat *certificate*. Kunci publik pengguna sendiri dianggap memiliki *complete trust*. Kunci publik *valid* yang lain dapat diberi *trust level*:

- *complete trust*,
- *marginal trust*, atau
- *untrusted*.

Kunci publik yang bukan *valid* otomatis dianggap *untrusted*. *Trust model* yang digunakan PGP dinamakan *web of trust*. Selain *valid*, status kunci publik bisa:

- *marginally valid* jika hanya ada satu *certificate* untuk kunci tersebut dan *certificate* dibuat menggunakan kunci dengan *marginal trust*, atau
- *invalid* jika tidak ada *certificate* untuk kunci tersebut yang dibuat menggunakan kunci dengan *complete trust* atau *marginal trust*.

Certificate berisi antara lain:

- nomor versi PGP,
- kunci publik pemegang *certificate*,
- informasi mengenai pemegang *certificate*, contohnya nama, alamat *email*,
- *self-signature* yaitu kunci publik pemegang *certificate* di*sign* menggunakan kunci privat pemegang *certificate* (ini merupakan bukti kepemilikan kunci privat),
- masa berlaku *certificate*, dan
- *preferred symmetric encryption algorithm*,

tetapi tidak terbatas pada itu. Untuk mendapatkan informasi yang lebih rinci mengenai standard format PGP yang terkini (dinamakan OpenPGP), silahkan membaca [cal07].

GnuPG (Gnu *Privacy Guard*) adalah suatu implementasi *open source* untuk OpenPGP yang banyak digunakan untuk *secure email* dan enkripsi *file* terutama oleh pengguna yang merasa X.509 terlalu rumit. Sesuatu yang menarik dengan GnuPG adalah dukungan terhadap penggunaan *smartcard*.

23.2 X.509

X.509 adalah bagian dari X.500, suatu percobaan yang sangat ambisius dari ITU (International Telecommunication Union) untuk standardisasi *directory services*. Judul dari standard X.509 adalah *The Directory: Public-key and attribute certificate frameworks*. X.500 sendiri dianggap terlalu rumit, namun format *certificate* X.509 dijadikan dasar untuk standard defacto oleh industri, dan kini standard *public key infrastructure* untuk internet yaitu PKIX berdasarkan pada format X.509.

Berbeda dengan "cara" PGP yang cenderung demokratis, "cara" X.509 lebih bersifat struktural hirarkis. Ini adalah warisan dari X.500 yang semula ditujukan untuk membuat direktori pengguna jaringan komputer seperti direktori telpon yang akan dikelola oleh berbagai perusahaan telpon di berbagai negara. Sifat-sifat struktural dalam X.509 antara lain:

- Konsep *relative distinguished name* (RDN) yang cukup rumit dalam membuat *distinguished name* (DN) (identitas).

- Konsep *certificate authority* yang semula dikaitkan dengan RDN.

Konsep *relative distinguished name* (RDN) bertujuan untuk membuat identitas berdasarkan pada posisi dalam suatu hirarki. Hirarki ini diatur berdasarkan negara, propinsi, organisasi, sub-unit organisasi dan seterusnya. Berbagai negara dan organisasi melakukan lokalisasi standard yang kerap disebut *profile* yang menambah rumit konsep RDN. Semula, *certificate authority* (CA) juga dikaitkan dengan RDN, dimana satu CA mempunyai otoritas untuk satu subhirarki. Namun dalam prakteknya ini tidak berjalan.

Karena konsep RDN terlalu rumit dan dapat bertentangan dengan berbagai kepentingan seperti:

- *privacy* (contohnya informasi pribadi), dan

- *confidentiality* (contohnya banyak perusahaan yang tidak menginginkan informasi mengenai struktur organisasinya dipublikasikan),

belum lagi pada kenyataannya dunia tidak sepenuhnya tersusun rapi secara hirarkis tanpa tumpang tindih, akibatnya boleh dikatakan tidak ada yang mengetahui bagaimana cara membuat DN yang "benar." Meskipun demikian, DN tetap merupakan bagian tak terpisahkan dari *certificate* X.509, dan X.509 sudah merupakan *defacto standard*. Berikut adalah berbagai komponen DN yang kerap digunakan:

Komponen	**Penjelasan**
CountryName(C)	Kode negara 2 huruf berdasarkan ISO 3166 (contohnya ID).
Organization(O)	Nama organisasi (contohnya PT Sendiri).
OrganizationalUnit(OU)	Nama unit organisasi (contohnya Sales).
StateOrProvince(SP)	Nama negara bagian atau propinsi (contohnya DKI).
Locality(L)	Nama daerah (contohnya Kebayoran Baru).
CommonName(CN)	Nama pemegang *certificate*, bisa perorangan atau bagian dari organisasi, bahkan nama komputer (contohnya Budi Santoso).

Setiap komponen kecuali CountryName besarnya maksimum 64 karakter, sedangkan CountryName besarnya 2 karakter. Masih banyak komponen DN lainnya yang dapat digunakan, namun komponen yang terpenting sudah masuk dalam daftar diatas. Minimal suatu DN harus mempunyai komponen C dan komponen CN. DN dapat mengidentifikasi orang, unit organisasi atau perangkat, dan digunakan baik untuk identifikasi pemegang *certificate* maupun untuk identifikasi pembuat *certificate*.

Sejak penggunaan internet menjadi dominan, ada alternatif dari DN yang disebut *general name* (GN) yang lebih berorientasi pada internet dibandingkan dengan DN. Komponen GN termasuk antara lain DNS *name*, IP *address*, *email address* dan *uniform resource identifier*. Untuk identifikasi perangkat, penggunaan GN mungkin lebih cocok dibandingkan DN.

Sekarang mari kita lihat isi dari *certificate* X.509. Sebetulnya ada bermacam jenis struktur untuk *certificate* X.509, termasuk

- *certificate* biasa,
- *attribute certificate*,
- *certification request* dan
- *certificate revocation list* (CRL).

Untuk *certificate* biasa, berikut adalah daftar komponen beserta penjelasannya:

Komponen	Penjelasan
Version	Versi X.509 yang digunakan.
SerialNumber	Nomor seri. Setiap *certificate* yang dibuat oleh suatu CA harus mempunyai nomor seri yang berbeda.
SignatureAlgorithm	Algoritma kriptografi yang digunakan untuk menanda-tangan *certificate*.
IssuerName	DN/GN untuk pembuat *certificate* (biasanya CA).
Validity	Masa berlaku *certificate*.
SubjectName	DN/GN untuk pemegang *certificate*.
SubjectPublicKeyInfo	Kunci publik pemegang *certificate*.
Extensions	Untuk berbagai macam informasi tambahan.

Attribute certificate fungsinya adalah sebagai *certificate* untuk berbagai atribut, jadi komponen SubjectPublicKeyInfo diganti oleh Attributes. *Certificate request* fungsinya adalah permintaan *certificate* untuk suatu kunci publik, jadi hanya berisi komponen berikut:

Komponen	Penjelasan
Version	Versi X.509 yang digunakan.
SubjectName	DN/GN untuk pemegang kunci.
SubjectPublicKeyInfo	Kunci publik yang diminta dibuatkan *certificate*.
Extensions	Untuk berbagai macam informasi tambahan.

Certificate revocation list (CRL) dimaksudkan sebagai daftar *certificate* yang dibatalkan oleh suatu CA. Untuk berbagai macam alasan, termasuk kunci yang dicuri, suatu *certificate* dapat dibatalkan oleh CA yang membuatnya. Secara berkala, suatu CA mempublikasikan daftar *certificate* yang dibatalkannya dalam bentuk CRL. CRL dapat digunakan untuk menentukan validitas suatu

certificate. Isi dari CRL adalah sebagai berikut:

Komponen	**Penjelasan**
Version	Versi X.509 yang digunakan.
SignatureAlgorithm	Algoritma kriptografi yang digunakan untuk menanda-tangan *certificate*.
IssuerName	DN/GN untuk CA.
ThisUpdate	Waktu CRL ini dibuat.
NextUpdate	Waktu CRL berikutnya akan dibuat.
UserCertificate	Daftar *certificate* yang dibatalkan.
RevocationDate	Tanggal *certificate* dibatalkan.

Meskipun dalam X.500 *certificate authority* (CA) dikaitkan dengan RDN, kini tidak ada lagi kaitan formal antara CA dengan RDN. *Certificate* bahkan tidak harus dibuat oleh CA. Namun konsep CA masih berguna, meskipun siapa yang patut menjadi CA tergantung pada aplikasi dan/atau situasi. Validitas dari suatu kunci publik ditentukan oleh *certificate* untuk kunci tersebut: jika *certificate* dibuat oleh orang, badan atau CA yang dipercaya oleh seorang pengguna, maka kunci publik dianggap valid oleh pengguna tersebut. Perbedaan utama antara "cara" X.509 dan "cara" PGP dalam menentukan validitas *certificate* adalah dengan X.509, CA biasanya tidak ditentukan oleh pengguna, sedangkan dengan PGP pengguna berperan lebih besar dalam menentukan siapa yang dipercaya sebagai pembuat *certificate* menggunakan mekanisme *web of trust*.

Dari segi teknis, masalah yang dihadapi oleh X.509 dalam manajemen validitas *certificate* jauh lebih rumit dibandingkan masalah yang dihadapi PGP. Ini antara lain karena:

- Berbagai CA dan aktor lainnya berperan aktif dalam manajemen validitas *certificate*.
- Bermacam skenario penggunaan *certificate* X.509 dapat melibatkan aktor yang beragam dan dalam skala yang besar.
- Konsep *certificate revocation* sukar untuk dipraktekkan secara efektif.

Sebagai contoh banyaknya aktor yang berperan dalam manajemen validitas *certificate*, kita bisa lihat begitu banyaknya *certificate authority* yang ada, baik

yang bersifat komersial seperti Verisign, maupun yang non-komersial. Contoh penggunaan *certificate* X.509 yang beragam termasuk untuk keperluan *website authentication* (lihat bagian 20.1), untuk *authentication* kunci publik suatu perangkat dalam jaringan yang menggunakan IPsec untuk VPN (lihat bagian 20.3), dan untuk *secure email* menggunakan S/MIME (lihat bab 21). Sukarnya menggunakan konsep *certificate revocation* secara efektif pernah dijelaskan oleh Peter Gutmann (arsitek dari Cryptlib, lihat bagian 24.3) dengan membuat analogi dengan *relational database* dimana mempublikasikan *certificate* ibarat operasi *prepare* dalam suatu *relational database*, sedangkan *certificate revocation* ibarat operasi *commit* dalam *relational database*:

- Dalam *relational database*, status setelah *prepare* tetapi sebelum *commit* tidak bisa dijamin konsisten.
- Analoginya untuk *certificate*, jika *certificate* telah dipublikasi tetapi *revocation* untuk *certificate* tersebut belum terlihat, maka status dari *certificate* tidak pasti, karena bisa saja *certificate* telah dibatalkan.

Akibat dari berbagai masalah tersebut, tidak ada satu standard protokol untuk manajemen *certificate*, tetapi ada beberapa protokol yang fungsinya beragam.

Salah satu standard protokol untuk manajemen *certificate* adalah standard *Lightweight Directory Access Protocol* (LDAP, lihat [ser06]). Protokol ini digunakan untuk mengakses *certificate store* (tempat penyimpanan *certificate*) yang menggunakan direktori LDAP. Suatu LDAP *server* melayani *client* dengan operasi penyimpanan *certificate* di direktori, penghapusan *certificate* dari direktori, dan pencarian *certificate* dalam direktori. Karena LDAP berorientasi pada X.500 yang terkenal rumit, implementasi LDAP tidak bisa efisien sehingga tidak bisa digunakan dalam skala besar. Untuk skala besar, *relational database* banyak digunakan untuk *certificate store*.

Selain LDAP, berbagai protokol berjenis *request/response* yang digunakan untuk manajemen *certificate* antara lain:

- CMP (*certificate management protocol*).
- OCSP (*online certificate status protocol*).
- TSP (*time-stamp protocol*).

CMP melayani *certificate requests* yaitu pembuatan *certificate* dan pembatalan (*revocation*) *certificate*. Komunikasi dengan CMP *server* dapat melalui http, *email*, atau cara lain. Informasi rinci mengenai CMP dapat dibaca di [ada05]. Karena CMP cukup rumit, sukar untuk menggunakannya dalam skala besar. Akibatnya, beberapa pembuat perangkat *networking* seperti Cisco menggunakan protokol yang lebih sederhana untuk digunakan dalam skala besar yaitu SCEP (*simple certificate enrollment protocol*).

OCSP adalah protokol untuk melayani pemeriksaan *certificate* yang berdasarkan pada concept *revocation* (informasi rinci mengenai OCSP bisa didapat di [mye99]). Jadi OCSP *server* hanya dapat menjawab bahwa *certificate* telah dibatalkan (*revoked*) atau tidak, dan informasi yang digunakan biasanya tidak *up-to-date*. Seperti telah dibahas diatas, Peter Gutmann menganggap solusi ini tidak memuaskan. Peter Gutmann menyarankan penggunaan RTCS (*real-time certificate status protocol*) sebagai *extension* dari OCSP yang bisa menjawab apakah *certificate valid* atau tidak secara *real-time*. Buku ini merekomendasikan penggunaan RTCS seperti yang disarankan oleh Peter Gutmann.

TSP adalah protokol yang digunakan untuk melayani *time-stamping requests*. Suatu *time-stamping authority* (TSA) dapat diimplementasi menggunakan TSP *server* yang melayani *time-stamping requests* dengan membuat *time-stamps* yang ditanda-tangan menggunakan kunci TSA. Informasi rinci mengenai TSP dapat dibaca di [ada01].

Kecuali untuk TSP, *server* untuk berbagai protokol *request/response* perlu menggunakan suatu *certificate store*. Tren saat ini adalah menggunakan *relational database* sebagai *back end* untuk *certificate store* agar dapat digunakan dalam skala besar.

Kita ahiri bagian ini dengan pembahasan penggunaan PKI yang berbasis X.509. PKI untuk komunitas terbuka (dimana tidak ada ikatan formal untuk anggota) biasanya tidak dapat berfungsi efektif. Ini antara lain karena:

- Keperluan dan karakteristik anggota komunitas terbuka beragam.
- Sukar untuk melakukan kontrol terhadap setiap anggota komunitas terbuka dan membuatnya bertanggung jawab atas penggunaan PKI.
- Sukar untuk mendapatkan CA yang dapat dipercaya oleh semua pihak dalam komunitas terbuka.

PKI biasanya hanya dapat berfungsi efektif jika digunakan oleh komunitas tertutup dimana ada ikatan formal untuk anggota, misalnya

- PKI untuk suatu perusahaan.
- PKI untuk suatu pemerintahan atau badan pemerintahan.
- PKI untuk komunitas jaringan bisnis tertentu misalnya jaringan bisnis suku cadang otomotif, dimana setiap anggota terdaftar secara formal dan mempunyai tanggung jawab yang jelas.
- PKI untuk VPN.

23.3 Ringkasan

Di bab ini telah dibahas *public key infrastructure* (PKI) yang pada dasarnya menambahkan fungsi *certificate management* ke fasilitas kriptografi *public key*. Dua standard PKI yang berbeda orientasi yaitu PGP dan X.509 juga dibahas. PKI yang berorientasi pada X.509 sangat sukar untuk dimanfaatkan pada komunitas yang terbuka, tetapi cocok untuk komunitas "tertutup." Sebaliknya, PGP lebih cocok untuk komunitas terbuka.

Bab 24

Aplikasi - Cryptographic Library

Cryptographic library sangat membantu untuk membuat kriptografi terintegrasi dengan aplikasi. Cara integrasi bisa *embedded* (di*compile* bersama aplikasi) atau melalui *module* yang dapat di*load* oleh aplikasi, contohnya

- *dynamically linked library* (DLL) untuk Microsoft Windows, atau
- `jar` untuk Java.

Penggunaan *cryptographic library* oleh aplikasi biasanya dilakukan melalui *application program interface* (API). Yang diberikan oleh *cryptographic library* antara lain:

- berbagai algoritma kriptografi, contohnya AES, RSA, dan DSA,
- kemampuan *public key infrastructure* (PKI), dan
- *secure sessions*, contohnya SSL/TLS.

Di bab ini kita akan bahas 3 *cryptographic library* yang populer yaitu

- OpenSSL,
- RSA BSafe, dan
- Cryptlib.

OpenSSL merupakan produk *open source*, RSA BSafe merupakan produk komersial (tetapi ada versi yang disumbangkan oleh RSA sebagai *open source*), dan Cryptlib adalah *open source* dengan dua macam lisensi (*dual license*) yaitu GPL dan komersial.

24.1 OpenSSL

OpenSSL merupakan produk *open source* dengan lisensi yang berbeda dan tidak kompatibel dengan GPL. OpenSSL dapat di*download* dari *website*

`http://www.openssl.org`.

OpenSSL terdiri dari 3 komponen utama yaitu:

- SSL/TLS *library*,
- Crypto *library*, dan
- `openssl` *command line tool*.

SSL/TLS *library* memberikan fasilitas untuk melakukan sesi SSL/TLS dalam bentuk *loadable module* (DLL untuk Microsoft Windows). Fungsi API untuk SSL/TLS *library* dibagi menjadi 5 golongan:

- *Protocol methods*, untuk menentukan versi SSL/TLS dan jenis layanan (*client*, *server*, atau *client-server*).
- *Ciphers*, untuk menentukan berbagai jenis enkripsi yang dapat digunakan.
- *Protocol contexts*, untuk menentukan *context* secara global selama program berjalan.
- *Sessions*, untuk menentukan berbagai parameter per sesi.
- *Connections*, untuk koneksi. Ini adalah fungsi utama SSL/TLS *library*, dan selama program berjalan fungsi dari golongan ini yang banyak digunakan.

Crypto library adalah implementasi dari berbagai algoritma enkripsi dan *hashing*. Selain digunakan oleh SSL/TLS *library* dan openssl *command line tool*, *crypto library* juga digunakan oleh OpenSSH, GnuPG dan implementasi standard kriptografi lainnya.

Command line tool `openssl` adalah program yang berorientasi Unix yang dapat digunakan untuk:

- Pembuatan dan manajemen kunci privat/publik.
- Operasi kriptografi *public key* termasuk *key management*, *digital signing* dan *digital signature checking*.
- Pembuatan dan manajemen *certificate* X.509.
- Kalkulasi *digest*.

- Enkripsi dan dekripsi.
- Testing SSL/TLS *client* dan *server*.
- S/MIME.
- Membuat *time-stamp request*, membuat *time-stamp* dan melakukan verifikasi *time-stamp*.

24.2 RSA BSafe

RSA BSafe adalah produk komersial dari RSA Security Inc., suatu divisi dari EMC Corporation. Selain versi komersial, RSA Security juga membuat versi *open source* untuk *platform* Microsoft Windows, Solaris dan Linux:

- RSA BSafe Share for C++, dan
- RSA BSafe Share for Java.

Perbedaan utama versi komersial dan versi *share* adalah versi komersial termasuk sertifikasi FIPS-140, mendukung PKCS#11 (perangkat kripto) dan mendukung *platform* HP-UX, AIX, *mainframe* dan *embedded* disamping mendukung Microsoft Windows, Solaris dan Linux. Versi *share* tidak termasuk sertifikasi, tidak mendukung PKCS#11, dan hanya mendukung *platform* Microsoft Windows, Solaris dan Linux. Operasi yang didukung oleh RSA BSafe Share antara lain:

- Operasi kriptografi, termasuk enkripsi/dekripsi, pembuatan *message digests*, pembuatan *message authentication codes*, pembuatan dan verifikasi *digital signatures* dan pembuatan *pseudo-random numbers*.
- Operasi kunci, termasuk pembuatan pasangan kunci privat dan kunci publik, melakukan Diffie-Hellman *key agreement*, dan *encoding/decoding asymmetric keys*.
- Operasi *certificate*, termasuk pembuatan *self-signed certificates*, pembuatan *certificate requests*, verifikasi *certificate chains*, pembuatan *certificate revocation list*, melakukan operasi protokol OCSP, dan melakukan operasi *certificate stores*.
- Operasi PKCS#7 (*cryptographic message syntax*).
- Operasi SSL/TLS.

RSA BSafe Share for C++ bersifat *toolkit* yang menyediakan:

- *header files*,

- *object file libraries*, dan
- *code samples*,

untuk digunakan dengan Microsoft Visual Studio C/C++ (khusus untuk RSA BSafe Share for C++ versi Microsoft Windows). RSA BSafe for Java juga memberikan fasilitas serupa untuk Java dengan 2 macam API:

- implementasi standard Java JCE API dan
- implementasi standard Java JSSE API.

RSA BSafe Share dapat di*download* lengkap dengan dokumentasi di:

`https://community.emc.com/`.

24.3 Cryptlib

Cryptlib adalah suatu sistem yang dikembangkan oleh Peter Gutmann berdasarkan disertasinya mengenai *cryptographic security architecture*. Disertasi tersebut diselesaikannya tahun 2000, dan telah direvisi dan dijadikan buku yang diterbitkan tahun 2004 (lihat [gut04]). Disertasi asli juga dapat di*download* dari *website* University of Auckland.

Selain mempunyai *security architecture* yang dirancang menurut prinsip keamanan tingkat tinggi, Cryptlib juga mempunyai *software architecture* yang sangat baik. Alhasil penggunaannya relatif mudah, lebih mudah dibandingkan OpenSSL maupun RSA BSafe. Oleh sebab itu, buku ini sangat merekomendasikan penggunaan Cryptlib dan akan membahas Cryptlib secara lebih rinci dibandingkan OpenSSL dan RSA BSafe.

Cryptlib diprogram dalam C/C++, namun, menggunakan *language binding*, dapat diakses dari program yang ditulis dalam:

- C/C++,
- C#/.NET,
- Delphi,
- Java,
- Python,
- Tcl, atau
- Visual Basic.

Khusus untuk Python atau Tcl, *language binding* harus dibuat menggunakan fasilitas *extension* dari sistem Python atau Tcl. Sebagai contoh, menggunakan platform Microsoft Windows, Cryptlib dapat di*build* menggunakan Microsoft Visual Studio 2008, dan menggunakan ActiveState ActivePython 2.6, membuat extension dengan melakukan

```
python setup.py install
```

dari *command prompt*, setelah terlebih dahulu melakukan

```
cd cl333/bindings
```

dimana `cl333` adalah *main folder* untuk Cryptlib. Yang perlu diperhatikan adalah versi *compiler* C/C++ yang digunakan untuk *build* Python harus sama dengan versi yang digunakan untuk *build* Cryptlib (ActiveState ActivePython 2.6 untuk Microsoft Windows di*build* menggunakan Microsoft Visual Studio 2008). Program yang menggunakan Cryptlib harus melakukan inisialisasi sebelum memanggil fungsi Cryptlib lainnya dan harus melakukan finalisasi setelah selesai dengan Cryptlib. Program C/C++ yang menggunakan Cryptlib mempunyai format sebagai berikut:

```
#include "cryptlib.h"

cryptInit();

/* Isi program yang menggunakan Cryptlib */

cryptEnd();
```

Untuk Python, format adalah sebagai berikut:

```
from cryptlib_py import *

cryptInit()

# Isi program yang menggunakan Cryptlib

cryptEnd()
```

Cryptlib diimplementasi dengan cara yang sangat *object-oriented*. API untuk Cryptlib memberikan fasilitas untuk menggunakan Cryptlib pada 3 tingkatan antarmuka:

- *high-level interface*, yang memanipulasi *container objects* berupa *sessions*, *envelopes*, dan *certificates*,
- *mid-level interface*, yang memanipulasi *action objects* berupa *encryption contexts* dan *container objects* berupa *keysets*, dan
- *low-level interface* untuk kustomisasi layanan pendukung.

Pada tingkat *high-level interface*, Cryptlib memberikan layanan untuk *security services* yang lengkap. Untuk *secure enveloping*, antarmuka ini memberikan fasilitas untuk:

- *secure* CMS *enveloping*,
- *secure* S/MIME *enveloping*, dan
- *secure* PGP/OpenPGP *enveloping*.

Untuk *secure session*, *high-level interface* memberikan layanan yang mudah digunakan, baik sebagai *client* maupun sebagai *server*, untuk jenis sesi:

- SSL/TLS, dan
- SSH.

Antarmuka untuk tingkat ini juga memberikan layanan yang mudah digunakan untuk CA *services*, termasuk sebagai *client* atau *server* untuk berbagai protokol berikut:

- CMP,
- SCEP,
- OCSP, dan
- RTCS.

Cryptlib bahkan dapat digunakan sebagai *plug-and-play* PKI. Selain untuk CA *services*, antarmuka tingkat ini juga mendukung protokol untuk *time-stamps* yaitu TSP. Mayoritas pengguna akan menggunakan *high-level interface*. Untuk yang tidak terlalu paham kriptografi secara rinci disarankan untuk hanya menggunakan *high-level interface*. Sebagai contoh penggunaan *high-level interface*, berikut adalah *code snippet* untuk melakukan S/MIME *encrypted enveloping* menggunakan kunci yang terdapat dalam suatu X.509 *certificate*:

```
CRYPT_ENVELOPE cryptEnvelope;
int bytesCopied;
```

```
cryptCreateEnvelope(&cryptEnvelope, cryptUser,
  CRYPT_FORMAT_SMIME);

/* Tambahkan certificate ke envelope */
cryptSetAttribute(cryptEnvelope, CRYPT_ENVINFO_PUBLICKEY,
  certificate);

/* Tambahkan informasi mengenai besarnya data ke envelope */
cryptSetAttribute(cryptEnvelope, CRYPT_ENVINFO_DATASIZE,
  messageLength);

/* Masukkan data ke envelope, lakukan proses (enkripsi),
   lalu keluarkan data yang telah diproses. */
cryptPushData(cryptEnvelope, message, messageLength,
  &bytesCopied);
cryptFlushData(cryptEnvelope);
cryptPopData(cryptEnvelope, envelopedData,
  envelopedDataBufferSize, &bytesCopied);

cryptDestroyEnvelope(cryptEnvelope);
```

Sedikit penjelasan mengenai *code snippet*:

- Data dimasukkan kedalam *envelope* menggunakan `cryptPushData`. Hasil untuk parameter `&bytesCopied` biasanya sama dengan `messageLength`, tetapi ada kalanya beda (misalnya tidak semua data dapat masuk karena sudah penuh).
- `cryptFlushData` digunakan untuk *processing*. Cryptlib mengetahui apa yang harus dikerjakan dalam *processing* berdasarkan nilai berbagai atribut pada *envelope*.
- `cryptPopData` digunakan untuk mengeluarkan data hasil *processing* dari *envelope*.

Sebagai contoh penggunaan *high-level interface* untuk *secure session*, berikut adalah *code snippet* untuk memulai sesi SSL/TLS untuk *client*:

```
CRYPT_SESSION cryptSession;

cryptCreateSession(&cryptSession, cryptUser,
  CRYPT_SESSION_SSL);
```

```
cryptSetAttributeString(cryptSession,
  CRYPT_SESSINFO_SERVER_NAME, serverName, serverNameLength);
cryptSetAttribute(cryptSession, CRYPT_SESSINFO_ACTIVE,1);
```

Berikut adalah *code snippet* yang menggunakan kemampuan *plug-and-play* PKI dari *high-level interface*:

```
CRYPT_SESSION cryptSession;

/* Buat sesi CMP dan tambahkan name/address server */
cryptCreateSession(&cryptSession, cryptUser,
  CRYPT_SESSION_CMP);
cryptSetAttributeString(cryptSession, CRYPT_SESSINFO_SERVER,
  server, serverLength);

/* Tambahkan username, password dan smartcard */
cryptSetAttributeString(cryptSession, CRYPT_SESSINFO_USERNAME,
  userName, userNameLength);
cryptSetAttributeString(cryptSession, CRYPT_SESSINFO_PASSWORD,
  password, passwordLength);
cryptSetAttribute(cryptSession, CRYPT_SESSINFO_CMP_PRIVKEYSET,
  cryptDevice);

/* Aktivasikan sesi */
cryptSetAttribute(cryptSession, CRYPT_SESSINFO_ACTIVE, TRUE);
```

Yang dilakukan oleh *code snippet* diatas adalah:

- Menggunakan *smart card* untuk membuat kunci untuk *signing* dan kunci untuk enkripsi (dua pasang kunci).
- Meminta *certificate* untuk kunci *signing* dari CA.
- Menggunakan *certificate* kunci *signing* untuk meminta *certificate* kunci enkripsi dan *certificate* lainnya dari CA.
- Menyimpan semua kunci dan *certificate* yang dihasilkan dalam *smartcard*.

Code snippet diatas menunjukkan bahwa pada tingkat *high-level interface* hanya dibutuhkan beberapa instruksi untuk melakukan banyak tugas.

Pada tingkat *mid-level interface*, pengguna dapat melakukan operasi agak lebih rinci seperti:

- *key generation*,
- *key management*,
- operasi enkripsi dan *digest*,
- *key exchange*, dan
- operasi *digital signature*.

Semua operasi pada tingkat *mid-level interface* melibatkan *encryption context*. Operasi *key management* juga dapat melibatkan *container object* jenis *keyset*. Berikut adalah *code snippet* yang memberi contoh *key generation* pasangan kunci RSA 2048 bit ke *encryption context*, dilanjutkan dengan penyimpanan kunci privat ke *keyset* berupa *file* dengan format PKCS#15:

```
CRYPT_CONTEXT privKeyContext;
CRYPT_KEYSET cryptKeyset;

cryptCreateContext(&privKeyContext, cryptUser, CRYPT_ALGO_RSA);
cryptSetAttributeString(privKeyContext, CRYPT_CTXINFO_LABEL,
  label, labelLength);
cryptSetAttribute(privKeyContext, CRYPT_CTXINFO_KEYSIZE,
  2048/8);

cryptGenerateKey(privKeyContext);

cryptKeysetOpen(&cyptKeyset, cryptUser, CRYPT_KEYSET_FILE,
  "/home/kelsey/keys.p15", CRYPT_KEYOPT_NONE);
cryptAddPrivateKey(cryptKeyset, privKeyContext, password);
```

Dalam *code snippet* diatas, `label` digunakan karena diperlukan saat *retrieval* dari *keyset* untuk identifikasi. Jenis *keyset* adalah *file*, yang diindikasikan menggunakan konstan `CRYPT_KEYSET_FILE`. Untuk skala besar, penggunaan *database* (RDBMS atau RDBMS dengan ODBC) disarankan, terutama untuk keperluan CA. Berikut adalah berbagai jenis *keyset* dalam Cryptlib:

Jenis Keyset	Penjelasan
CRYPT_KEYSET_FILE	*File* PKCS#15 atau PGP *ring*.
CRYPT_KEYSET_HTTP	URL untuk lokasi *certificate*/CRL.
CRYPT_KEYSET_LDAP	Direktori LDAP.
CRYPT_KEYSET_DATABASE	RDBMS.
CRYPT_KEYSET_ODBC	ODBC RDBMS.
CRYPT_KEYSET_PLUGIN	RDBMS lewat *database network plugin interface*.
CRYPT_KEYSET_DATABASE_STORE	RDBMS untuk CA.
CRYPT_KEYSET_ODBC_STORE	ODBC RDBMS untuk CA.
CRYPT_KEYSET_PLUGIN_STORE	RDBMS lewat *database network plugin interface* untuk CA.

Jika pada tingkat *high-level interface* enkripsi dilakukan pada data dalam *container object*, pada tingkat *mid-level interface* enkripsi dilakukan langsung pada buffer *in place*, contohnya seperti dalam *code snippet* berikut:

```
cryptEncrypt(cryptContext, buffer, length);
```

Untuk *key exchange* kunci simetris, `cryptExportKey` dan `cryptImportKey` digunakan. Jika kunci simetris dibuat oleh satu pihak, pembuat kunci simetris melakukan `cryptExportKey` dan penerima melakukan `cryptImportKey`. *Code snippet* berikut adalah contoh untuk pembuat kunci simetris, dimana kunci simetris dienkripsi menggunakan kunci publik penerima (hasilnya berada dalam *buffer* dengan *pointer* `encryptedKey`):

```
CRYPT_CONTEXT pubKeyContext, cryptContext;
void *encryptedKey;
int encryptedKeyLength;

cryptCreateContext(&cryptContext, cryptUser, CRYPT_ALGO_AES);
```

```
cryptGenerateKey(cryptContext);

encryptedKey = malloc(encryptedKeyMaxLength);

cryptExportKey(encryptedKey, encryptedKeyMaxLength,
  &encryptedKeyLength, pubKeyContext, cryptContext);
```

Digital signing pada tingkat *mid-level interface* dilakukan menggunakan fungsi `cryptCreateSignature` sedangkan fungsi `cryptCheckSignature` digunakan untuk verifikasi. Namun fungsi `cryptCreateSignature` hanya melakukan bagian enkripsi, jadi *hashing* harus dilakukan terlebih dahulu. Berikut adalah *code snippet* untuk *digital signing*:

```
CRYPT_CONTEXT sigKeyContext, hashContext;
void *signature;
int signatureLength;

cryptCreateContext(&hashContext, cryptUser, CRYPT_ALGO_SHA);

cryptEncrypt(hashContext, data, dataLength);
cryptEncrypt(hashContext, data, 0);

signature = malloc(signatureMaxLength);

cryptCreateSignature(signature, signatureMaxLength,
  &signatureLength, sigKeyContext, hashContext);
cryptDestroyContext(hashContext);
```

Dan berikut adalah *code snippet* untuk verifikasi *digital signature*, dimana jika verifikasi gagal, akan menghasilkan *error* `CRYPT_ERROR_SIGNATURE`:

```
CRYPT_CONTEXT sigCheckContext, hashContext;

cryptCreateContext(&hashContext, cryptUser, CRYPT_ALGO_SHA);

cryptEncrypt(hashContext, data, dataLength);
cryptEncrypt(hashContext, data, 0);

cryptCheckSignature(signature, signatureLength,
  sigCheckContext, hashContext);
```

```
cryptDestroyContext(hashContext);
```

Pada tingkat *low-level interface*, berbagai implementasi algoritma enkripsi dan *authentication* dapat digunakan, misalnya untuk implementasi protokol kriptografi yang tidak standard. Namun untuk itu pengguna harus menguasai berbagai konsep dan algoritma kriptografi secara rinci. Tingkat *low-level interface* juga memberikan fasilitas untuk:

- kustomisasi *database plugin*,
- kustomisasi *network plugin*, atau
- kustomisasi *crypto plugin*.

Untuk dapat bertransaksi dengan berbagai *database* yang digunakan sebagai *certificate store*, Cryptlib memberi fasilitas *database plugin interface* yang dapat dikustomisasi sesuai dengan jenis *database* yang digunakan. Ada 5 fungsi yang harus dibuat untuk suatu *database plugin* yaitu:

- `openDatabase`, untuk memulai sesi dengan *database*,
- `closeDatabase`, untuk mengahiri sesi dengan *database*,
- `performUpdate`, untuk mengirim data ke *database* menggunakan instruksi SQL,
- `performQuery`, untuk mendapatkan data dari *database* menggunakan instruksi SQL, dan
- `performErrorQuery`, untuk mendapatkan informasi mengenai *error* yang terjadi dari *database*.

Untuk dapat menggunakan berbagai protokol komunikasi, Cryptlib memberi fasilitas *network plugin interface* yang dapat dikustomisasi sesuai dengan protokol. Ada 5 fungsi yang harus dibuat untuk suatu *network plugin* yaitu:

- `transportOKFunction`, untuk mengecek status dari *transport layer*,
- `transportConnectFunction`, untuk memulai koneksi dengan *server* atau *remote client*,
- `transportDisconnectFunction`, untuk mengahiri koneksi dengan *server* atau *remote client*,
- `transportReadFunction`, untuk mendapatkan data dari *server* atau *remote client*, dan

- `transportWriteFunction`, untuk mengirim data ke *server* atau *remote client*.

Untuk dapat menambahkan atau mengganti berbagai kapabilitas enkripsi dengan implementasi baru, Cryptlib memberi fasilitas *crypto plugin interface*. Berbeda dengan *database* dan *network plugin interface*, *crypto plugin interface* memberi akses ke antarmuka kapabilitas enkripsi internal dari Cryptlib. Implementasi baru dapat menjadi bagian dari program, atau dapat berupa implementasi eksternal misalnya menggunakan perangkat kripto. Jika implementasi baru menggantikan implementasi Cryptlib, implementasi lama harus di*unplug* dan implementasi baru di*plugin*.

Cryptlib dan dokumentasi dapat di*download* dari

`http://www.cs.auckland.ac.nz/~pgut001/cryptlib/`

atau

`http://www.cryptlib.com`.

24.4 Ringkasan

Di bab ini telah dibahas 3 *cryptographic library* yang populer yaitu OpenSSL, RSA Bsafe dan Cryptlib. OpenSSL bersifat *open source*, RSA BSafe bersifat komersial meskipun ada versi *open source*, sedangkan Cryptlib adalah *open source* dengan pilihan lisensi GPL atau komersial. Karena Cryptlib merupakan *library* yang terlengkap dan mudah digunakan, buku ini merekomendasikan Cryptlib dan membahasnya lebih rinci dibandingkan OpenSSL dan RSA BSafe.

Bab 25

Analisa Protokol Kriptografi

Merancang protokol kriptografi bukan sesuatu yang mudah, bahkan untuk para ahli. Sebagai contoh, Needham-Schroeder *public key protocol* rentan terhadap *man-in-the-middle attack*. Needham-Schroeder *symmetic-key protocol*, protokol yang digunakan oleh Kerberos versi pertama, juga rentan terhadap *replay attack* jika kunci sesi bocor, sehingga perlu dimodifikasi dengan mekanisme *timestamp*. Kelemahan dari suatu protokol kriptografi bisa digolongkan:

- kelemahan algoritma enkripsi/*hashing* atau
- kelemahan logika dari protokol.

Kelemahan algoritma enkripsi/*hashing* dapat dianalisa menggunakan *cryptanalysis*. Analisa protokol kriptografi fokus pada analisa logika dari protokol dengan mengasumsi (untuk sementara) bahwa tidak ada kelemahan dalam algoritma enkripsi/*hashing*.

Analisa protokol kriptografi memerlukan, sebagai dasar, suatu teori logika mengenai berbagai jenis komponen informasi seperti kunci dan data, berbagai operasi terhadap komponen informasi seperti enkripsi data menggunakan kunci, dan informasi apa yang bisa didapat oleh seseorang dari suatu himpunan komponen informasi. Protokol kemudian dirumuskan terdiri dari berbagai operasi dengan urutan tertentu dan melibatkan berbagai aktor dan komponen informasi. Analisa dilakukan untuk melihat efek dari urutan operasi terhadap pengetahuan berbagai aktor mengenai informasi.

Untuk analisa protokol kriptografi, ada dua jenis logika yang kerap digunakan oleh para peneliti, yaitu:

- *modal logic* seperti BAN (Burrows-Abadi-Needham) *logic*, atau

- *classical logic*.

Analisa biasanya dilakukan dengan bantuan alat:

- Untuk *modal logic*, analisa biasanya dibantu dengan *model checker*.
- Untuk *classical logic*, analisa biasanya dibantu dengan *theorem prover*.

Cara *model checker* bekerja adalah dengan mengecek semua kemungkinan *state*, sedangkan *theorem prover* lebih bersifat manipulasi simbol.

Suatu hal yang kurang memuaskan dengan BAN *logic* (lihat [bur90]) adalah *formal semantics* untuk konsep *freshness* yang menjadi bagian dari logika, tidak jelas. Jadi tidak jelas apa arti sesungguhnya dari konsep tersebut. Menggunakan *classical logic*, konsep *freshness* didefinisikan secara langsung, jadi bisa lebih jelas apa yang dimaksud. Oleh sebab itu kita akan fokus pada penggunaan *classical logic* dalam pembahasan lebih lanjut. Penggunaan *classical logic* juga dilakukan oleh Paulson (lihat [pau98]) dan Bolignano (lihat [bol96]).

Pertama, kita akan bahas bagaimana kita dapat membuat suatu teori menggunakan *classical logic* yang meliputi:

- komponen informasi,
- konstruksi menggunakan komponen informasi yang menghasilkan komponen informasi yang lebih besar, dan
- informasi apa yang bisa didapat dari suatu himpunan komponen informasi.

Kita namakan teori tersebut *message theory* dimana suatu *message* merupakan informasi yang terstruktur berdasarkan komponen, atau singkatnya, *message* merupakan komponen informasi. Jadi *message* merupakan apa yang dinamakan *abstract data type*, tepatnya suatu *recursive disjoint union*. Suatu *recursive disjoint union* adalah suatu *type* yang merupakan *union* dari beberapa *subtype* yang *disjoint*. Diantaranya, ada *subtype* yang bersifat *recursive*, yaitu ada nilai *subtype* yang merupakan konstruksi dimana ada subkomponen langsung atau tidak langsung yang mempunyai *subtype* yang sama, bahkan nilai *subtype* bisa mempunyai subkomponen dengan *type message*. Beberapa *subtype* dari *message* bersifat atomik (tidak terdiri dari subkomponen). Ada 6 *subtype* yang bersifat atomik yaitu:

- *text*, untuk naskah,
- *principal*, untuk identitas,
- *nonce*, untuk nilai acak, dimana setiap nilai hanya digunakan sekali dalam aplikasi protokol kriptografi,

- *symmetric key* (*symkey*),
- *private key* (*privkey*), dan
- *public key* (*pubkey*).

Kita definisikan *key* sebagai:

$$key = symkey \cup privkey \cup pubkey$$

dan *atomic* sebagai:

$$atomic = text \cup principal \cup nonce \cup key.$$

Ada 3 *subtype* non-atomik yaitu:

- *encryption*, dengan konstruktor $encrypt(m, k)$, dimana $m \in message$ dan $k \in key$,
- *aggregation*, konstruktornya $combine(m_1, m_2)$, dimana $m_1 \in message$ dan $m_2 \in message$ (karena *combine* bersifat *associative*, kita dapat gunakan $combine(a, b, c)$ untuk $combine(combine(a, b), c)$), dan
- *hashed*, dengan konstruktor $hash(m)$, dimana $m \in message$.

Jadi *message* adalah:

$$message = atomic \cup encryption \cup aggregation \cup hashed.$$

Konstruktor *hash* bersifat *injective*, jadi kita melakukan idealisasi dimana *hash* bersifat *collision-free*. Dalam teori ini terdapat juga fungsi

$$inversekey : key \longrightarrow key$$

dimana

$$\begin{array}{ll} inversekey(k) = k & \text{jika } k \in symkey, \\ inversekey(k) \in pubkey & \text{jika } k \in privkey, \\ inversekey(k) \in privkey & \text{jika } k \in pubkey. \end{array}$$

Lagi kita lakukan idealisasi dengan membuat *inversekey* bersifat *injective*. Konsep penting dalam teori *message* menggunakan fungsi atau predikat *known*:

$$known : message \times messageSet \longrightarrow boolean$$

dimana

$$messageSet = \{m | m \in message\}.$$

Predikat $known(m, s)$ dapat diinterpretasikan sebagai: *message* m dapat diketahui jika setiap *message* dalam s diketahui. Untuk singkatnya kita katakan

bahwa *message* m diketahui dari himpunan s. Ada beberapa *axiom* mengenai *known* dalam teori *message*. *Axiom* pertama adalah mengenai pengetahuan langsung:

$$\begin{aligned} &\forall m \in message, s \in messageSet : \\ &\quad m \in s \Longrightarrow known(m, s) \end{aligned} \tag{25.1}$$

Axiom kedua adalah mengenai *transitivity*:

$$\begin{aligned} &\forall m \in message, s_1, s_2 \in messageSet : \\ &\quad (known(m, s_1) \wedge \forall c \in s_1 : known(c, s_2)) \Longrightarrow known(m, s_2) \end{aligned} \tag{25.2}$$

Axiom 25.2 mengatakan bahwa jika *message* m diketahui dari himpunan s_1 dan setiap elemen dari s_1 diketahui dari himpunan s_2, maka m diketahui dari himpunan s_2. *Axiom* berikut mengatakan *known* bersifat monotonik berdasarkan himpunan.

$$\begin{aligned} &\forall m \in message, s_1, s_2 \in messageSet : \\ &\quad (known(m, s_1) \wedge s_1 \subseteq s_2) \Longrightarrow known(m, s_2) \end{aligned} \tag{25.3}$$

Tiga *axiom* berikutnya mengatakan bahwa *message* yang merupakan hasil konstruksi komponen-komponen yang diketahui juga diketahui.

$$\begin{aligned} &\forall m \in message, k \in key, s \in messageSet : \\ &\quad (known(m, s) \wedge known(k, s)) \Longrightarrow known(encrypt(m, k), s) \end{aligned} \tag{25.4}$$

$$\begin{aligned} &\forall m_1, m_2 \in message, s \in messageSet : \\ &\quad (known(m_1, s) \wedge known(m_2, s)) \Longrightarrow known(combine(m_1, m_2), s) \end{aligned} \tag{25.5}$$

$$\begin{aligned} &\forall m \in message, s \in messageSet : \\ &\quad known(m, s) \Longrightarrow known(hash(m), s) \end{aligned} \tag{25.6}$$

Dua *axiom* berikutnya masing-masing menunjukkan bagaimana caranya untuk mendapatkan komponen *message* dari *encrypt* dan kedua komponen dari *combine*.

$$\begin{aligned} &\forall m \in message, k \in key, s \in messageSet : \\ &\quad (known(encrypt(m, k), s) \wedge known(inversekey(k), s)) \Longrightarrow \\ &\qquad known(m, s) \end{aligned} \tag{25.7}$$

$$\begin{aligned} &\forall m_1, m_2 \in message, s \in messageSet : \\ &\quad known(combine(m_1, m_2), s) \Longrightarrow (known(m_1, s) \wedge known(m_2, s)) \end{aligned} \tag{25.8}$$

Untuk mengetahui komponen *message* dari *encrypt*, *inversekey* dari kunci perlu diketahui. Untuk *combine*, kedua komponen bisa didapat langsung.

Selanjutnya kita definisikan fungsi *parts* yang akan digunakan dalam definisi konsep *fresh*.

$$parts : messageSet \longrightarrow messageSet.$$

Fungsi *parts* menambahkan ke himpunan, semua komponen dari setiap *message* dalam himpunan, semua subkomponen dari setiap komponen, dan seterusnya. *Message* yang atomik tidak mempunyai komponen:

$$\begin{aligned} &\forall m \in atomic, s \in messageSet : \\ &\quad parts(\{m\} \cup s) = \{m\} \cup parts(s). \end{aligned} \tag{25.9}$$

Untuk *encrypt*, *combine* dan *hash*, kita tambahkan komponen ke *parts*.

$$\begin{aligned} &\forall m \in message, k \in key, s \in messageSet : \\ &\quad parts(\{encrypt(m,k)\} \cup s) = \\ &\qquad \{encrypt(m,k)\} \cup parts(\{m,k\} \cup s). \end{aligned} \tag{25.10}$$

$$\begin{aligned} &\forall m_1, m_2 \in message, s \in messageSet : \\ &\quad parts(\{combine(m_1,m_2)\} \cup s) = \\ &\qquad \{combine(m_1,m_2)\} \cup parts(\{m_1,m_2\} \cup s). \end{aligned} \tag{25.11}$$

$$\begin{aligned} &\forall m \in message, s \in messageSet : \\ &\quad parts(\{hash(m)\} \cup s) = \{hash(m)\} \cup parts(\{m\} \cup s). \end{aligned} \tag{25.12}$$

Selesailah pembuatan *message theory*.

Berikutnya, kita kembangkan mekanisme untuk melakukan simulasi protokol berupa *state machine*. *State* dari *state machine* terdiri dari:

- himpunan semua *message* yang telah terlihat, kita namakan *seen*,
- himpunan semua *message* yang telah diterima atau dibuat oleh setiap *principal*, kita namakan *storage*, dan
- sejarah dari *events* yang telah terjadi, kita namakan *history*.

Komponen *seen* merupakan himpunan *message* dan merepresentasikan *medium* komunikasi yang terbuka (dapat disadap dan dapat diinjeksi). Komponen *storage* merupakan kumpulan dari himpunan *message* yang diindeks dengan *principal*. Jadi $storage[a]$ adalah himpunan *message* yang telah dibuat atau diterima oleh *principal* a. Untuk memudahkan analisa, kita melakukan idealisasi dengan mengumpamakan bahwa setiap kunci publik dan setiap *principal* tidak perlu dirahasiakan, tetapi merupakan informasi publik.

$$public = pubKey \cup principal.$$

Kita definisikan konsep $forgeable(p,m)$ (*principal* p dapat membuat *message* m) sebagai berikut:

$$forgeable(p,m) = known(m, storage[p] \cup seen \cup public).$$

Selain *forgeable* kita perlu definisikan konsep *freshness*:

$$fresh(m) = m \notin parts((\bigcup_p storage[p]) \cup seen \cup public).$$

Komponen *history* adalah deretan *event* yang telah terjadi. Suatu langkah protokol adalah suatu *event* yang bisa berupa:

- $send(s, m)$, pengiriman *message* m oleh *principal* s,
- $receive(r, m)$, penerimaan *message* m oleh *principal* r,
- $outOfBand(s, r, m)$, *transfer message* m dari *principal* s ke *principal* r, melalui jalur khusus yang aman,
- $generate(p, m)$, pembuatan *message* m yang *atomic* oleh *principal* p,
- $construct(p, m)$, konstruksi *message* m oleh *principal* p, dan
- $intruder(p, m)$, injeksi *message* m oleh *principal* p.

Suatu simulasi protokol adalah evolusi dari *state machine* dimana setiap langkah mempunyai dua macam persyaratan:

- persyaratan *state machine*, contohnya untuk *send*, *message* yang dikirimkan harus ada dalam *storage principal*, dan
- persyaratan protokol, biasanya berupa persyaratan bahwa *event* dengan atribut tertentu telah ada dalam *history*.

Untuk setiap langkah, *event* untuk langkah tersebut ditambahkan ke deretan *event* dalam *history*. Persyaratan *state machine* dan efek dari suatu langkah terhadap *state* (selain penambahan *event* ke *history*) adalah sebagai berikut:

Langkah	**Persyaratan**	**Efek**
$send(s, m)$	$m \in storage[s]$	$seen \leftarrow seen \cup \{m\}$
$receive(r, m)$	$m \in seen$	$storage[r] \leftarrow storage[r] \cup \{m\}$
$outOfBand(s, r, m)$	$m \in storage[s]$	$storage[r] \leftarrow storage[r] \cup \{m\}$
$generate(p, m)$	$m \in atomic$ $fresh(m)$	$storage[p] \leftarrow storage[p] \cup \{m\}$
$construct(p, m)$	$forgeable(p, m)$	$storage[p] \leftarrow storage[p] \cup \{m\}$
$intruder(p, m)$	$m \in storage[s]$	$seen \leftarrow seen \cup \{m\}$

Beberapa hal yang perlu diperhatikan adalah:

- Langkah *generate* dimaksudkan untuk membuat *nonce*, *text*, *symKey* atau *privKey* (*atomic message* yang bisa dirahasiakan). Karena *pubKey* dan *principal* keduanya tidak *fresh*, *generate* tidak bisa digunakan untuk membuat kunci publik atau *principal*.
- Langkah *intruder* sebetulnya sama dengan langkah *send*. Perbedaan hanya pada penggunaan, langkah *intruder* tidak mempunyai persyaratan protokol, jadi dapat terjadi kapan saja, asalkan persyaratan *state machine* terpenuhi.
- Langkah *construct* dimaksudkan untuk membuat *message* yang akan dikirim menggunakan *send*, *outOfBand* atau *intruder*.
- Untuk keperluan analisa protokol, setiap langkah dapat diberi *label* dan *id* agar persyaratan protokol dapat dirumuskan.

Mari kita gunakan Needham-Schroeder *symmetric key protocol* sebagai contoh untuk melihat bagaimana mekanisme yang telah dibuat dapat digunakan untuk menganalisa protokol. Secara garis besar, menggunakan protokol ini, A ingin berkomunikasi dengan B secara aman dengan bantuan *server* S. Pada awalnya, A, B dan S mengetahui beberapa hal:

- A dan S mengetahui kunci simetris K_{AS}. Selain A dan S, tidak ada yang mengetahui K_{AS}.
- B dan S mengetahui kunci simetris K_{BS}. Selain B dan S, tidak ada yang mengetahui K_{BS}.

Langkah pertama dalam protokol adalah

$$A \longrightarrow S : A, B, N_A$$

dimana A mengidentifikasi dirinya dan B, dan N_A adalah suatu *nonce* yang *fresh*. Langkah kedua adalah

$$S \longrightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$

dimana K_{AB} adalah kunci simetris *fresh* yang dibuat oleh S untuk A dan B. A mendekripsi $\{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$ untuk mendapatkan K_{AB} dan $\{K_{AB}, A\}_{K_{BS}}$. Langkah ketiga adalah

$$A \longrightarrow B : \{K_{AB}, A\}_{K_{BS}}$$

dimana $\{K_{AB}, A\}_{K_{BS}}$ dapat didekripsi oleh B untuk mendapatkan K_{AB} dan A. Langkah keempat adalah

$$B \longrightarrow A : \{N_B\}_{K_{AB}}$$

dimana N_B adalah *nonce* yang *fresh*. Langkah terahir adalah

$$A \longrightarrow B : \{N_B - 1\}_{K_{AB}}$$

sebagai konfirmasi dari A. Selanjutnya A dan B dapat berkomunikasi menggunakan kunci simetris K_{AB} untuk mengenkripsi komunikasi.

Mari kita coba formalisasikan Needham-Schroeder *symmetric key protocol* menggunakan mekanisme yang telah kita buat. Untuk *initial state*, pengetahuan mengenai kunci dapat kita formalkan sebagai berikut:

$$\begin{array}{l} K_{AS}, K_{BS} \in symKey, \\ K_{AS} \in storage[A], \\ K_{BS} \in storage[B], \\ K_{AS}, K_{BS} \in storage[S], \\ K_{AS}, K_{BS} \notin seen, \\ \forall p \in principal : K_{AS} \in storage[p] \Longrightarrow p = A \vee p = S, \\ \forall p \in principal : K_{AB} \in storage[p] \Longrightarrow p = B \vee p = S. \end{array}$$

Langkah pertama kita bagi menjadi empat langkah: 1_a, 1_b, 1_c dan 1_d. Langkah 1_a adalah

$$generate(a, n_a, 1_a, id)$$

dengan syarat belum ada sesi dengan identifikasi id, dimana n_a adalah *nonce* yang dibuat, 1_a adalah *label*, dan id adalah identifikasi untuk sesi (unik untuk setiap sesi). Langkah 1_b adalah

$$construct(a, combine(a, b, n_a), 1_b, id)$$

dengan persyaratan:

- $generate(a, n_a, 1_a, id)$ ada dalam *history*.

Langkah 1_c adalah

$$send(a, combine(a, b, n_a), 1_c, id)$$

dengan persyaratan:

- $construct(a, combine(a, b, n_a), 1_b, id)$ ada dalam *history*.
- $send(a, combine(a, b, n_a), 1_c, id)$ belum ada dalam *history*.

Langkah 1_d adalah

$$receive(S, combine(a, b, n_a), 1_d, id)$$

dengan persyaratan:

- $send(a, combine(a, b, n_a), 1_c, id)$ ada dalam *history*.
- $receive(S, combine(a, b, n_a), 1_d, id)$ belum ada dalam *history*.

Langkah kedua juga kita bagi menjadi empat langkah: 2_a, 2_b, 2_c dan 2_d. Langkah 2_a adalah

$$generate(S, k_{ab}, 2_a, id)$$

dengan persyaratan:

- $receive(S, combine(a, b, n), 1_c, id)$ ada dalam *history* untuk *nonce* n yang sembarang.
- $generate(S, k, 2_a, id)$ belum ada dalam *history* untuk sembarang kunci k.

Langkah 2_b adalah

$$construct(S, m, 2_b, id)$$

dengan persyaratan:

- $generate(S, k_{ab}, 2_a, id)$ ada dalam *history*.
- n didapat dari $receive(S, combine(a, b, n), 1_c, id)$,

dimana

$$m = encrypt(combine(n, k_{ab}, b, encrypt(combine(k_{ab}, a), K_{bS})), K_{aS}).$$

Langkah 2_c adalah

$$send(S, m, 2_c, id)$$

dengan persyaratan:

- $construct(S, m, 2_b, id)$ ada dalam *history*.
- $send(S, m, 2_c, id)$ belum ada dalam *history*,

dimana

$$m = encrypt(combine(n, k_{ab}, b, encrypt(combine(k_{ab}, a), K_{bS})), K_{aS})$$

Langkah 2_d adalah

$$receive(a, m, 2_d, id)$$

dengan persyaratan:

- $send(S, m, 2_c, id)$ ada dalam *history*.
- $receive(a, m, 2_d, id)$ belum ada dalam *history*,

dimana

$$m = encrypt(combine(n, k_{ab}, b, encrypt(combine(k_{ab}, a), K_{bS})), K_{aS}).$$

Langkah ketiga kita bagi menjadi tiga langkah: 3_a, 3_b dan 3_c. Langkah 3_a adalah

$$construct(a, encrypt(combine(k_{ab}, a), K_{bS}), 3_a, id)$$

dengan persyaratan $receive(a, m, 2_d, id)$ ada dalam *history* dimana

$$m = encrypt(combine(n, k_{ab}, b, encrypt(combine(k_{ab}, a), K_{bS})), K_{aS}).$$

Langkah 3_b adalah

$$send(a, encrypt(combine(k_{ab}, a), K_{bS}), 3_b, id)$$

dengan persyaratan:

- $construct(a, encrypt(combine(k_{ab}, a), K_{bS}), 3_a, id)$ ada dalam *history*.
- $send(a, encrypt(combine(k_{ab}, a), K_{bS}), 3_b, id)$ belum ada dalam *history*.

Langkah 3_c adalah

$$receive(b, encrypt(combine(k_{ab}, a), K_{bS}), 3_c, id)$$

dengan persyaratan:

- $send(a, encrypt(combine(k_{ab}, a), K_{bS}), 3_b, id)$ ada dalam *history*.
- $receive(b, encrypt(combine(k_{ab}, a), K_{bS}), 3_c, id)$ belum ada dalam *history*.

Langkah keempat kita bagi menjadi empat langkah: 4_a, 4_b, 4_c dan 4_d. Langkah 4_a adalah

$$generate(b, n_b, 4_a, id)$$

dengan persyaratan:

- $receive(b, encrypt(combine(k_{ab}, a), K_{bS}), 3_c, id)$ ada dalam *history*.
- $generate(b, n, 4_a, id)$ belum ada dalam *history* untuk sembarang *nonce* n.

Langkah 4_b adalah

$$construct(b, encrypt(n_b, k_{ab}), 4_b, id)$$

dengan persyaratan $generate(b, n_b, 4_a, id)$ ada dalam *history*. Langkah 4_c adalah

$$send(b, encrypt(n_b, k_{ab}), 4_c, id)$$

dengan persyaratan:

- $construct(b, encrypt(n_b, k_{ab}), 4_b, id)$ ada dalam *history*.
- $send(b, encrypt(n_b, k_{ab}), 4_c, id)$ belum ada dalam *history*.

Langkah 4_d adalah

$$receive(a, encrypt(n_b, k_{ab}), 4_d, id)$$

dengan persyaratan:

- $send(b, encrypt(n_b, k_{ab}), 4_c, id)$ ada dalam *history*.
- $receive(a, encrypt(n_b, k_{ab}), 4_d, id)$ belum ada dalam *history*.

Langkah terahir kita bagi menjadi tiga langkah: 5_a, 5_b dan 5_c. Langkah 5_a adalah

$$construct(a, encrypt(combine(n_b, 1), k_{ab}), 5_a, id)$$

dengan persyaratan $receive(a, encrypt(n_b, k_{ab}), 4_d, id)$ ada dalam *history*. Kita gunakan $combine(n_b, 1)$ untuk $n_b - 1$ karena efeknya serupa. Langkah 5_b adalah

$$send(a, encrypt(combine(n_b, 1), k_{ab}), 5_b, id)$$

dengan persyaratan:

- $construct(a, encrypt(combine(n_b, 1), k_{ab}), 5_a, id)$ ada dalam *history*.
- $send(a, encrypt(combine(n_b, 1), k_{ab}), 5_b, id)$ belum ada dalam *history*.

Langkah 5_c adalah

$$receive(b, encrypt(combine(n_b, 1), k_{ab}), 5_c, id)$$

dengan persyaratan:

- $send(a, encrypt(combine(n_b, 1), k_{ab}), 5_b, id)$ ada dalam *history*.
- $receive(b, encrypt(combine(n_b, 1), k_{ab}), 5_c, id)$ belum ada dalam *history*.

Perumusan protokol telah dibuat *general* sehingga a dapat diperankan oleh A atau B, demikian juga dengan b. Jika $a = A$ dan $b = B$ maka $K_{aS} = K_{AS}$ dan $K_{bS} = K_{BS}$. Sebaliknya, jika $a = B$ dan $b = A$ maka $K_{aS} = K_{BS}$ dan $K_{bS} = K_{AS}$. Secara umum, huruf besar digunakan untuk nilai konstan, sedangkan huruf kecil digunakan untuk nilai yang tidak konstan. Huruf k kecil digunakan untuk k_{ab} karena nilai k_{ab} berbeda untuk sesi yang berbeda.

Langkah-langkah protokol yang telah dirumuskan dapat dilakukan kapan saja asalkan persyaratan *state machine* dan persyaratan protokol dipenuhi. Selain langkah protokol, langkah *principal* yang bersifat *intruder* dapat dilakukan kapan saja, asalkan persyaratan *state machine* dipenuhi. *Intruder* dapat melakukan berbagai langkah termasuk:

- langkah *generate*,
- langkah *construct*, dan
- langkah *intruder*.

Bermula dengan *initial state* dimana *history* masih kosong, berbagai langkah protokol dan *intruder* dapat dilakukan untuk membuat suatu evolusi. Contoh dari analisa adalah mencoba membuktikan bahwa suatu rahasia tidak bocor untuk semua kemungkinan evolusi. Satu cara untuk membuktikan bahwa rahasia tidak bocor adalah dengan menggunakan induksi. Untuk setiap langkah protokol dan setiap langkah *intruder* dicoba buktikan bahwa jika rahasia belum bocor sebelum langkah, maka rahasia tetap tidak bocor setelah langkah.

Kita coba buktikan bahwa K_{AS} dan K_{BS} tidak dibocorkan oleh langkah 1_a. Jadi, sebelum langkah, asumsi mengenai kunci pada *initial state* tetap berlaku. Bocor bisa dirumuskan sebagai

$$forgeable(I, K_{AS}) \vee forgeable(I, K_{BS})$$

dimana *principal* I adalah intruder dan $I \neq A$, $I \neq B$ dan $I \neq S$. Mari kita lihat efek dari langkah 1_a:

$$generate(a, n_a, 1_a, id)$$

yaitu

$$storage[a] \leftarrow storage[a] \cup \{n_a\}.$$

Jika $a \neq I$ maka pembuktian mudah karena

$$storage[I] \cup seen \cup public$$

tidak berubah, jadi

$$forgeable(I, m) = known(m, storage[I] \cup seen \cup public)$$

juga tidak berubah untuk semua $m \in message$. Jadi tidak ada *message* baru yang diketahui *intruder* I, dengan kata lain langkah 1_a tidak membocorkan informasi. Jika $a = I$, maka pembuktiannya agak lebih rumit. Pada dasarnya kita harus buktikan

$$\begin{aligned} &\forall k \in key, s \in messageSet, n \in nonce : \\ &\quad \neg known(k, s) \Longrightarrow \neg known(k, s \cup \{n\}). \end{aligned}$$

Kita tidak akan membuktikannya disini. Jelas bahwa untuk membuktikan bahwa protokol tidak membocorkan rahasia, teori perlu diperkuat dengan pembuktian berbagai teorema.

Percobaan analisa Needham-Schroeder *symmetric key protocol* menggunakan mekanisme yang telah kita bangun menunjukkan bahwa pembuktian keamanan protokol secara formal memang tidak mudah. Akan tetapi mekanisme yang telah kita bangun memberi kerangka yang cukup baik, dan apa yang diperlukan secara garis besar telah ditunjukkan. Untuk menyelesaikan pembuktian diperlukan ketekunan yang luar biasa dan kerja keras. Tentunya alat seperti *theorem prover* dapat digunakan untuk membantu pembuktian.

25.1 Ringkasan

Di bab ini kita telah bahas cara melakukan analisa protokol kriptografi. Analisa protokol kriptografi berfokus pada logika dari protokol, bukan kekuatan algoritma enkripsi. Analisa diperlukan karena logika protokol yang bermasalah dapat membocorkan rahasia, meskipun algoritma enkripsi yang digunakan sangat kuat. Ada dua jenis logika yang dapat digunakan sebagai dasar untuk analisa, yaitu *modal logic* dan *classical logic*. Biasanya analisa menggunakan *modal logic* dibantu dengan *model checker*, sedangkan analisa menggunakan *classical logic* dibantu dengan *theorem prover*. Cara analisa yang dikembangkan dan dibahas di bab ini menggunakan *classical logic* sebagai dasar. Analisa Needham-Schroeder *symmetric key protocol* digunakan sebagai contoh.

Bab 26

Kendala Penggunaan Kriptografi

Penggunaan kriptografi memang tidak dapat dihindarkan, dan kemajuan dalam ilmu dan teknologi kriptografi telah membuat penggunaan kriptografi cukup mudah untuk berbagai aplikasi. Namun masih ada beberapa kendala yang menghambat penggunaan kriptografi secara lebih luas. Kita akan bahas beberapa diantaranya di bab ini.

26.1 Manajemen Kunci

Manajemen kunci jelas merupakan sesuatu yang penting dalam penggunaan kriptografi. Komputerisasi manajemen kunci biasanya dilakukan menggunakan *public key infrastructure*. Di bab 23 telah kita bahas dua pendekatan yang berbeda untuk *public key infrastructure*:

- pendekatan PGP, dan
- pendekatan X.509.

Pendekatan PGP cocok untuk komunitas terbuka, namun tidak dapat digunakan dalam skala besar. Sebaliknya, pendekatan X.509 dapat digunakan dalam skala besar, namun tidak cocok untuk komunitas terbuka. Sampai saat ini belum ada pendekatan *public key infrastructure* yang efektif untuk penggunaan skala besar dalam komunitas terbuka.

Pada tingkat manajemen kunci yang lebih rinci, juga terdapat beberapa kendala, antara lain dalam hal:

- penyimpanan kunci privat oleh pengguna, dan

- manajemen *certificate*.

Dalam hal penyimpanan kunci privat oleh pengguna, ada solusi yang cukup baik jika ongkos bukan merupakan penghambat, yaitu solusi perangkat seperti *crypto device*. Akan tetapi, untuk penggunaan lebih luas, solusi perangkat masih terlalu mahal. Solusi murah dan populer saat ini adalah dengan *password protection* dimana kunci biasanya disimpan dalam *file*. Namun solusi ini rentan terhadap masalah *password*, diantaranya:

- *password* dapat terlupakan,
- *password* lebih mudah untuk dicuri dibandingkan perangkat, dan
- banyak *password* yang mudah untuk diterka.

Dalam hal manajemen *certificate*, hal yang dapat menjadi kendala penggunaan secara efektif adalah sulitnya untuk menentukan apakah suatu *certificate* masih *valid*. Untuk keperluan tertentu seperti identitas *web site*, ini mungkin bukan masalah besar. Akan tetapi untuk suatu komunitas dimana nilai suatu transaksi bisa sangat besar, ini menjadi masalah penting. Di bagian 23.2, kita telah bahas bagaimana konsep *certificate revocation* tidak efektif untuk keperluan ini. Mekanisme yang lebih baik daripada *certificate revocation* diperlukan untuk manajemen validitas *certificate*.

26.2 Sistem Terlalu Rumit

Yang dimaksud dengan sistem terlalu rumit disini adalah secara konseptual. Jika sistem terlalu rumit secara konseptual, maka sulit bagi pengguna untuk memahami cara kerja sistem. Akibat dari ketidak-pahaman atau kesalah-pahaman pengguna terhadap cara kerja sistem bisa jadi:

- sistem tidak digunakan dengan benar, atau
- sistem tidak digunakan.

Sebagai contoh, cara kerja suatu *public key infrastructure* tidak dipahami oleh sebagian pengguna karena terlalu rumit. Kesalah-pahaman dapat diperparah oleh antarmuka yang terlalu menyederhanakan, misalnya dalam penggunaan *certificate* untuk *secure web browsing*. Akibatnya, sistem bisa digunakan dengan tidak benar, misalnya menganggap bahwa transaksi dengan *web site* yang mempunyai *certificate* dianggap aman, padahal *certificate* hanya memberi jaminan identitas *web site*, tidak memberi jaminan bahwa pemilik *web site* dapat dipercaya. Pengguna yang tidak paham dengan cara kerja *public key infrastructure* dapat juga menghindar dari penggunaannya, misalnya dengan tidak

mengenkripsi *email* yang sensitif. Ini jelas berbahaya karena *email* dapat disadap, dan merupakan contoh dimana pengguna melakukan hal yang tidak aman karena merasa terlalu sulit untuk melakukan hal yang aman.

Standard aplikasi kriptografi kadang juga menggunakan konsep yang terlalu rumit. Misalnya konsep *distinguished name* dalam *certificate* berbasis X.509. Ini diperparah dengan adanya berbagai *profile* (istilah yang digunakan untuk lokalisasi standard) yang dibuat oleh berbagai negara dan organisasi. Akibatnya, berbagai program memproses *certificate* secara berbeda, *inter-operability* menjadi korban: *certificate* yang dibuat menggunakan suatu program dapat ditolak oleh program lain karena dianggap tidak memiliki format yang benar, padahal kedua program mengimplementasi standard X.509.

Rumitnya cara kerja suatu sistem secara konseptual kadang diperparah oleh implementasi yang menambah kerumitan. Sebagai contoh, antarmuka yang tidak intuitif dan tidak sesuai dengan cara kerja secara konseptual, dapat mempersulit penggunaan sistem. Antarmuka suatu sistem harus dibuat sesederhana mungkin, tanpa terlalu menyederhanakan, dan sesuai dengan cara kerja sistem secara konseptual.

26.3 Sistem Tidak Sesuai Kebutuhan

Banyak sistem yang tidak sesuai kebutuhan, contohnya sistem e-commerce yang mengandalkan *certificate authority* untuk pengamanan. Padahal yang diperlukan oleh kedua belah pihak adalah suatu *approval* oleh entitas yang dapat memberi jaminan bahwa transaksi bebas dari penipuan.

Solusi yang diberikan oleh *vendor* kerap tidak sesuai dengan kebutuhan *client*. Ini bisa terjadi karena:

- *vendor* tidak memahami kebutuhan *client*, atau
- *vendor* lebih mementingkan penjualan solusi daripada kebutuhan *client*.

Ketidak-pahaman *vendor* terhadap kebutuhan *client* biasanya terjadi karena masalah komunikasi. *Vendor* cenderung melihat dari sisi teknologi, sedangkan *client* kerap tidak paham dengan teknologi. Kerap *client* juga tidak tahu atau tidak bisa menjelaskan apa yang dibutuhkan. Terlalu sering proses *requirements analysis* diremehkan kedua belah pihak padahal proses itu sangat penting. Proses mencari tahu kebutuhan dan kemauan *client* sebaiknya merupakan proses yang cukup panjang dimana:

1. *Client* mengungkapkan keinginannya mengenai sistem.
2. *Vendor* mempelajari keinginan *client* lalu membuat *prototype* atau/dan *mockup* sistem sebagai rencana solusi.

3. *Vendor* memberi demonstrasi *prototype* atau/dan *mockup*.
4. *Client* memberi *feedback* berdasarkan demonstrasi.
5. *Vendor* merevisi *prototype* atau/dan *mockup* berdasarkan *feedback* dari *client*.

Langkah 3 sampai dengan 5 dapat diulang beberapa kali hingga *client* dan *vendor* cukup puas dengan rencana solusi.

Masalah *vendor* lebih mementingkan penjualan solusi daripada kebutuhan *client* dapat diatasi jika *client* tidak terlalu gampang terpengaruh oleh *sales pitch* dari *vendor* dan cukup mengetahui esensi dari kebutuhannya.

26.4 Ringkasan

Meskipun kemajuan dalam ilmu dan teknologi kriptografi telah membuat penggunaan kriptografi cukup mudah untuk berbagai aplikasi, masih terdapat beberapa kendala yang menghambat penggunaan kriptografi yang lebih luas. Di bab ini telah dibahas beberapa kendala penggunaan kriptografi, termasuk masalah manajemen kunci, sistem yang terlalu rumit dan sistem yang tidak sesuai dengan kebutuhan.

Bab 27

Masa Depan Kriptografi

Masa depan kriptografi akan dipengaruhi oleh perkembangan ilmu dan teknologi terkait. Ilmu dan teknologi yang perkembangannya akan sangat berpengaruh pada masa depan kriptografi termasuk:

- matematika,
- *hardware*, dan
- *quantum computing*.

Ketiga hal tersebut akan dibahas di bab ini.

27.1 Perkembangan Matematika

Tahun 1976, Martin Gardner menulis dalam Scientific American bahwa kunci RSA sebesar 129 digit akan aman untuk sekitar 40 *quadrillion* tahun. Kurang dari 20 tahun kemudian, tepatnya tahun 1994, kunci tersebut dapat diuraikan menggunakan metode *quadratic sieve*. Ini adalah contoh bagaimana terobosan di bidang matematika dan algoritma dapat mempengaruhi kriptografi secara signifikan. Dewasa ini metode *number field sieve* bahkan lebih efisien dibandingkan metode *quadratic sieve* dalam menguraikan bilangan yang sangat besar (lebih dari 100 digit).

Perkembangan lain di bidang matematika yang telah mempengaruhi kriptografi adalah penggunaan *elliptic curves over finite field*. Di masa yang akan datang, kriptografi *public key* yang berdasarkan pada penggunaan *elliptic curve* berpotensi mengambil alih posisi RSA sebagai algoritma yang dominan. Ini karena dengan kemajuan di bidang *hardware*, besarnya kunci yang dperlukan akan meningkat. Keunggulan kriptografi *public key* versi *elliptic curve* adalah keperluan peningkatan besar kunci tidak sedrastis untuk RSA, seperti terlihat

di tabel berikut yang menunjukkan perbandingan besar kunci dalam bit untuk kekuatan yang sama.

RSA	ECDSA/ECES
1024	160
2048	224
3072	256
7680	384
15360	512

RSA menggunakan kunci 1024 bit kekuatannya ekuivalen dengan kriptografi versi *elliptic curve* (ECDSA/ECES) menggunakan kunci 160 bit. RSA menggunakan kunci 15360 bit kekuatannya ekuivalen dengan ECDSA/ECES menggunakan kunci 512 bit. Dimana kunci RSA besarnya naik menjadi 15 kali lipat, untuk ECDSA/ECES kunci hanya diperlukan naik menjadi sekitar 3 kali lipat. Ini jelas menunjukkan keunggulan kriptografi *public key* versi *elliptic curve*.

Perkembangan dimasa depan dalam matematika dan algoritma, terutama dalam:

- penguraian bilangan bulat,
- komputasi logaritma diskrit, dan
- aljabar abstrak,

akan terus mempengaruhi kriptografi.

27.2 Perkembangan Hardware

DES sudah tidak digunakan lagi bukan karena algoritmanya lemah, melainkan besar kunci terlalu kecil. Saat ini kunci sebesar 56 bit dapat dicari secara *brute force* menggunakan *hardware* kini, dalam waktu yang tidak terlalu lama, dengan ongkos yang relatif murah.

Dengan perkembangan *hardware* di masa depan yang akan semakin cepat dan semakin murah, besar kunci untuk enkripsi mungkin perlu ditingkatkan. Saat ini enkripsi simetris dengan kunci 256 bit masih memiliki ruang cukup besar. Akan tetapi bisa saja terjadi terobosan di bidang *hardware* yang akan mengancam keamanan enkripsi simetris dengan kunci 256 bit.

Perkembangan *hardware* di masa depan tidak akan hanya berfokus pada peningkatan *clock speed*, namun juga pada peningkatan *parallelism*. Peningkatan *parallelism* akan terjadi di berbagai bagian, mulai dari bagian terkecil *processor* yang dapat dibuat *parallel*, sampai dengan *multi-processor* yang mempunyai interkoneksi dengan *bandwidth* yang sangat tinggi. Jenis *parallelism* juga akan

ada yang bersifat simetris dan akan ada yang bersifat asimetris misalnya menggunakan *co-processor*. Tentunya peningkatan *parallelism* di *hardware* juga akan diiringi dengan peningkatan penggunaan *parallelism* di *software*, baik yang secara otomatis dilakukan oleh *compiler*, maupun yang dilakukan secara *manual* oleh *programmer* misalnya menggunakan *threads*.

Tentunya jika *quantum computing* menjadi realitas, jenis kriptografi yang dapat digunakan secara efektif akan berbeda dari yang digunakan sekarang. Kita akan bahas *quantum computing* di bagian berikut.

27.3 Quantum Computing

Sekitar tahun 1982, Richard Feynman sedang mencoba melakukan simulasi interaksi beberapa partikel dalam fisika kuantum. Yang ia temukan adalah, jika menggunakan cara komputasi klasik (ekuivalen dengan penggunaan Turing *machine*), maka secara umum simulasi memerlukan sumber daya yang bersifat *exponential*. Untuk interaksi n partikel, simulasi menggunakan komputasi klasik membutuhkan sumber daya yang *exponential* dalam n, sedangkan alam dapat melakukannya hanya menggunakan n partikel dalam *real time*. Ini mengindikasikan bahwa komputasi klasik bukanlah cara paling efisien untuk melakukan komputasi, dan menjadi inspirasi untuk konsep *quantum computing* (komputasi kuantum). Ada komputasi yang mempunyai kompleksitas *exponential* dalam komputasi klasik tetapi mempunyai kompleksitas linear dalam komputasi kuantum. Persoalannya adalah bagaimana ini dapat dimanfaatkan.

Ada dua konsep fisika kuantum yang menjadi dasar dari komputasi kuantum:

- *superposition* dari *quantum states*, dan
- *quantum entanglement*.

Menurut fisika kuantum, suatu partikel bisa berada dalam suatu *quantum state* yang merupakan *superposition* dari dua *quantum state* murni sekaligus, dimana suatu *quantum state* murni adalah *quantum state* yang dapat diobservasi secara klasik. Sebagai contoh kita gunakan *spin* (perputaran) dari suatu partikel relatif terhadap suatu arah. Jika tidak nol, perputaran relatif terhadap suatu arah dapat diobservasi secara klasik sebagai *down* (0) atau *up* (1), yang masing-masing adalah *quantum state* murni. Menggunakan notasi fisika kuantum, kedua *quantum state* murni tersebut adalah

$$|0\rangle \text{ dan } |1\rangle.$$

Superposition ψ dari kedua *quantum state* murni adalah kombinasi linear

$$\psi = \alpha|0\rangle + \beta|1\rangle$$

dimana secara umum α dan β masing-masing bisa berupa bilangan kompleks, tetapi untuk keperluan kita cukup merupakan bilangan nyata. α adalah apa yang disebut *probability amplitude* untuk $|0\rangle$, sedangkan β adalah *probability amplitude* untuk $|1\rangle$. Jika observasi dilakukan terhadap partikel yang berada dalam *superposition* seperti diatas, maka kemungkinan bahwa partikel berada pada *state* $|0\rangle$ adalah α^2, kemungkinan bahwa partikel berada pada *state* $|1\rangle$ adalah β^2, sedangkan kemungkinan lain tidak ada. Jadi

$$\alpha^2 + \beta^2 = 1.$$

Yang menarik adalah setiap *probability amplitude*, yang merupakan bagian internal dari *superposition state*, bisa berupa bilangan negatif. Jika $|0\rangle$ dan $|1\rangle$ kita anggap sebagai basis, maka *state* dari partikel dapat direpresentasikan sebagai vektor

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

Dalam fisika kuantum, jika sepasang partikel telah berinteraksi maka terjadi *quantum entanglement* dimana apa yang terjadi pada satu partikel secara instan mempengaruhi partikel pasangannya. Masing-masing partikel bisa berada dalam suatu *superposition state*, tetapi pada saat observasi secara klasik dilakukan pada satu diantaranya (yang membuatnya "memilih" suatu *state* murni), maka partikel pasangannya langsung memasuki *state* murni yang sesuai. Contohnya, jika pasangan bersifat komplementer dan observasi membuat satu partikel memasuki *state* $|0\rangle$, maka partikel pasangannya langsung memasuki *state* $|1\rangle$.

Meskipun belum ada penjelasan yang memuaskan mengenai apa yang sebenarnya terjadi dengan *superposition* dan *entanglement* (masalah ini dijuluki *quantum interpretation problem*), dari segi matematika tidak ada masalah. Kita tidak akan bahas masalah *quantum interpretation* dan akan fokus pada matematika yang digunakan.

Jika unit informasi terkecil dalam komputasi klasik adalah bit, maka unit informasi terkecil dalam komputasi kuantum adalah qubit. Berbeda dengan bit yang hanya bisa mempunyai nilai 0 atau 1, qubit bisa mempunyai nilai yang merupakan *superposition* dengan representasi $\alpha|0\rangle + \beta|1\rangle$. Komputasi kuantum dilakukan menggunakan 3 macam *gate* dengan input 1 qubit, dan 1 macam *gate* dengan input 2 qubit. *Gate* pertama adalah *not gate* N yang dapat direpresentasikan menggunakan matrik transformasi sebagai berikut:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Efek dari transformasi *not* terhadap qubit $\alpha|0\rangle + \beta|1\rangle$ adalah

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix},$$

jadi menghasilkan qubit $\beta|0\rangle+\alpha|1\rangle$. *Gate* kedua adalah Hadamard *gate* H yang dapat direpresentasikan menggunakan matrik transformasi sebagai berikut:

$$\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}.$$

Efek dari transformasi Hadamard terhadap qubit $\alpha|0\rangle + \beta|1\rangle$ adalah

$$\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}}(\alpha+\beta) \\ \frac{1}{\sqrt{2}}(\alpha-\beta) \end{bmatrix},$$

jadi menghasilkan qubit $\frac{1}{\sqrt{2}}(\alpha+\beta)|0\rangle + \frac{1}{\sqrt{2}}(\alpha-\beta)|1\rangle$. Perhatikan bahwa jika transformasi Hadamard *gate* dilakukan terhadap $|0\rangle$ maka kita akan dapatkan $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, yang merupakan nilai *superposition* dengan probabilitas yang sama untuk menghasilkan $|0\rangle$ atau $|1\rangle$ jika observasi secara klasik dilakukan. Jika transformasi Hadamard *gate* dilakukan terhadap $|1\rangle$ maka kita akan dapatkan $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$, yang juga mempunyai probabilitas yang sama untuk menghasilkan $|0\rangle$ atau $|1\rangle$ jika observasi secara klasik dilakukan. *Gate* ketiga adalah *gate* Z yang dapat direpresentasikan menggunakan matrik transformasi sebagai berikut:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Efek dari transformasi *gate* Z terhadap qubit $\alpha|0\rangle + \beta|1\rangle$ adalah

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ -\beta \end{bmatrix},$$

jadi menghasilkan qubit $\alpha|0\rangle - \beta|1\rangle$. Jadi *gate* Z melakukan negasi terhadap *probability amplitude* untuk $|1\rangle$.

Ketiga transformasi diatas merupakan apa yang disebut *unitary transformation*. Matrik U disebut *unitary* jika $UU\dagger = I$, dimana $U\dagger$ didapat dengan men-*transpose* U kemudian melakukan *complex conjugate* terhadap hasilnya. Untuk ketiga transformasi diatas, $U\dagger = U$, jadi U merupakan *inverse* untuk U. Kita dapat periksa bahwa $NN = I$, $HH = I$ dan $ZZ = I$. Jadi setelah transformasi terhadap qubit, transformasi dapat juga digunakan untuk mengembalikan qubit ke *state* semula. Sebagai contoh, jika transformasi Hadamard dilakukan terhadap $\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle$, maka kita akan dapatkan kembali $|0\rangle$. Dalam komputasi kuantum, semua transformasi bersifat *reversible*.

Gate dengan input 2 qubit yang diperlukan adalah apa yang dinamakan *controlled-not* . Dalam komputasi kuantum, 2 qubit merupakan produk tensor,

jadi jika $|\psi_1\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$ dan $|\psi_2\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$, maka

$$|\psi_1, \psi_2\rangle = |\psi_1\rangle|\psi_2\rangle = \begin{bmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{bmatrix}.$$

Jika 2 qubit direpresentasikan seperti diatas, maka transformasi *controlled-not* dapat direpresentasikan dengan matrik

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Hasil transformasi *controlled-not* terhadap $|\psi_1, \psi_2\rangle$ adalah

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{bmatrix} = \begin{bmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\beta_2 \\ \beta_1\alpha_2 \end{bmatrix}.$$

Tidak terlalu sulit untuk melihat bahwa transformasi *controlled-not* merupakan *unitary transformation*. Mari kita lihat efek transformasi jika $|\psi_1\rangle = \alpha_1|0\rangle + \beta_1|1\rangle = |0\rangle$. Karena $\alpha_1 = 1$ dan $\beta_1 = 0$, maka hasilnya adalah

$$\begin{bmatrix} \alpha_2 \\ \beta_2 \\ 0 \\ 0 \end{bmatrix},$$

yang berarti tidak ada efek karena hasil sama dengan keadaan semula yaitu

$$\begin{bmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{bmatrix}.$$

Sebaliknya, jika $|\psi_1\rangle = |1\rangle$, maka hasilnya adalah

$$\begin{bmatrix} 0 \\ 0 \\ \beta_2 \\ \alpha_2 \end{bmatrix},$$

yang berarti terjadi *negation* pada $|\psi_2\rangle$ (transformasi menukar dua baris terahir).

Transformasi *controlled not* mengakibatkan *quantum entanglement* dimana kedua qubit menjadi saling tergantung. Sebagai contoh, kita mulai dengan

$$\alpha_1 = \frac{1}{\sqrt{2}}, \beta_1 = \frac{1}{\sqrt{2}}, \alpha_2 = 1, \beta_2 = 0,$$

jadi qubit pertama berada pada *superposition* dengan probabilitas yang sama untuk menjadi 0 atau 1, sedangkan qubit kedua mempunyai nilai 0. Setelah transformasi *controlled-not*, maka nilai 2 qubit sebagai vektor adalah

$$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix},$$

yang berarti qubit kedua juga berada pada *superposition* dengan probabilitas yang sama untuk menjadi 0 atau 1. Akan tetapi, jika observasi secara klasik dilakukan pada qubit pertama dan sebagai contoh memberi hasil 1 dengan *probability amplitude* 1, maka vektor berubah menjadi

$$\begin{bmatrix} 0 \\ 0 \\ \beta_2 \\ \alpha_2 \end{bmatrix}.$$

Jadi *state* untuk qubit kedua berubah menjadi

$$\begin{aligned} |\psi_2\rangle &= \beta_2|0\rangle + \alpha_2|1\rangle \\ &= |1\rangle, \end{aligned}$$

yang berarti qubit kedua dipaksa mempunyai nilai 1. Perubahan *state* yang disebabkan oleh observasi secara klasik disebut *decoherence*.

Yang sangat menarik dengan komputasi kuantum adalah, jika terdapat n qubit dan setiap qubit berada pada *superposition* dengan kemungkinan yang sama untuk menjadi 0 atau 1, maka komputasi yang dilakukan sekaligus dilakukan pada 2^n *state*. Akan tetapi komputasi yang dilakukan harus bersifat *reversible*, dan kita harus mengharapkan bahwa *decoherence* memberi hasil yang kita inginkan. Oleh sebab itu, biasanya cara kerja komputasi kuantum adalah sebagai berikut:

- Mulai dengan *state space* yang besar.
- Secara bertahap lakukan transformasi yang memperbesar probabilitas *decoherence* akan memberikan hasil yang diinginkan.

- Jika probabilitas sudah cukup besar bahwa *decoherence* akan memberikan hasil yang diinginkan, maka lakukan observasi untuk mendapatkan jawaban.

Pada tahun 1994 Peter Shor (lihat [sho9]) berhasil membuat algoritma untuk menguraikan bilangan bulat yang menggunakan komputasi kuantum sebagai *subroutine*. Untuk menguraikan suatu bilangan ganjil n, pilih secara acak suatu bilangan bulat x dimana $1 < x < n$ dan $\gcd(x, n) = 1$, lalu cari *order* dari x dalam *multiplicative group* modulo n (lihat bagian 10.4), sebut saja r. Setelah r didapat, maka lakukan kalkulasi $\gcd(x^{r/2} - 1, n)$. Karena

$$\begin{aligned}(x^{r/2} - 1)(x^{r/2} + 1) &= x^r - 1 \\ &\equiv 0 \pmod{n},\end{aligned}$$

maka $\gcd(x^{r/2} - 1, n)$ bukan merupakan pembagi yang *non-trivial* dari n hanya jika r ganjil atau $x^{r/2} \equiv -1 \pmod{n}$. Dapat ditunjukkan bahwa probabilitas kegagalan $< 1/2^{k-1}$ dimana k adalah banyaknya bilangan prima ganjil yang membagi n. Dengan menggunakan komputasi kuantum untuk mencari r, kompleksitas dari algoritma ini adalah *polynomial* dalam $\log n$. Jadi jelas jika komputasi kuantum dapat dilakukan dalam skala cukup besar, ini merupakan ancaman terhadap kriptografi yang keamanannya berbasis pada sukarnya penguraian seperti RSA.

27.4 Ringkasan

Di bab ini kita telah bahas perkembangan di masa depan yang dapat mempengaruhi kriptografi secara signifikan, yaitu perkembangan matematika, perkembangan *hardware*, dan *quantum computing*. Terutama jika *quantum computing* menjadi kenyataan, maka jenis kriptografi *public key* yang digunakan harus berubah secara revolusioner.

Appendix A

Daftar Notasi, Singkatan dan Istilah

$A \implies B$	Notasi logika untuk "A hanya jika B" (atau "B jika A").
$A \iff B$	Notasi logika untuk "A jika dan hanya jika B."
$A \wedge B$	Notasi logika untuk "A dan B."
$A \vee B$	Notasi logika untuk "A atau B."
$\neg A$	Notasi logika untuk "Tidak A."
$\forall x : P(x)$	Notasi logika untuk "untuk setiap $x : P(x)$."
$\forall x \in S : P(x)$	Notasi logika untuk "untuk setiap $x \in S : P(x)$."
$\exists x : P(x)$	Notasi logika untuk "terdapat $x : P(x)$."
$\exists x \in S : P(x)$	Notasi logika untuk "terdapat $x \in S : P(x)$."
$x \in S$	Notasi untuk "x adalah elemen himpunan S."
$\{x \in S \mid P(x)\}$	Notasi untuk himpunan terdiri dari elemen-elemen $x \in S$ yang dipilih jika proposisi $P(x)$ berlaku.
$\{f(x) \mid x \in S\}$	Notasi untuk himpunan yang merupakan *image* dari S berdasarkan pemetaan fungsi f.

$S_1 \cup S_2$	$a \in (S_1 \cup S_2) \Longrightarrow a \in S_1 \vee a \in S_2$.
$S_1 \cap S_2$	$a \in (S_1 \cap S_2) \Longrightarrow a \in S_1 \wedge a \in S_2$.
$S_1 \subseteq S_2$	$\forall x \in S_1 : x \in S_2$.
$S_1 \subset S_2$	$S_1 \neq S_2 \wedge \forall x \in S_1 : x \in S_2$.
$S_1 \setminus S_2$	$a \in (S_1 \setminus S_2) \Longrightarrow a \in S_1 \wedge a \notin S_2$.
$\sum_{i=0}^{n} a_i$	$a_0 + a_1 + \ldots + a_n$.
$\prod_{i=0}^{n} a_i$	$a_0 \cdot a_1 \cdot \ldots \cdot a_n$.
$S_1 \oplus S_2$	Notasi untuk "S_1 *bitwise exclusive or* dengan S_2."
AES	Advanced Encryption Standard.
bijection	Pemetaan atau fungsi yang bersifat *bijective*.
bijective	Bersifat *injective* dan *surjective*.
birthday attack	Attack yang dasar logikanya sama dengan birthday paradox.
birthday paradox	Jika 23 orang secara sembarang dikumpulkan dalam satu ruangan, maka probabilitas bahwa ada yang mempunyai ulang tahun yang sama melebihi 50 persen.
block cipher	Teknik enkripsi per blok.
brute force search	Pencarian dengan memeriksa semua kemungkinan.
ciphertext	Naskah acak (hasil enkripsi).
CA	Certificate Authority.
CAST	Carlisle Adams / Stafford Tavares.
CBC	Cipher Block Chaining.

CFB	Cipher Feedback.
CRL	Certificate Revocation List.
cryptanalysis	Analisa untuk memecahkan enkripsi.
DES	Data Encryption Standard.
DH	Diffie-Hellman, algoritma untuk *key agreement*.
digital signature	Tanda-tangan digital menggunakan kriptografi, biasanya menggunakan RSA atau DSA/DSS.
DSA	Digital Signature Algorithm.
DSS	Digital Signature Standard.
ECB	Electronic Code Book.
embedding	Injective homomorphism.
field	Ring dimana setiap elemen $a \neq 0$ mempunyai *inverse* a^{-1} ($aa^{-1} = 1$).
gcd	Greatest common divisor.
HMAC	Hash Message Authentication Code.
homomorphism	Pemetaan yang mempertahankan struktur aljabar.
ideal	Subset dari ring dengan sifat *closed* untuk +, dan *inside-outside multiplication*.
injection	Pemetaan atau fungsi yang bersifat *injective*.
injective	φ *injective*: $\varphi(x) = \varphi(y) \Longrightarrow x = y$.
IPsec	Internet Protocol security.
isomorphism	Bijective homomorphism.

IV	Initialization Vector.
key generation	Pembuatan kunci secara acak.
key management	Manajemen kunci, termasuk *key generation*, *transmission*, *storage*.
Las Vegas	Algoritma probabilistik yang, jika sukses, hasilnya dijamin benar.
LDAP	Lightweight Directory Access Protocol.
LFSR	Linear Feedback Shift Register.
MD5	Message Digest 5.
mod	Modulo.
Monte Carlo	Algoritma probabilistik yang hampir selalu benar.
NIST	National Institute of Standards and Technology.
NSA	National Security Agency.
OFB	Output Feedback.
one-time pad	Teknik enkripsi menggunakan *keystream* yang tidak mempunyai periode.
permutation	Perubahan urutan sub-unit data.
PGP	Pretty Good Privacy.
PKI	Public Key Infrastructure.
plaintext	Naskah asli (sebelum enkripsi atau sesudah dekripsi).
PRNG	Pseudo-random number generator.

ring	Struktur aljabar dengan + dan · dimana + membentuk Abelian group dengan *identity* 0, · membentuk Abelian monoid dengan *identity* 1, · *distributive* terhadap +.
RNG	Random number generator.
RSA	Rivest, Shamir, Adleman, algoritma untuk *public key cryptography*.
SHA	Secure Hash Algorithm.
S/MIME	Secure / Multipurpose Internet Mail Extensions.
SSH	Secure Shell.
SSL/TLS	Secure Socket Layer / Transport Layer Security.
stream cipher	Teknik enkripsi menggunakan *keystream* yang mempunyai periode.
substitution	Penukaran unit data.
surjection	Pemetaan atau fungsi yang bersifat *surjective*.
surjective	$\varphi : A \longrightarrow B$ *surjective*: $\varphi(A) = B$.
unit	Elemen ring yang mempunyai *multiplicative inverse*.
Vernam cipher	Teknik enkripsi menggunakan *keystream*.
VPN	Virtual Private Network.
WEP	Wireless Equivalent Privacy.
WPA	Wi-Fi Protected Access.
zero divisor	Suatu elemen $a \neq 0$ dari ring R, dimana $\exists b \in R : b \neq 0 \wedge ab = 0$.

Appendix B

Tabel untuk *cipher f* DES

E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Tabel B.1: Tabel Transformasi Expansi E

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Tabel B.2: Tabel Permutasi P

Tabel B.1 menunjukkan transformasi ekspansi E. Bit 1 output ekspansi didapat dari bit 32 input, bit 2 output dari bit 1 input, bit 3 output dari bit 2 input, dan seterusnya, sampai bit 48 output dari bit 1 input.

Tabel B.2 menunjukkan permutasi P. Bit 1 output permutasi didapat dari bit 16 input, bit 2 output dari bit 7 input, bit 3 output dari bit 20 input, dan seterusnya.

Tabel B.5 memperlihatkan fungsi substitusi S1 sampai dengan S8. Setiap subtabel mempunyai 4 baris (dengan indeks 0, 1, 2, 3) dan 16 kolom (dengan indeks 0 sampai dengan 15). Input 6 bit digunakan sebagai indeks baris (menurut tabel B.3) dan indeks kolom (menurut tabel B.4). Sebagai contoh, jika input 6 bit adalah 011011, maka indeks baris adalah 1 (karena bit 1, 6 mempunyai nilai 01), dan indeks kolom adalah 13 (karena bit 2, 3, 4, 5 mempunyai nilai 1101). Dengan input 011011, S1 akan menghasilkan 4 bit 0101 karena baris 1 kolom 13 dalam tabel S1 mempunyai nilai 5, yang dalam notasi biner 4 bit adalah 0101.

Bit 1	Bit 6	Nilai Indeks Baris
0	0	0
0	1	1
1	0	2
1	1	3

Tabel B.3: Kalkulasi Indeks Baris Untuk S1-S8

Bit 2	Bit 3	Bit 4	5	Nilai Indeks Kolom
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Tabel B.4: Kalkulasi Indeks Kolom Untuk S1-S8

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tabel B.5: Tabel untuk S1-S8

Appendix C

Tabel S-box AES

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabel C.1: Tabel S-box AES

Tabel C.1 adalah tabel untuk transformasi S-box terhadap byte. Sebagai contoh, jika sebelum transformasi byte memiliki nilai 01101110 (6e dalam notasi hexadecimal), kita cari baris 6 kolom e dalam tabel, dan kita dapatkan 9f (10011111 dalam notasi biner) sebagai nilai byte setelah transformasi. Tabel C.2 adalah tabel untuk *inverse* transformasi S-box.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Tabel C.2: Tabel Inverse S-box AES

Appendix D

Tabel untuk algoritma MD5

Tabel D.1 adalah tabel yang digunakan algoritma MD5 dalam proses *hashing*, dimana nilai isi tabel dalam hexadecimal.

$T[1]$	d76aa478	$T[17]$	f61e2562	$T[33]$	fffa3942	$T[49]$	f4292244
$T[2]$	e8c7b756	$T[18]$	c040b340	$T[34]$	8771f681	$T[50]$	432aff97
$T[3]$	242070db	$T[19]$	265e5a51	$T[35]$	6d9d6122	$T[51]$	ab9423a7
$T[4]$	c1bdceee	$T[20]$	e9b6c7aa	$T[36]$	fde5380c	$T[52]$	fc93a039
$T[5]$	f57c0faf	$T[21]$	d62f105d	$T[37]$	a4beea44	$T[53]$	655b59c3
$T[6]$	4787c62a	$T[22]$	2441453	$T[38]$	4bdecfa9	$T[54]$	8f0ccc92
$T[7]$	a8304613	$T[23]$	d8a1e681	$T[39]$	f6bb4b60	$T[55]$	ffeff47d
$T[8]$	fd469501	$T[24]$	e7d3fbc8	$T[40]$	bebfbc70	$T[56]$	85845dd1
$T[9]$	698098d8	$T[25]$	21e1cde6	$T[41]$	289b7ec6	$T[57]$	6fa87e4f
$T[10]$	8b44f7af	$T[26]$	c33707d6	$T[42]$	eaa127fa	$T[58]$	fe2ce6e0
$T[11]$	ffff5bb1	$T[27]$	f4d50d87	$T[43]$	d4ef3085	$T[59]$	a3014314
$T[12]$	895cd7be	$T[28]$	455a14ed	$T[44]$	4881d05	$T[60]$	4e0811a1
$T[13]$	6b901122	$T[29]$	a9e3e905	$T[45]$	d9d4d039	$T[61]$	f7537e82
$T[14]$	fd987193	$T[30]$	fcefa3f8	$T[46]$	e6db99e5	$T[62]$	bd3af235
$T[15]$	a679438e	$T[31]$	676f02d9	$T[47]$	1fa27cf8	$T[63]$	2ad7d2bb
$T[16]$	49b40821	$T[32]$	8d2a4c8a	$T[48]$	c4ac5665	$T[64]$	eb86d391

Tabel D.1: Tabel untuk *hashing* MD5

Daftar Pustaka

[ada01] C. Adams, P. Cain, D. Pinkas, R. Zuccherato, "Internet X.509 Public Key Infrastucture Time-Stamp Protocol," *RFC 3161*, The Internet Society, August 2001.

[ada05] C. Adams, S. Farrell, T. Kause, T. Mononen, "Internet X.509 Public Key Infrastucture Certificate Management Protocol," *RFC 4210*, The Internet Society, September 2005.

[adl78] L.M. Adleman, R.L. Rivest, A. Shamir, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM 21*, pp. 120-126, 1978.

[agr04] Manindra Agrawal, Neeraj Kayal, Nitin Saxena, "PRIMES in P," *Annals of Mathematics 160*, 2004.

[asp82] Alain Aspect, Philippe Grangier, Gérard Roger, "Experimental Realization of Einstein-Podolsky-Rosen *Gedankenexperiment*: A New Violation of Bell's Inequalities," *Physical Review Letters Vol. 49 No. 2*, pp. 91-94, 1982.

[atk93] A.O.L. Atkin, F. Morain, "Elliptic Curves and Primality Proving," *Mathematics of Computation Vol. 61*, pp. 29-68, July 1993.

[bac96] Eric Bach, Jeffrey Shallit, *Algorithmic Number Theory Vol. 1*, MIT Press, 1996.

[ben84] Charles H. Bennett, Gilles Brassard, "Quantum Cryptography: Public Key Distribution and Coin Tossing," *International Conference on Computers, Systems and Signal Processing*, 1984.

[bih91] Eli Biham and Adi Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology Vol. 4*, pp. 3-72, 1991.

[bol96] Dominique Bolignano, "An Approach to the Formal Verification of Cryptographic Protocols," *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pp. 106-118, 1996.

[buh93] J.P. Buhler, H.W. Lenstra, Jr., Carl Pomerance, "Factoring Integers with the Number Field Sieve," in A.K. Lenstra, H.W. Lenstra, Jr. (eds.), "The Development of the Number Field Sieve," pp. 50-94, Springer Verlag 1993.

[bur90] Michael Burrows, Martin Abadi, Roger Needham, "A Logic of Authentication," *ACM Transactions on Computer Systems, Vol. 8, No. 1*, pp. 18-36, February 1990.

[cal07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, R. Thayer, "OpenPGP Message Format," *RFC 4880*, The Internet Society, November 2007.

[ced08] Jörgen Cederlöf, Jan-Åke Larsson, "Security Aspects of the Authentication Used in Quantum Cryptography," *IEEE Transactions on Information Theory, Vol. 54, No. 4*, pp. 1735-1741, 2008.

[coh84] H. Cohen and H.W. Lenstra, Jr., "Primality testing and Jacobi sums," *Mathematics of Computation Vol. 42*, pp. 297-330, 1984.

[cop84] D. Coppersmith "Evaluating Logarithms in $\mathbf{GF}(2^n)$," *Proceedings of the 16th ACM Symposium on Theory of Computing*, pp. 201-207, 1984.

[cus06] F. Cusack, M. Forssen, "Generic Message Exchange Authentication for The Secure Shell (SSH) Protocol," *RFC 4256*, The Internet Society, January 2006.

[die99] T. Dierks, C. Allen, "The TLS Protocol Version 1.0," *RFC 2246*, The Internet Society, 1999.

[die08] T. Dierks, E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," *RFC 5246*, The Internet Society, 2008.

[dix81] J.D. Dixon, "Asymptotically Fast Factorization of Integers," *Mathematics of Computation Vol. 36*, pp. 255-260, 1981.

[fin97] Benjamin Fine, Gerhard Rosenberger, *The Fundamental Theorem of Algebra*, Undergraduate Texts in Mathematics, Springer-Verlag, 1997.

[flu01] Scott Fluhrer, Itsik Mantin and Adi Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," *Eighth Annual Workshop on Selected Areas in Cryptography*, 2001.

[fri96] A. Frier, P. Karlton, P. Kocher, "The SSL 3.0 Protocol," Netscape Communications Corporation, 1996.

[gol86] Shafi Goldwasser, Joe Kilian, "Almost All Primes can Be Quickly Certified," *Proceedings of the 18th Annual Symposium on the Theory of Computing*, pp. 316-329, 1986.

[gut04] Peter Gutmann, *Cryptographic Security Architecture: Design and Verification*, Springer, 2004.

[har79] G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers, 5th Edition*, Oxford University Press, 1979.

[hel78] M.E. Hellman, R.C. Merkle, "Hiding Information and Signatures in Trapdoor Knapsacks," *IEEE Transactions on Information Theory*, pp. 525-530, 1978.

[hic95] Hickman, Kipp, "The SSL Protocol," Netscape Communications Corporation, 1995.

[kau05] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol," *RFC 4306*, The Internet Society, December 2005.

[ken05a] S. Kent, K. Seo, "Security Architecture for the Internet Protocol," *RFC 4301*, The Internet Society, December 2005.

[ken05b] S. Kent, "IP Authentication Header," *RFC 4302*, The Internet Society, December 2005.

[ken05c] S. Kent, "IP Encapsulating Security Payload (ESP)," *RFC 4303*, The Internet Society, December 2005.

[kle07] Andreas Klein, "Attacks on RC4 Stream Cipher," *submitted to Designs, Codes and Cryptography,* 2007.

[kle10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, Paul Zimmermann, "Factorization of 768-bit RSA Modulus," version 1.3, January 2010.

[kob94] Neal Koblitz, *A Course in Number Theory and Cryptography, Second Edition*, Springer, 1994.

[leh06] S. Lehtinen, C. Lonvick, "The Secure Shell (SSH) Protocol Assigned Numbers," *RFC 4250*, The Internet Society, January 2006.

[len82] A.K. Lenstra, H.W. Lenstra Jr, L Lovász, "Factoring Polynomials with Rational Coefficients," *Mathematische Annalen 261*, pp. 515-534, Springer-Verlag 1982.

[len05] Arjen Lenstra, Xiaoyun Wang, Benne de Weger, "Colliding X.509 Certificates," technical paper, 2005.

[mak09] V. Makarov, A. Anisimov, S. Sauge, "Quantum Hacking: Adding a Commercial Actively-Quenched Module to the List of Single-Photon Detectors Controllable by Eve," 2009.

[mat93] Mitsuru Matsui, "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology - Eurocrypt 93*, 1993.

[men95] Alfred Menezes, "Elliptic Curve Cryptosystems," *CryptoBytes, Vol. 1, No. 2*, RSA Laboratories, Summer 1995.

[mer79] Ralph Charles Merkle, "Secrecy, Authentication, and Public Key Systems," Stanford University PhD Dissertation, 1979.

[mor75] Michael A. Morrison, John Brillhart, "A Method of Factoring and the Factorization of F7," *Mathematics of Computation Vol. 29*, pp. 183-205, 1975.

[mye99] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adamas, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol," *RFC 2560*, The Internet Society, June 1999.

[nis99] National Institute of Standards and Technology, *Data Encryption Standard*, FIPS PUB 46-3, U.S. Department of Commerce, October 1999.

[nis00] National Institute of Standards and Technology, *Digital Signature Standard*, FIPS PUB 186-2, U.S. Department of Commerce, January 2000.

[nis01] National Institute of Standards and Technology, *Advanced Encryption Standard*, FIPS PUB 197, U.S. Department of Commerce, November 2001.

[nis07] National Institute of Standards and Technology, *Establishing Wireless Robust Security Networks: A Guide to IEEE802.11i*, NIST SP 800-97, U.S. Department of Commerce, February 2007.

[nis08] National Institute of Standards and Technology, *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, NIST SP 800-67 Version 1.1, U.S. Department of Commerce, May 2008.

[nis08a] National Institute of Standards and Technology, *Guide to Securing Legacy IEEE 802.11 Wireless Networks*, NIST SP 800-48 Rev. 1, U.S. Department of Commerce, July 2008.

[nis08b] National Institute of Standards and Technology, *Guide to Bluetooth Security*, NIST SP 800-121, U.S. Department of Commerce, September 2008.

[pau98] Lawrence C. Paulson, "The Inductive Approach to Verifying Cryptographic Protocols," *Journal of Computer Security, Vol. 6*, pp. 85-128, 1998.

[pom82] C. Pomerance, "Analysis and Comparison of some Integer Factoring Algorithms," *Computational Methods in Number Theory* pp. 89-139, 1982.

[rab80] M.O. Rabin, "Probabilistic Algorithm for Testing Primality," *Journal on Number Theory, Vol. 12*, pp. 128-138, 1980.

[ram04] B. Ramsdell, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Format," *RFC 3851*, The Internet Society, July 2004.

[riv92] Ron Rivest, "The MD5 Message-Digest Algorithm," *RFC 1321*, IETF, April 1992.

[rob97] M.J.B. Robshaw, Yiqun Lisa Lin, "Overview of Elliptic Curve Cryptosystems," *Technical Note*, RSA Laboratories, June 1997.

[sch06] J. Schlyter, W. Griffin, "Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints," *RFC 4255*, The Internet Society, January 2006.

[sch85] R. Schoof, "Elliptic Curves over Finite Fields and the Computation of Square Roots Mod p," *Mathematics of Computation Vol. 44*, pp. 483-494, 1985.

[ser06] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol," *RFC 4511*, The Internet Society, June 2006.

[sha02] National Institute of Standards and Technology, *Secure Hash Standard*, FIPS PUB 180-2, U.S. Department of Commerce, August 2003.

[sha82] A. Shamir, "A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystems," *Proceedings of the 23rd Annual Symposium on the Foundations of Computer Science*, pp. 145-152, 1982.

[sha48] Claude Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal, Vol. 27*, pp. 379-423, 623-656, July, October 1948.

[sha49] Claude Shannon, "Communication Theory of Secrecy Systems," 1949.

[sho9] P.W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, pp. 124-134, IEEE Computer Society Press, 1994.

[sol77] R. Solovay, V. Strassen, "A Fast Monte-Carlo Test for Primality," *SIAM Journal on Computing, Vol. 6*, pp. 84-85, 1977.

[stu01] Adam Stubblefield, John Ioannidis, Aviel Rubin, "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP," ATT Labs Technical Report TD-4ZCPZZ, 2001.

[tew07] Erik Tews, Ralph-Philipp Weinmann, Andrei Pyshkin, "Breaking 104 bit WEP in less than 60 seconds," technical paper, TU Darmstadt, 2007.

[wae66] B.L. van der Waerdan, *Algebra*, 7th edition, Springer-Verlag, 1966.

[wan05] Xiaoyun Wang, Hongbo Yu, "How to Break MD5 and Other Hash Functions," *Advances in Cryptology - Eurocrypt 2005*, May 2005.

[wyy05] Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu, "Finding Collisions in the Full SHA-1," *Advances in Cryptology - Eurocrypt 2005*, May 2005.

[weg81] M. Wegman, L. Carter, "New Hash Functions and Their Use in Authentication and Set Equality," *Journal of Computer and System Sciences, Vol. 22*, pp. 265-279, 1981.

[wei63] E. Weiss, *Algebraic Number Theory*, McGraw-Hill, 1963.

[ylo06a] T. Ylönen, C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," *RFC 4251*, The Internet Society, January 2006.

[ylo06b] T. Ylönen, C. Lonvick, "The Secure Shell (SSH) Authentication Protocol," *RFC 4252*, The Internet Society, January 2006.

[ylo06c] T. Ylönen, C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol," *RFC 4253*, The Internet Society, January 2006.

[ylo06d] T. Ylönen, C. Lonvick, "The Secure Shell (SSH) Connection Protocol," *RFC 4254*, The Internet Society, January 2006.

Indeks

Buku ini menjelaskan teori dibalik kriptografi secara komprehensif dan ditujukan terutama untuk pembaca yang ingin mendalami ilmu kriptografi.

Sebagai motivasi, konsep dasar enkripsi dibahas, termasuk konsep acak, *one-time pad*, *cryptanalysis* dan manajemen kunci rahasia. Selanjutnya berbagai teknik enkripsi klasik (simetris) dijelaskan, mulai dari yang sederhana sampai dengan teknik modern untuk *stream cipher* dengan contoh RC4 dan *block cipher* dengan contoh DES dan AES. Setiap teknik dianalisa kelemahannya, antara lain menggunakan *cryptanalysis*, yang hasilnya kemudian dijadikan motivasi untuk teknik yang lebih canggih. Berbagai teknik untuk kriptografi *public key* (enkripsi asimetris) juga dibahas, termasuk RSA, Diffie-Hellman, DSA, ElGamal, *knapsack* dan berbagai *zero knowledge protocol* termasuk Fiat-Shamir, Feige-Fiat-Shamir dan Guillou-Quisquater. *Secure hashing* dengan contoh MD5 dan SHA, dan *quantum key distribution* juga dibahas di buku ini.

Untuk setiap teknik enkripsi, matematika yang menjadi dasar teknik tersebut dijelaskan sebelumnya. Matematika yang dibahas buku ini antara lain aritmatika modular, *polynomial field*, teorema kecil Fermat, *finite field*, *quadratic residue* dan *algebraic number*. Dua algoritma penting yang umurnya ribuan tahun juga dibahas yaitu algoritma Euclid dan *Chinese remainder theorem*. Selain itu berbagai teknik untuk test bilangan prima (termasuk Solovay-Strassen dan Miller-Rabin), penguraian bilangan bulat (termasuk metode Rho, metode Dixon, *continued fraction*, *quadratic sieve* dan *number field sieve*) dan logaritma diskrit (termasuk metode Silver-Pohlig-Hellman, *baby steps - giant steps* dan *index calculus*) juga dijelaskan. Penguraian bilangan bulat dan logaritma diskrit sangat penting karena jika dapat dilakukan secara efisien maka berbagai teknik untuk kriptografi *public key* menjadi tidak berguna. *Elliptic curve* dan penggunaannya dalam kriptografi *public key* juga dijelaskan.

Berbagai aplikasi kriptografi dibahas buku ini, termasuk untuk pengamanan sesi (SSL/TLS, SSH dan IPsec), pengamanan email, *authentication* (Kerberos), *public key infrastructure* (PGP dan X.509) dan *cryptographic library* (OpenSSL, RSA BSafe dan Cryptlib).

Buku ini diahiri dengan pembahasan masa depan kriptografi, termasuk potensi dari *quantum computing*.

ISBN 978-602-96233-0-7

SPK IT
Consulting

9 786029 623307