

# Tutorial Dasar-dasar SageMath untuk Kriptografi Sederhana

Ari M. Barmawi, Muhammad Arzaki, Farah Afianti  
Laboratorium *Computing*, Fakultas Informatika, Universitas Telkom

*draft* versi 1.0 Agustus 2021

# Daftar Isi

<b>Daftar Isi</b>	i
<b>Daftar Gambar</b>	iii
<b>Daftar Tabel</b>	v
<b>Pengantar</b>	vi
<b>0 Instalasi SageMath</b>	1
0.1 Instalasi SageMath pada Windows . . . . .	1
0.1.1 Instalasi SageMath dengan Windows <i>Native Binary Installer</i> . . . . .	2
0.1.2 Instalasi SageMath dengan <i>Virtual Machine</i> pada Windows . . . . .	4
0.2 Instalasi SageMath pada Linux (Ubuntu) . . . . .	6
0.2.1 Instalasi SageMath pada Linux dengan <i>Binary Package</i> . . . . .	7
0.2.2 Instalasi SageMath pada Linux Melalui Terminal . . . . .	10
0.2.3 Instalasi SageMath pada Linux Melalui SnapStore . . . . .	10
0.3 Instalasi SageMath pada macOS . . . . .	11
<b>1 Dasar-dasar Python pada SageMath</b>	13
1.1 Perkenalan SageMath Notebook . . . . .	13
1.1.1 Pembuatan <i>file</i> Notebook Baru . . . . .	13
1.1.2 Antarmuka Notebook . . . . .	13
1.2 Perkenalan sintaks dasar Python . . . . .	15
1.2.1 Penugasan, Perbandingan, dan Operasi Dasar Aritmetika . . . . .	15
1.2.2 Pengkondisian . . . . .	18
1.2.3 Perulangan . . . . .	19
1.2.4 <i>List</i> dan <i>Tuple</i> . . . . .	20
1.2.5 Fungsi . . . . .	23
1.2.6 Beberapa Operasi Simbolis Sederhana pada SageMath . . . . .	24
<b>2 Dasar-dasar Aljabar pada SageMath</b>	29
2.1 Operasi-operasi Dasar pada Bilangan Bulat . . . . .	30
2.1.1 Teorema Pembagian . . . . .	30
2.1.2 Faktor Persekutuan Terbesar dan Kelipatan Persekutuan Terkecil . . . . .	32
2.1.3 Bilangan Prima dan Faktorisasi Prima . . . . .	35
2.2 Grup . . . . .	38
2.2.1 Definisi Grup dan Contohnya . . . . .	39
2.2.2 Grup Permutasi (Grup Simetri) . . . . .	40
2.2.3 Grup Siklis . . . . .	43
2.3 Dasar-dasar Ring dan <i>Field</i> pada SageMath . . . . .	44
2.3.1 Definisi Ring dan <i>Field</i> serta Beberapa Contohnya . . . . .	44

---

2.3.2	Ring $\mathbb{Z}$ dan $\mathbb{Z}_m$ pada SageMath . . . . .	45
2.3.3	<i>Prime Field</i> $\mathbb{Z}_p$ atau $\mathbb{F}_p$ pada SageMath . . . . .	51
2.4	Polinom atas Ring $\mathbb{Z}_m$ dan Polinom atas <i>Field</i> $\mathbb{F}_p$ pada SageMath . . . . .	52
2.4.1	Teorema Pembagian pada Ring Polinom . . . . .	54
2.4.2	gcd dan lcm pada Ring Polinom . . . . .	55
2.4.3	Akar, Polinom Tak Tereduksi, dan Faktorisasi Polinom . . . . .	57
2.5	<i>Field</i> $GF(p^k)$ atau $\mathbb{F}_{p^k}$ . . . . .	60
2.6	Vektor dan Matriks atas Ring dan <i>Field</i> pada SageMath . . . . .	62
2.6.1	Vektor, Modul, dan Ruang Vektor . . . . .	62
2.6.2	Matriks atas Ring dan <i>Field</i> Pada SageMath . . . . .	65
2.7	Kurva Eliptik ( <i>Elliptic Curve</i> ) pada SageMath . . . . .	69
<b>3</b>	<b>Contoh Kriptografi Klasik pada SageMath</b>	<b>73</b>
3.1	Domain Karakter dan Konversi dari Huruf ke Angka . . . . .	73
3.1.1	Domain Karakter Huruf Kapital . . . . .	73
3.1.2	Domain Karakter Heksadesimal . . . . .	74
3.1.3	Domain Karakter String Biner . . . . .	75
3.1.4	Domain Karakter Oktal . . . . .	75
3.1.5	Domain Karakter Radix-64 . . . . .	75
3.1.6	Konversi Huruf ke Angka Menggunakan ASCII . . . . .	76
3.2	<i>Substitution Cipher</i> (Sandi Substitusi) . . . . .	77
3.2.1	<i>Affine Cipher</i> (Sandi Affine) . . . . .	78
3.2.2	<i>Caesar Cipher</i> (Sandi Caesar atau Sandi Geser) . . . . .	81
3.2.3	<i>Decimation Cipher</i> . . . . .	83
3.2.4	<i>Hill Cipher</i> (Sandi Hill) . . . . .	83
3.2.5	<i>Vigenère Cipher</i> (Sandi Vigenère) . . . . .	88
3.3	<i>Transposition Cipher</i> (Sandi Transposisi) . . . . .	90
<b>4</b>	<b>Contoh Kriptografi Simetrik pada SageMath</b>	<b>94</b>
4.1	DES yang Disederhanakan ( <i>Simplified DES</i> ) . . . . .	94
4.2	Mini AES . . . . .	105
<b>5</b>	<b>Contoh Kriptografi Asimetrik pada SageMath</b>	<b>110</b>
5.1	Pertukaran Kunci Diffie-Hellman . . . . .	110
5.2	Pertukaran Kunci Diffie-Hellman dengan Kurva Eliptik . . . . .	115
5.3	Konversi String ke Bilangan Bulat dan Sebaliknya . . . . .	120
5.4	Sistemkripto RSA . . . . .	123
5.4.1	Protokol RSA untuk Enkripsi dan Dekripsi . . . . .	123
5.4.2	Skema Tanda Tangan Digital RSA . . . . .	128
5.4.3	Beberapa Serangan Elementer pada RSA . . . . .	133
5.5	Sistemkripto Elgamal . . . . .	135
<b>A</b>	<b>Repositori Skrip SageMath</b>	<b>140</b>
<b>Penutup</b>		<b>141</b>
<b>Daftar Pustaka</b>		<b>142</b>

# Daftar Gambar

0.1	Tampilan SageMathCell yang dapat diakses secara daring. . . . .	2
0.2	Tampilan awal instalasi SageMath dengan Windows <i>native binary installer</i> . . .	3
0.3	Ilustrasi persetujuan <i>License Agreement</i> pada instalasi SageMath di Windows 7.	3
0.4	Menentukan pengguna sistem yang dapat mengakses SageMath. . . . .	4
0.5	Ilustrasi pemilihan direktori instalasi SageMath pada Windows 7. . . . .	4
0.6	Rincian kebutuhan ruang penyimpanan pada instalasi SageMath di Windows dengan <i>native binary installer</i> . . . . .	5
0.7	Langkah-langkah awal untuk instalasi VirtualBox pada Windows. . . . .	6
0.8	Langkah awal untuk menjalankan ekstensi .ova. . . . .	6
0.9	Memilih <i>file</i> SageMath dalam ekstensi .ova. . . . .	7
0.10	Tampilan SageMath yang telah berhasil di- <i>install</i> dalam <i>virtual box</i> . . . . .	7
0.11	Informasi terkait lokasi <i>server</i> yang dapat digunakan untuk mengunduh <i>binary package</i> SageMath untuk Linux Ubuntu. Gambar diambil dari <a href="https://www.sagemath.org/download-linux.html">https://www.sagemath.org/download-linux.html</a> pada Juli 2021 . . . . .	8
0.12	Salah satu ilustrasi navigasi pada terminal menuju direktori SageMath di Ubuntu 20.04. . . . .	8
0.13	Tampilan SageMath ketika dijalankan dalam terminal pada Ubuntu 20.04. . . .	9
0.14	Contoh penggunaan <i>interpreter</i> terminal pada SageMath. Di sini operasi aritmetika yang dilakukan adalah menghitung nilai $x$ yang memenuhi $x = 2^3 + 4$ .	9
0.15	Cara untuk keluar dari <i>interpreter</i> SageMath pada terminal. . . . .	10
0.16	Cara memanggil Jupyter notebook untuk SageMath dari terminal pada Linux Ubuntu dengan mengetikkan sage -n jupyter. . . . .	10
0.17	Tampilan Jupyter notebook yang digunakan Linux Ubuntu. Tampilan serupa juga dapat dilihat ketika SageMath dijalankan di sistem operasi yang lain. . . .	11
0.18	Salah satu indikasi bahwa SageMath sudah ter- <i>install</i> melalui SnapStore pada Ubuntu 20.04. . . . .	11
0.19	Cara menjalankan SageMath dengan memilih ikon aplikasi yang terdapat pada Linux Ubuntu. . . . .	12
0.20	Informasi terkait lokasi <i>server</i> yang dapat digunakan untuk mengunduh <i>binary package</i> SageMath untuk macOS. Gambar diambil dari <a href="https://www.sagemath.org/download-mac.html">https://www.sagemath.org/download-mac.html</a> pada Juli 2021. . . . .	12
1.1	Cara membuat <i>file</i> baru pada <i>dashboard</i> Jupyter notebook. . . . .	14
1.2	Cara membuat <i>file</i> baru pada Jupyter notebook aktif. . . . .	14
1.3	Bagian-bagian notebook pada Jupyter notebook yang digunakan pada SageMath.	14
1.4	Ilustrasi interaksi <i>input</i> dari pengguna pada Jupyter notebook. . . . .	17
1.5	Contoh penggunaan fungsi show( . . . ) untuk menyelesaikan ekspresi matematika pada SageMath. . . . .	25

---

1.6	Contoh penggunaan fungsi <code>show(...)</code> untuk menyelesaikan ekspresi matematika pada SageMath. Perhatikan bahwa solusi dari suatu ekspresi matematika dapat berupa sebuah persamaan. . . . .	26
1.7	Hasil penggambaran grafik dari fungsi $f(x) = x^2$ untuk $-2 \leq x \leq 1$ dan $g(y) = -y + 1$ untuk $-2 \leq y \leq 1.5$ . . . . .	27
1.8	Hasil penggambaran grafik pada data dari sebuah <i>list</i> . . . . .	27
1.9	Hasil penggambaran grafik dari fungsi $f(x, y) = x^2 + 3y$ untuk $-5 \leq x, y \leq 5$ . . . . .	28
2.1	Cara memulai penggunaan SageMath pada sistem operasi Windows. . . . .	29
2.2	Penggunaan SageMath pada <i>shell</i> (gambar bagian kiri) dan <i>console</i> (gambar bagian kanan) pada sistem operasi Windows. . . . .	30
2.3	Salah satu cara membuat <i>file</i> SageMath baru pada Jupyter notebook. . . . .	30
2.4	Operasi div dan mod pada Jupyter notebook. Kita memiliki $15 \text{ div } 4 = 3$ , $15 \text{ mod } 4 = 3$ , $-15 \text{ div } 4 = -4$ , $-15 \text{ mod } 4 = 1$ . . . . .	31
2.5	Tampilan hasil dari fungsi <code>show (A)</code> dan <code>show (B)</code> untuk dua matriks <b>A</b> dan <b>B</b> . . . . .	66
4.1	Ilustrasi satu putaran sistem Feistel yang digunakan pada SDES. . . . .	95
4.2	Ilustrasi komputasi fungsi $f(R_{i-1}, K_i)$ pada (4.1). . . . .	97
4.3	Ilustrasi fungsi pengembang ( <i>expander function</i> ) yang digunakan pada definisi fungsi $f$ . . . . .	97
4.4	Ilustrasi masukan dan keluaran fungsi <code>SDES_Enc (P, K)</code> . . . . .	104
4.5	Ilustrasi masukan dan keluaran fungsi <code>SDES_Dec (P, K)</code> . . . . .	104
4.6	Ilustrasi cara mengubah pesan $P$ ke dalam matriks <b>P</b> yang berukuran $2 \times 2$ atas $GF(16)$ . . . . .	105
4.7	Satu putaran ekripsi pada mini AES. . . . .	106
4.8	Contoh keluaran simulasi proses enkripsi dan dekripsi pada mini AES. . . . .	108
4.9	Ilustrasi masukan dan keluaran fungsi <code>maes_enc (P, K)</code> . . . . .	109
4.10	Ilustrasi masukan dan keluaran fungsi <code>maes_dec (P, K)</code> . . . . .	109

# Daftar Tabel

1.1	Notasi atau sintaks operasi aritmetika pada SageMath. . . . .	16
1.2	Contoh hasil dari Latihan 1.2.1. . . . .	17
1.3	Contoh hasil dari Latihan 1.2.2. . . . .	19
1.4	Contoh hasil dari Latihan 1.2.4. . . . .	20
1.5	Perintah dasar untuk <i>list</i> pada SageMath. . . . .	21
1.6	Contoh hasil dari Latihan 1.2.5. . . . .	22
1.7	Perintah dasar untuk <i>tuple</i> pada SageMath. . . . .	22
1.8	Contoh hasil dari Latihan 1.2.6. . . . .	23
1.9	Contoh hasil dari Latihan 1.2.7. . . . .	24

# Pengantar

Kriptografi modern merupakan suatu bidang keilmuan yang melibatkan banyak aspek multidisiplin, seperti matematika, ilmu komputer, dan teknik elektro. Saat ini ada banyak referensi yang dapat digunakan untuk mempelajari kriptografi, namun hanya sedikit referensi berbahasa Indonesia yang memberikan penjelasan kriptografi menggunakan tutorial simulasi komputasi yang terkait dengannya.

Tutorial ini membahas penggunaan SageMath untuk dasar-dasar kriptografi sederhana. Sebagian dari skrip simulasi komputasi dirangkum dari [1]. Meskipun tutorial ini juga membahas dasar-dasar teori yang digunakan pada kriptografi, tutorial ini bukanlah dokumen yang tepat untuk digunakan sebagai referensi utama. Pembaca yang tertarik mempelajari kriptografi secara lebih dalam dapat memulainya dengan membaca buku seperti [2], [3], atau [4]. Tutorial ini lebih ditujukan sebagai pendamping dalam pembelajaran kriptografi yang dilakukan secara mandiri maupun terstruktur di tingkat sarjana di Indonesia. Meskipun tidak ada prasyarat khusus untuk dapat memahami dokumen tutorial ini dengan baik, pengetahuan dasar mengenai pemrograman terstruktur atau prosedural, teori bilangan, struktur aljabar sederhana, maupun aljabar linier elementer akan sangat membantu pembaca pada pembuatan program maupun pemahaman teori.

Tutorial ini membahas beberapa simulasi dalam kriptografi sederhana menggunakan SageMath. SageMath, sebelumnya dikenal dengan SAGE (akronim dari *System for Algebra and Geometry Experimentation*), merupakan sebuah sistem aljabar komputasional (*Computer Algebra System*, CAS) yang digunakan untuk banyak bidang matematika, seperti aljabar, kombinatorika, teori graf, analisis numerik, teori bilangan, dan statistika [5]. SageMath sudah dikembangkan sejak tahun 2005 dan ditujukan sebagai alternatif perangkat lunak yang dapat digunakan secara bebas (gratis) dan berprinsip sumber terbuka (*open source*) dari perangkat-perangkat lunak lain yang lebih dulu ada, seperti Magma, Maple, Mathematica, dan MATLAB.

Berbeda dengan pendahulunya, SageMath tidak menggunakan bahasa yang dibangun sendiri, melainkan menggunakan Python sebagai bahasa pemrograman utamanya. Penggunaan Python dan sifatnya yang bersumber terbuka membuat *library* yang digunakan pada SageMath terus berkembang, salah satunya adalah *library* yang digunakan untuk kriptografi. Python juga merupakan salah satu bahasa pemrograman tingkat tinggi yang mudah digunakan karena sifatnya yang menggunakan *dynamic typing*. Sintaks pada Python tidak serumit bahasa-bahasa pemrograman lain seperti C, C++, maupun Java. Berdasarkan hasil survei pada <https://statisticstimes.com/tech/top-computer-languages.php>, Python merupakan bahasa pemrograman yang paling populer di dunia pada tahun 2021.

Tutorial ini dibagi menjadi enam topik utama. Topik 0 membahas instalasi SageMath pada sistem operasi Windows, Linux, maupun MacOS. Meskipun awalnya SageMath dikembangkan di sistem operasi berbasis Unix, sejak tahun 2017 SageMath versi 8.0 dapat dijalankan secara *native* pada sistem operasi Windows. Topik 1 mengulas dasar-dasar pemrograman dengan Python pada SageMath. Pembaca yang sudah terbiasa menggunakan Python dapat langsung membaca Subtopik 1.2.6 mengenai operasi simbolis pada SageMath yang merupakan salah satu aspek yang membedakan Python 3 biasa dengan SageMath. Kajian mengenai dasar-dasar aljabar yang digunakan pada kriptografi klasik maupun kriptografi modern berikut penerapannya dalam SageMath dibahas pada Topik 2. Topik ini mengulas teori bilangan elementer, definisi

---

grup, definisi ring dan *field*, polinom (atau suku banyak), vektor dan matriks, serta kurva eliptik. Pada Topik 3 dibahas mengenai penerapan SageMath untuk kriptografi klasik. Pembahasan meliputi sandi *affine* beserta variannya, sandi Hill, sandi Vigenére, dan sandi transposisi. Topik 4 membahas penyederhanaan dari dua contoh sistemkripto simetrik modern, yaitu DES dan AES. Tutorial ini membahas implementasi DES yang disederhanakan (*simplified DES*, SDES) dan mini AES. Topik 5 yang merupakan topik terakhir mendiskusikan tentang simulasi pertukaran kunci dan sistemkripto asimetrik pada SageMath. Bahasan pada topik ini meliputi kajian tentang pertukaran kunci Diffie-Hellman dan variannya, sistemkripto RSA, dan sistemkripto Elgamal.

Penyusun berterima kasih kepada para dosen maupun taruna dan taruni Politeknik Siber dan Sandi Negara atas diskusi yang bermanfaat dalam pembuatan tutorial ini. Pembaca yang menemukan kesalahan pada dokumen ini atau ingin memberikan masukan dapat mengirimkan email ke [arzaki@telkomuniversity.ac.id](mailto:arzaki@telkomuniversity.ac.id).

# Topik 0

## Instalasi SageMath

Pada topik ini akan dijelaskan mengenai instalasi SageMath berdasarkan dokumentasi yang tersedia pada [6]. Pengguna yang berencana menggunakan SageMath secara intensif untuk kegiatan riset terkini sangat dianjurkan untuk melakukan instalasi pada sistem operasi Linux. Versi Linux yang didukung setidaknya harus Debian versi 9 atau Ubuntu versi 18.04.

Penjelasan instalasi SageMath pada dokumen ini dibagi menjadi tiga bagian, yaitu instalasi pada sistem operasi Windows (dijelaskan pada Subtopik 0.1), instalasi pada sistem operasi Linux (dijelaskan pada Subtopik 0.2), dan instalasi pada sistem operasi macOS (dijelaskan pada Subtopik 0.3). SageMath versi terbaru dapat berjalan dengan baik pada sistem-sistem operasi yang telah disebutkan yang didukung oleh prosesor IA-32, x86-64, ARM, Itanium, maupun SPARC [5]. Baik pada sistem operasi Windows, Linux, maupun macOS, sebelum instalasi pengguna sangat disarankan untuk menyiapkan beberapa hal penting berikut:

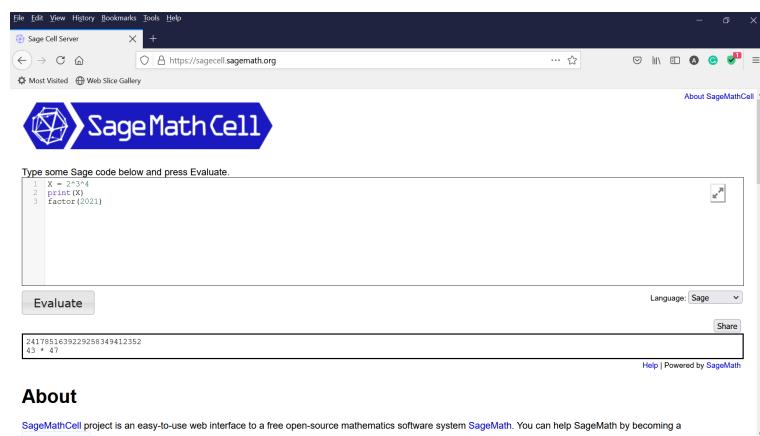
1. Jika instalasi dilakukan menggunakan *pre-built binaries*, maka ruang penyimpanan yang diperlukan setidaknya adalah 4 GB [6].
2. Jika instalasi dilakukan dari *source code*, maka ruang penyimpanan minimal yang diperlukan adalah 6 GB [6].
3. Hampir pada semua kondisi SageMath dapat berjalan dengan RAM minimal 2 GB, namun sangat disarankan untuk menggunakan setidaknya RAM 4 GB. Jika SageMath digunakan secara *multitasking*, maka pengguna disarankan memakai setidaknya RAM sebesar 8 GB.

Pengguna yang belum berencana melakukan instalasi SageMath namun ingin mencobanya dapat menggunakan SageMathCell yang tersedia secara daring di <https://sagecell.sagemath.org/>. Tampilan dari SageMathCell dapat dilihat pada Gambar 0.1. Perlu diingat bahwa proses komputasi pada SageMathCell terbatas dan bergantung pada kondisi jaringan Internet yang dipakai.

### 0.1 Instalasi SageMath pada Windows

Pada bagian ini akan dibahas dua metode instalasi SageMath pada sistem operasi Microsoft Windows, yaitu:

1. instalasi menggunakan Windows *native binary installer* (.exe), hal ini dijelaskan pada Subtopik 0.1.1,
2. instalasi menggunakan aplikasi tambahan seperti *virtual machine* atau *virtual box*, hal ini dijelaskan pada Subtopik 0.1.2.



Gambar 0.1: Tampilan SageMathCell yang dapat diakses secara daring.

Untuk kriptografi dasar dan sederhana, instalasi SageMath menggunakan cara pertama (Windows *native binary installer*) sudah cukup.

### 0.1.1 Instalasi SageMath dengan Windows *Native Binary Installer*

Sebelum melakukan instalasi SageMath dengan Windows *native binary installer*, kita harus memastikan bahwa perangkat yang kita pakai memenuhi persyaratan berikut:

1. Sistem operasi yang digunakan minimal Windows 7 64-bit (atau lebih baru). **Instalasi SageMath dengan Windows *native binary installer* tidak dapat dilakukan pada sistem operasi Windows 32-bit.**
2. Perangkat yang digunakan memiliki ruang penyimpanan (*storage*) minimal 5 GB (tidak termasuk *file native binary installer* yang diunduh maupun *file* pekerjaan yang akan disimpan).

Uji coba instalasi SageMath dengan Windows *native binary installer* sudah dilakukan pada Windows 7 maupun Windows 10. Versi terbaru yang cukup stabil dari SageMath (per bulan Juli 2021), yaitu SageMath versi 9.2 (Windows *installer* 0.6.2) yang berukuran 815 MB dapat diunduh pada tautan

<https://github.com/sagemath/sage-windows/releases/download/0.6.2-9.2/SageMath-9.2-Installer-v0.6.2.exe>

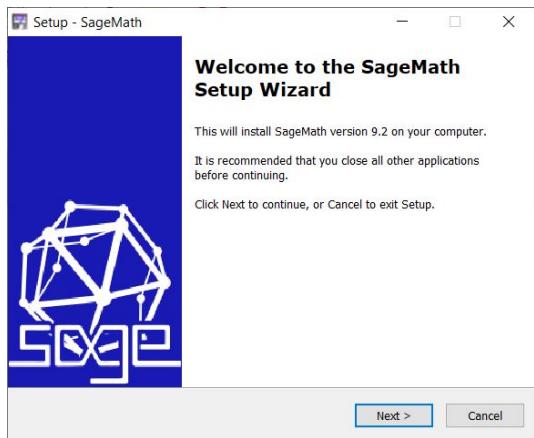
Berikut adalah langkah-langkah instalasi SageMath dengan Windows *native binary installer*:

1. Jalankan *file* instalasi yang terunduh.

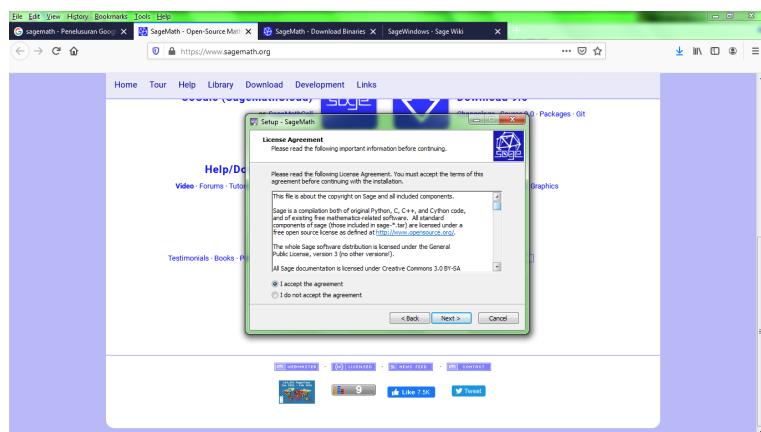
*File* instalasi dapat dijalankan dengan cara mengeklik ganda atau melakukan klik kanan lalu pilih *open* atau *Run as administrator*. Secara *default* instalasi SageMath adalah untuk pengguna tunggal (*single user*) sehingga tidak membutuhkan izin dari *administrator* pada sistem operasi Windows. Pada perangkat yang memiliki banyak pengguna (*multi user*) SageMath dapat di-*install* untuk semua pengguna asalkan instalasi mendapat izin *administrator*. Hal ini dapat dilakukan dengan cara klik kanan pada *file* instalasi lalu pilih “Jalankan sebagai Administrator (*Run as Administrator*)” hingga tampil jendela seperti pada Gambar 0.2, kemudian pilih *Next* untuk menuju ke halaman *License Agreement*.

2. Menyetujui *License Agreement* dan memilih mode penggunaan SageMath.

Untuk dapat memakai SageMath, kita harus menyetujui *License Agreement* sebagaimana



Gambar 0.2: Tampilan awal instalasi SageMath dengan Windows *native binary installer*.



Gambar 0.3: Ilustrasi persetujuan *License Agreement* pada instalasi SageMath di Windows 7.

dilakukan pada Gambar 0.3. Setelah menyetujui berbagai kondisi dari SageMath yang akan di-*install*, instalasi dilanjutkan dengan pemilihan pengguna yang dapat mengakses SageMath sebagaimana dijelaskan pada Gambar 0.4. Jika kita menginginkan SageMath dapat diakses oleh semua pengguna maka kita memilih “*Install for all users (installer must be run with Administrator privileges)*”. Akan tetapi jika kita membatasi pengguna SageMath pada pengguna yang melakukan instalasi saja maka kita memilih “*Install just for the current user*”.

### 3. Menentukan direktori penyimpanan SageMath.

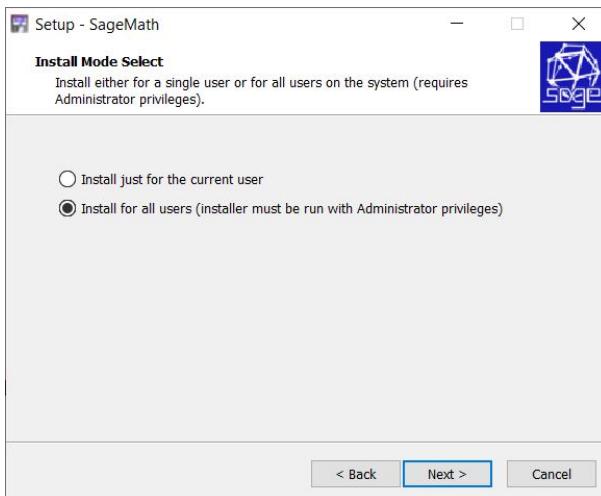
Secara *default* SageMath akan di-*install* pada direktori C:\Program File \SageMath <Versi>. Namun jika ruang penyimpanan *default* tidak terlalu besar atau alasan lain, SageMath dapat di-*install* pada direktori yang lain atau *hard-drive* sekunder (misalnya pada *drive D*:). Kita cukup mengubah direktori instalasi dari SageMath sesuai dengan yang diinginkan sebagaimana dicontohkan pada Gambar 0.5.

### 4. Melakukan pemilihan pada komponen yang akan di-*install*.

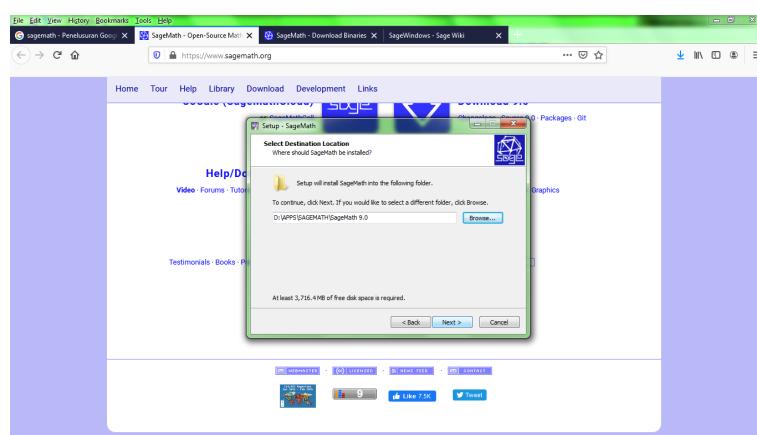
Rincian kebutuhan penyimpanan aplikasi SageMath ditunjukkan pada Gambar 0.6. Pengguna memerlukan minimal 4 GB untuk SageMath Core untuk SageMath versi 9.2. Dokumentasi HTML untuk bantuan penggunaan dari sintaks SageMath dapat tidak dipilih untuk menghemat ruang penyimpanan. Dalam hal ini dokumentasi HTML ini tetap dapat diakses secara daring selama ada akses Internet.

### 5. Melakukan proses instalasi dan menambahkan ikon *shortcut*.

Proses instalasi membutuhkan waktu beberapa menit. Setelah instalasi selesai pilihan un-



Gambar 0.4: Menentukan pengguna sistem yang dapat mengakses SageMath.



Gambar 0.5: Ilustrasi pemilihan direktori instalasi SageMath pada Windows 7.

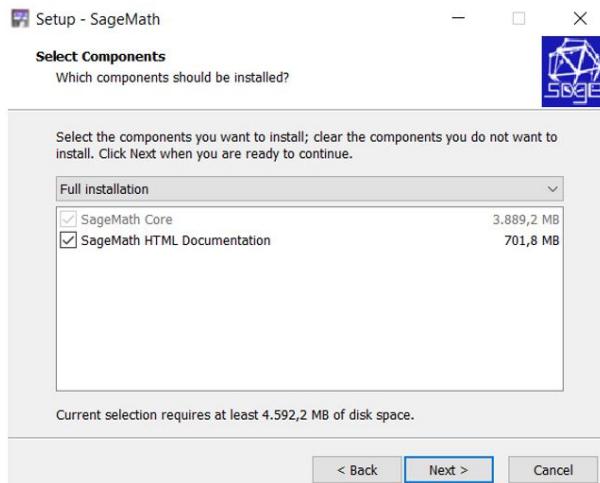
tuk menambahkan ikon *shortcut* pada *desktop* maupun *start menu* akan muncul. *Shortcut* dari aplikasi SageMath juga dapat ditemukan di lokasi instalasi SageMath yang secara *default* terletak di C:\Program File \SageMath <Versi>.

### 0.1.2 Instalasi SageMath dengan *Virtual Machine* pada Windows

Instalasi SageMath dengan *virtual machine* pada Windows menjadi alternatif apabila versi Windows yang digunakan adalah Windows 32-bit. Selain itu instalasi SageMath dengan cara ini menjadi alternatif ketika RAM yang ada pada komputer yang digunakan tidak terlalu besar (di bawah 4 GB). Sebelum melakukan instalasi, kita perlu memastikan bahwa perangkat yang digunakan memenuhi syarat-syarat berikut:

1. Sistem operasi yang digunakan adalah Windows 7 32-bit atau lebih baru. Instalasi SageMath dengan *virtual machine* juga dapat dijalankan pada sistem operasi Windows 64-bit. Namun, ketika pengguna memakai sistem operasi Windows 64-bit, maka instalasi SageMath dengan *native binary installer* sebagaimana dijelaskan pada Subtopik 0.1.1 lebih disarankan.
2. RAM yang ada pada komputer minimal 512 MB.

Instalasi SageMath pada Windows dengan *virtual machine* memerlukan dua aplikasi terpisah. Pertama kita memerlukan *virtual machine*, salah satu aplikasi *virtual machine* yang cukup



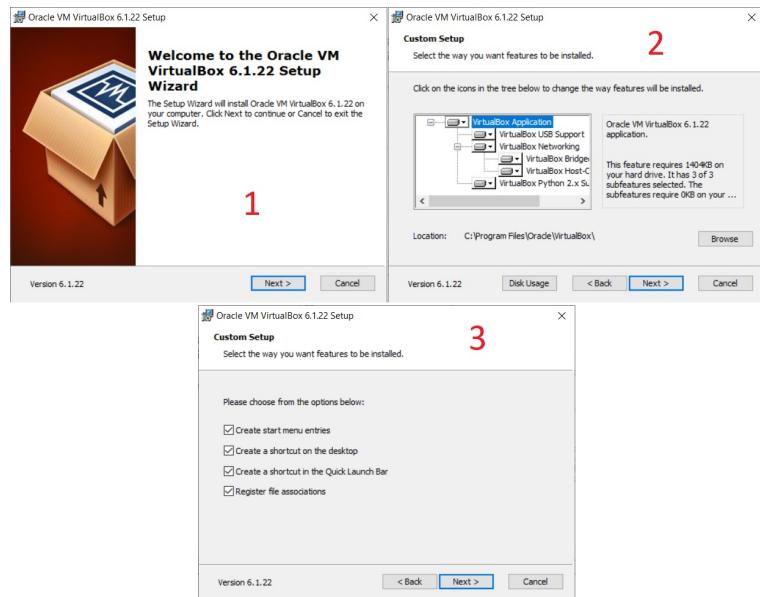
Gambar 0.6: Rincian kebutuhan ruang penyimpanan pada instalasi SageMath di Windows dengan *native binary installer*.

mudah digunakan adalah VirtualBox versi 6.1.22 dengan ukuran 103 MB yang dapat diunduh melalui tautan <https://download.virtualbox.org/virtualbox/6.1.22/VirtualBox-6.1.22-144080-Win.exe>. Aplikasi kedua yang diperlukan adalah aplikasi SageMath yang akan dijalankan melalui *virtual machine* dalam ekstensi .ova. Format nama file yang digunakan biasanya adalah sage-x.y.z.ova. Untuk memperolehnya kita dapat melakukan salah satu di antara cara-cara berikut:

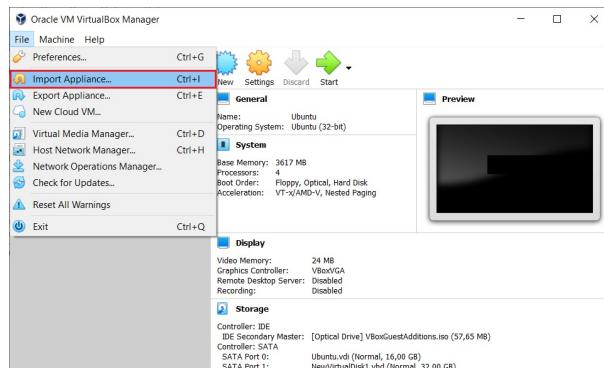
1. Mengunduh berkas .ova secara langsung, misalnya pada <https://mirror-hk.koddos.net/sagemath/ova/sage-8.1.ova>.
2. Menggunakan *torrent* (<https://www.bittorrent.com/downloads/complete/>). Mohon diingat bahwa dalam penggunaan *torrent* kita mungkin perlu mematikan antivirus yang digunakan atau melakukan pengaturan tertentu pada *browser* yang digunakan. Beberapa tautan yang dapat digunakan dalam cara ini adalah:
  - (a) <http://mirror.aarnet.edu.au/pub/sage/torrents.html>,
  - (b) <http://mirror.aarnet.edu.au/pub/sage/ova/meta/sage-8.1.ova.torrent>.

Langkah-langkah instalasi SageMath pada Windows dengan *virtual machine* dapat dilakukan sebagai berikut:

1. Lakukan instalasi *virtual machine* (VirtualBox)  
Pilih “Next” di setiap langkah sesuai penjelasan pada Gambar 0.7 untuk contoh detail. Jika langkah-langkah tersebut dilakukan dengan benar, maka VirtualBox secara *default* ter-install pada C:\Program Files \Oracle \VirtualBox \.
2. Menjalankan ekstensi .ova dengan VirtualBox.  
Untuk menjalankan ekstensi .ova dengan VirtualBox kita dapat memilih *File* lalu *Import Appliance...* sebagaimana diilustrasikan pada Gambar 0.8.
3. Memilih file SageMath dalam ekstensi .ova yang sebelumnya sudah diunduh.  
Untuk memilih file SageMath dalam ekstensi .ova, kita dapat memilih *File* lalu ikon *Folder* sebagaimana dijelaskan pada Gambar 0.9. Arahkan pemilihan *folder* pada file sage-x.y.z.ova yang telah diunduh sebelumnya.



Gambar 0.7: Langkah-langkah awal untuk instalasi VirtualBox pada Windows.



Gambar 0.8: Langkah awal untuk menjalankan ekstensi .ova.

#### 4. Menyelesaikan proses instalasi dan menjalankan SageMath pada VirtualBox.

Ikuti pengaturan *default* dari VirtualBox untuk memulai instalasi SageMath. Setelah berhasil, SageMath akan tampak seperti sebuah mesin di dalam VirtualBox yang perlu dinyalakan setiap akan digunakan seperti yang tampak pada Gambar 0.10.

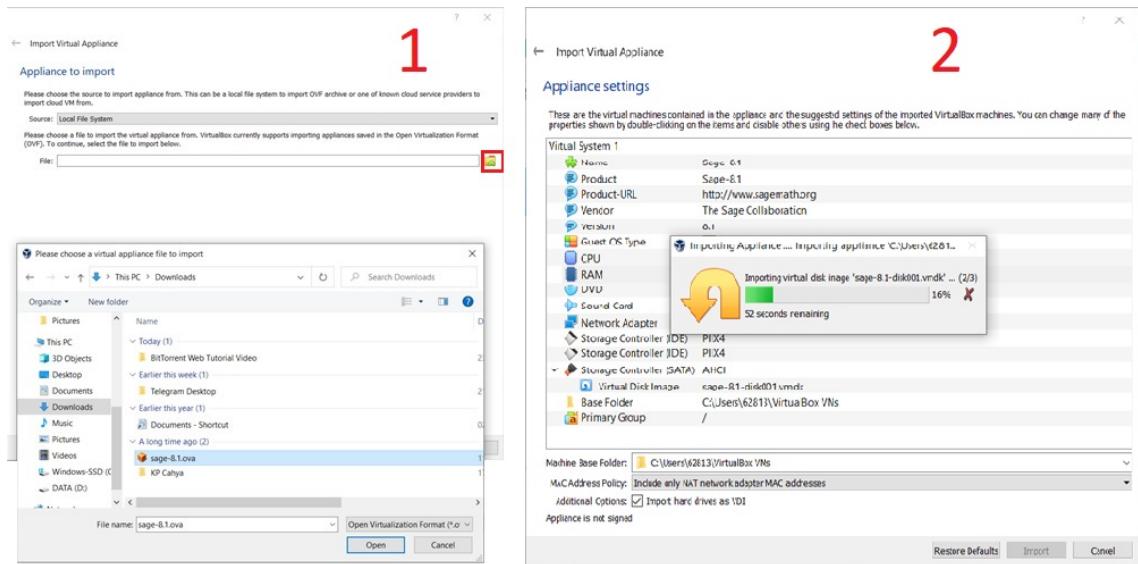
**Latihan 0.1.1** Jika Anda menggunakan Windows sebagai sistem operasi yang digunakan sehari-hari, lakukan instalasi SageMath pada Windows.

## 0.2 Instalasi SageMath pada Linux (Ubuntu)

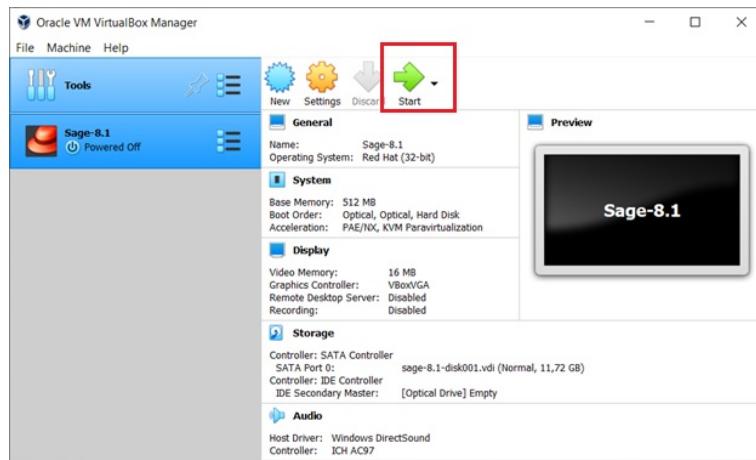
Sebelum tahun 2016 SageMath tidak dapat dijalankan secara *native* pada Windows. Ketika itu satu-satunya cara untuk menjalankan SageMath pada Windows adalah menggunakan *virtual machine* (yang menjalankan sistem operasi berbasis Linux). Oleh karenanya, salah satu pilihan alternatif yang cukup banyak digunakan adalah melakukan instalasi SageMath (secara *native*) pada sistem berbasis Unix, misalnya Linux Ubuntu.

Instalasi SageMath pada Linux Ubuntu atau distro Linux yang lain dapat dilakukan menggunakan *binary package* yang tersedia pada tautan

<https://www.sagemath.org/download-linux.html>.



Gambar 0.9: Memilih file SageMath dalam ekstensi .ova.



Gambar 0.10: Tampilan SageMath yang telah berhasil di-install dalam virtual box.

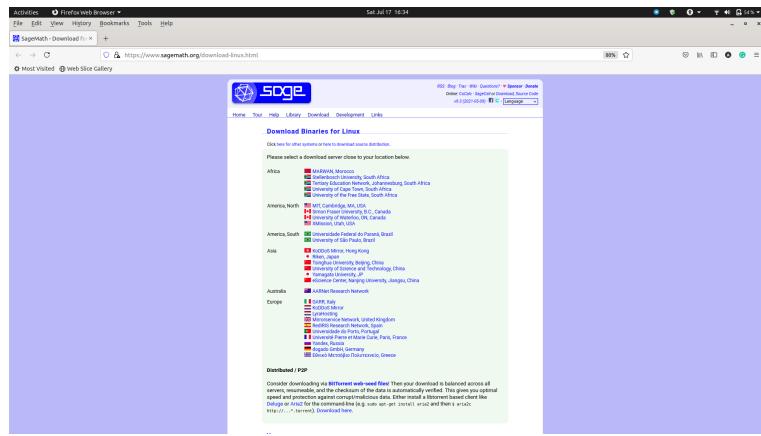
*Binary package* dapat diunduh dari *server* yang terdekat dengan lokasi kita dari pilihan-pilihan yang ada sebagaimana dijelaskan pada Gambar 0.11. Versi Linux minimal yang mendukung SageMath versi 9 (yang relatif baru pada tahun 2021) adalah Debian versi 9 dan Linux Ubuntu versi 18.04.

Kita dapat mengunduh *binary package*, menyimpannya pada direktori yang diinginkan, melakukan ekstraksi, dan menjalankannya. Meskipun demikian mungkin terdapat beberapa dependensi (misalnya dengan Python 3) yang harus diperiksa dan disesuaikan terlebih dulu. Salah satu tutorial instalasi SageMath versi 9.0 untuk Linux Ubuntu versi 20.04 yang dapat diikuti dapat dilihat di <https://www.youtube.com/watch?v=obtYxRDWAqU>.

Pada tutorial ini akan dijelaskan tiga cara instalasi SageMath pada Ubuntu, yaitu menggunakan *binary package* (dijelaskan pada Subtopik 0.2.1), menggunakan terminal (dijelaskan pada Subtopik 0.2.2), dan menggunakan SnapStore (dijelaskan pada Subtopik 0.2.3).

## 0.2.1 Instalasi SageMath pada Linux dengan *Binary Package*

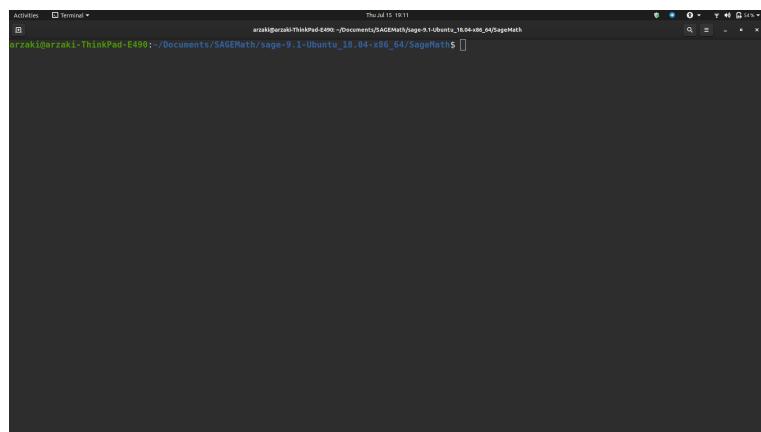
Pada bagian ini akan dijelaskan instalasi SageMath dengan *binary package* yang diunduh pada <https://www.sagemath.org/download-linux.html>. Instalasi sudah diuji coba pada Linux Ubuntu 20.04. Perlu diingat bahwa versi Linux yang didukung untuk Sage-



Gambar 0.11: Informasi terkait lokasi *server* yang dapat digunakan untuk mengunduh *binary package* SageMath untuk Linux Ubuntu. Gambar diambil dari <https://www.sagemath.org/download-linux.html> pada Juli 2021

Math versi 9.0 atau terbaru minimal adalah Debian versi 9 atau Ubuntu versi 18.04. Langkah-langkah instalasi adalah sebagai berikut:

1. Unduh *binary package* dari <https://www.sagemath.org/download-linux.html> dan simpan pada direktori yang diinginkan.
2. Lakukan ekstraksi dari berkas SageMath yang diperoleh pada langkah 1.
3. Pada awalnya SageMath menggunakan Python versi 2 hingga September 2019. Sejak September 2019 SageMath versi 8.9 berjalan menggunakan Python 3. Ada baiknya Python 3 sudah terpasang pada Linux Ubuntu sebelum SageMath di-*install*. Selain itu ada baiknya Python 2 tidak dipasang pada komputer bila memang tidak digunakan (untuk mencegah adanya konflik yang diakibatkan karena versi Python yang berbeda).
4. Untuk menjalankan SageMath, kita dapat menggunakan terminal dalam melakukan navigasi ke direktori yang telah ditentukan. Misalkan SageMath disimpan pada direktori berikut: `Documents/SAGEMath/sage-9.1-Ubuntu_18.04-x86_64/` pada Linux Ubuntu, maka kita dapat melakukan navigasi seperti pada Gambar 0.12.



Gambar 0.12: Salah satu ilustrasi navigasi pada terminal menuju direktori SageMath di Ubuntu 20.04.

- 
5. Kita dapat menjalankan SageMath dengan menambahkan perintah `./sage` pada direktori tersebut. Jika instalasi dilakukan dengan benar, maka kita dapat melihat tampilan seperti pada Gambar 0.13.

```
Activities Terminal Thu Jul 15 19:14
arzaki@arzaki-ThinkPad-E490: ~/Documents/SAGEMath/sage-9.1-Ubuntu_18.04-x86_64/SageMath$ ./sage
SageMath version 9.1, Release Date: 2020-05-20
Using Python 3.7.3. Type "help()" for help.

sage: 
```

Gambar 0.13: Tampilan SageMath ketika dijalankan dalam terminal pada Ubuntu 20.04.

6. Pada langkah 5 *interpreter* SageMath sudah siap untuk dipakai. Untuk memeriksanya kita dapat melakukan operasi aritmetika sederhana dengan sintaks Python 3 sebagaimana diilustrasikan pada Gambar 0.14.

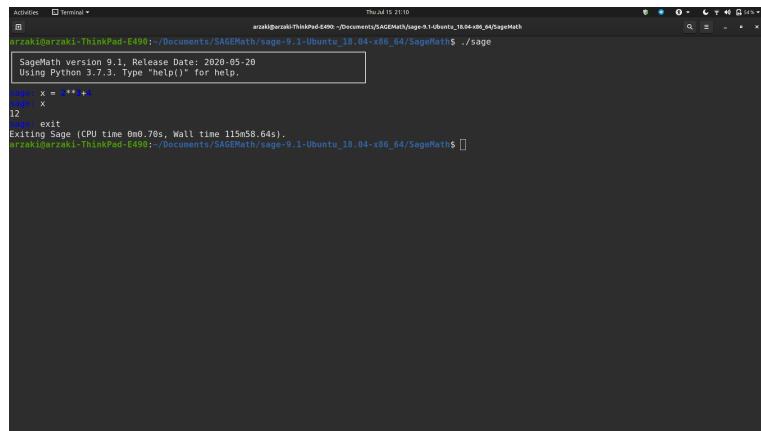
```
Activities Terminal Thu Jul 15 19:16
arzaki@arzaki-ThinkPad-E490: ~/Documents/SAGEMath/sage-9.1-Ubuntu_18.04-x86_64/SageMath$ ./sage
SageMath version 9.1, Release Date: 2020-05-20
Using Python 3.7.3. Type "help()" for help.

sage: x = 2^3 + 4
12
sage: 
```

Gambar 0.14: Contoh penggunaan *interpreter* terminal pada SageMath. Di sini operasi aritmetika yang dilakukan adalah menghitung nilai  $x$  yang memenuhi  $x = 2^3 + 4$ .

7. Untuk keluar dari *interpreter*, kita dapat mengetikkan perintah `exit` pada *interpreter* sebagaimana dijelaskan pada Gambar 0.15.
8. Untuk menjalankan Jupyter notebook pada SageMath kita dapat mengetikkan perintah `sage -n jupyter` pada terminal di direktori instalasi SageMath sebagaimana dijelaskan pada Gambar 0.16.
9. Salah satu contoh tampilan Jupyter notebook dapat dilihat pada Gambar 0.17. Ketika Jupyter notebook digunakan, maka secara *default* alamat URL halaman utamanya adalah `http://localhost:8888/tree`.

**Catatan 0.2.1** *Instalasi SageMath secara penuh (instalasi untuk semua library yang dapat dipakai pada Linux Ubuntu) untuk SageMath versi 9.2 memerlukan tempat penyimpanan lebih dari 11 GB (sekitar 11.7 GB).*

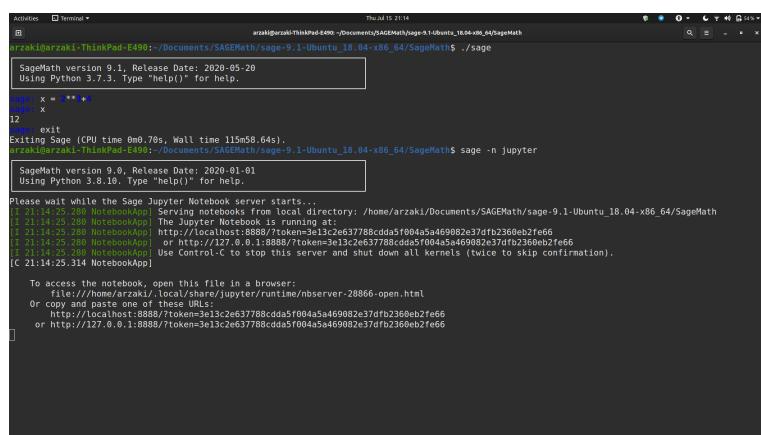


```

arzaki@arzaki-ThinkPad-E490:~/Documents/SAGEMath/sage-9.1-Ubuntu_18.04-x86_64/SageMath$ sage
SageMath version 9.1, Release Date: 2020-05-20
Using Python 3.7.3. Type "help()" for help.

sage: x = 2^3^4
sage: x
12
sage: exit
Exiting Sage (CPU time 0m0.70s, Wall time 115m58.64s).
arzaki@arzaki-ThinkPad-E490:~/Documents/SAGEMath/sage-9.1-Ubuntu_18.04-x86_64/SageMath$ 
```

Gambar 0.15: Cara untuk keluar dari *interpreter* SageMath pada terminal.



```

arzaki@arzaki-ThinkPad-E490:~/Documents/SAGEMath/sage-9.1-Ubuntu_18.04-x86_64/SageMath$ sage -n jupyter
SageMath version 9.1, Release Date: 2020-05-20
Using Python 3.8.10. Type "help()" for help.

sage: x = 2^3^4
sage: x
12
sage: exit
Exiting Sage (CPU time 0m0.70s, Wall time 115m58.64s).
arzaki@arzaki-ThinkPad-E490:~/Documents/SAGEMath/sage-9.1-Ubuntu_18.04-x86_64/SageMath$ sage -n jupyter
SageMath version 9.0, Release Date: 2020-01-01
Using Python 3.8.10. Type "help()" for help.

Please wait while the Sage Jupyter Notebook server starts...
[21:14:25.288 NotebookApp] Serving notebooks from local directory: /home/arzaki/Documents/SAGEMath/sage-9.1-Ubuntu_18.04-x86_64/SageMath
[21:14:25.288 NotebookApp] The Jupyter Notebook is running at:
[21:14:25.288 NotebookApp] http://127.0.0.1:8888/?token=3e13c2e637788ccda5f004a5a4d9082e37dfb2360eb2fe66
[21:14:25.288 NotebookApp] or http://127.0.1:8888/?token=3e13c2e637788ccda5f004a5a4d9082e37dfb2360eb2fe66
[21:14:25.288 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[21:14:25.314 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/arzaki/.local/share/jupyter/runtime/notebook-28866-open.html
Or copy and paste one of these URLs:
http://127.0.0.1:8888/?token=3e13c2e637788ccda5f004a5a4d9082e37dfb2360eb2fe66
or http://127.0.1:8888/?token=3e13c2e637788ccda5f004a5a4d9082e37dfb2360eb2fe66

```

Gambar 0.16: Cara memanggil Jupyter notebook untuk SageMath dari terminal pada Linux Ubuntu dengan mengetikkan sage -n jupyter.

## 0.2.2 Instalasi SageMath pada Linux Melalui Terminal

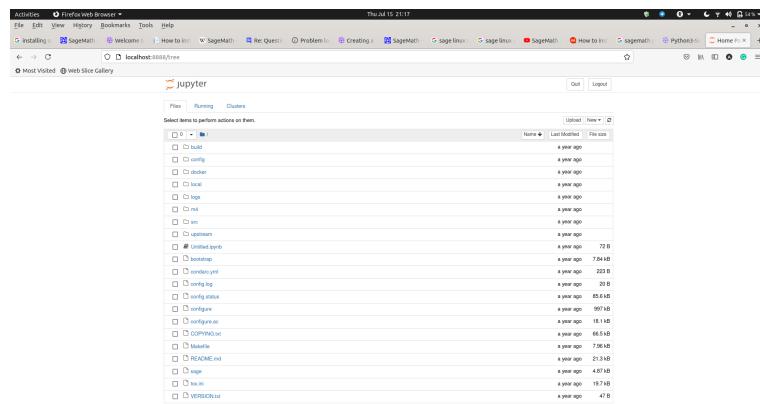
Instalasi SageMath melalui terminal pada Linux Ubuntu cukup mudah. Hal ini dapat dilakukan pada langkah-langkah berikut:

1. Jika pengguna komputer tidak menggunakan Python 2 sebaiknya Python 2 tidak perlu dipasang sebelum SageMath di-*install*. Hal ini untuk mengurangi risiko konflik versi Python yang mungkin terjadi.
2. Pada *interpreter* ketik:  
`sudo apt install sagemath`  
 untuk meng-*install* seluruh *dependencies* yang diperlukan atau cukup  
`sudo apt install sagemath-common`  
 untuk meng-*install* *dependencies* SageMath yang umum digunakan (tidak meng-*install* seluruh *dependencies* yang diperlukan).

## 0.2.3 Instalasi SageMath pada Linux Melalui SnapStore

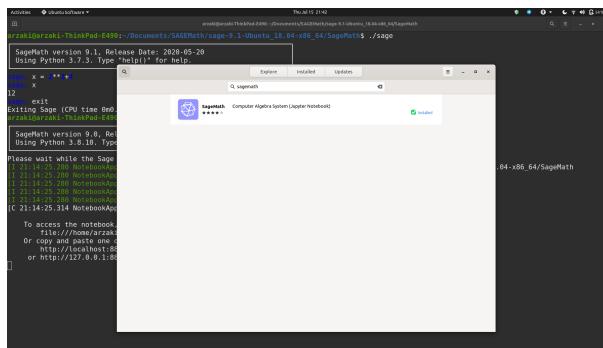
Instalasi SageMath melalui SnapStore dimungkinkan pada Linux Ubuntu 20.04 dengan langkah-langkah berikut:

1. Buka SnapStore pada Ubuntu dan ketik SageMath.



Gambar 0.17: Tampilan Jupyter notebook yang digunakan Linux Ubuntu. Tampilan serupa juga dapat dilihat ketika SageMath dijalankan di sistem operasi yang lain.

2. Instalasi dapat dilakukan dengan mengeklik ikon instalasi SageMath.
3. Jika SageMath sudah ter-*install*, maka salah satu indikasinya diilustrasikan pada Gambar 0.18.



Gambar 0.18: Salah satu indikasi bahwa SageMath sudah ter-*install* melalui SnapStore pada Ubuntu 20.04.

4. Untuk menjalankan SageMath, kita dapat mengetik SageMath dan memilih ikon yang bersesuaian pada aplikasi yang ter-*install* di Ubuntu sebagaimana diilustrasikan pada Gambar 0.19.

**Catatan 0.2.2** *Jika pengguna melakukan instalasi SageMath menggunakan SnapStore, maka secara default ketika ikon SageMath diklik jendela Jupyter notebook akan terbuka secara otomatis pada default browser yang kita pakai.*

**Latihan 0.2.3** Jika Anda menggunakan Linux Ubuntu sebagai sistem operasi yang digunakan sehari-hari, lakukan instalasi SageMath pada Linux Ubuntu.

## 0.3 Instalasi SageMath pada macOS

SageMath dapat di-*install* secara *native* pada macOS mengingat sistem operasi ini berbasis Unix sebagaimana Linux. Untuk melakukan instalasi, kita dapat menggunakan *binary package* yang tersedia di <https://www.sagemath.org/download-mac.html> sebagaimana pada Gambar 0.20. Ada dua metode instalasi SageMath pada macOS yang dapat digunakan:



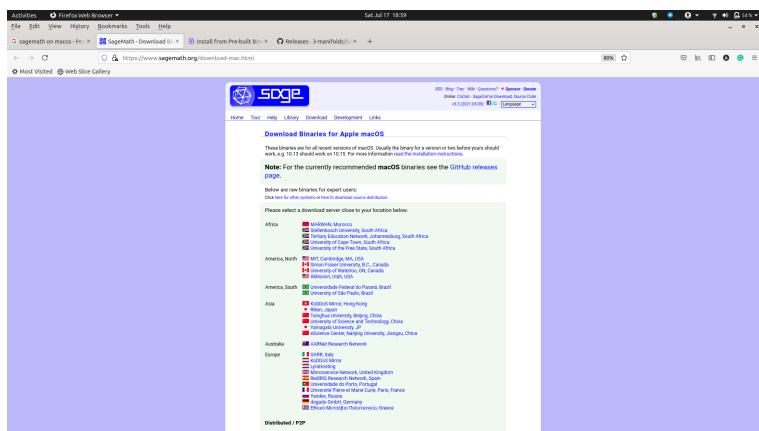
Gambar 0.19: Cara menjalankan SageMath dengan memilih ikon aplikasi yang terdapat pada Linux Ubuntu.

1. Pengguna melakukan pengunduhan *binary tarball* (file dalam ekstensi `tar.bz2`) sebagaimana instalasi pada Linux Ubuntu.
2. Pengguna menggunakan *file binary* yang terkompresi dalam ekstensi `.dmg`.

Instalasi menggunakan *binary tarball* serupa dengan instalasi SageMath menggunakan *binary package* sebagaimana dijelaskan pada Subtopik 0.2.1 untuk sistem operasi Linux. Instalasi menggunakan *file .dmg* dilakukan dengan langkah-langkah berikut:

1. Setelah mengunduh *file .dmg* kita dapat melakukan *mount* untuk *file* tersebut (dapat dilakukan dengan klik ganda).
2. Lakukan *drag* terhadap *folder* SageMath yang muncul ke direktori `/Applications/`.
3. Lakukan pengujian pada aplikasi SageMath sebagaimana dijelaskan pada Subtopik 0.2 dalam instalasi untuk Linux Ubuntu.

**Latihan 0.3.1** Jika Anda menggunakan macOS sebagai sistem operasi yang digunakan sehari-hari, lakukan instalasi SageMath pada macOS.



Gambar 0.20: Informasi terkait lokasi *server* yang dapat digunakan untuk mengunduh *binary package* SageMath untuk macOS. Gambar diambil dari <https://www.sagemath.org/download-mac.html> pada Juli 2021.

# Topik 1

## Dasar-dasar Python pada SageMath

Antarmuka dari SageMath dapat diakses menggunakan *Command Line Interface* (CLI) yang dinamakan dengan SageMath *console* (*interpreter* atau terminal) maupun Jupyter Notebook *Interface* yang dinamakan dengan SageMath notebook. Pada topik ini akan dibahas mengenai dasar-dasar Python pada SageMath. Pada dasarnya semua fitur pada Python 3 dapat digunakan pada SageMath, namun tidak sebaliknya. Pembaca yang ingin mengetahui fitur-fitur Python 3 secara lebih rinci dapat membaca tutorial Python secara lengkap, misalnya pada [7]. Penjelasan dasar mengenai fitur-fitur komputasi simbolik untuk SageMath dapat dilihat secara lebih rinci di 8.

### 1.1 Perkenalan SageMath Notebook

Notebook pada dasarnya mengembangkan *console* menjadi aplikasi berbasis *web* yang cocok untuk menangkap seluruh proses komputasi dan penelitian, seperti pada proses pengembangan, dokumentasi, eksekusi kode, dan cara penyampaian hasilnya. Dokumen yang dihasilkan secara internal memiliki format JSON yang tersimpan dengan ekstensi `.ipynb`. Dokumen ini dapat tersedia melalui URL publik yang dapat dibagikan melalui Jupyter Notebook Viewer (*nbviewer*). Cara mengakses SageMath notebook adalah dengan menjalankan dulu *notebook server*. Pada sistem operasi Linux hal ini dapat dilakukan melalui perintah `jupyter notebook`<sup>1</sup>, sedangkan pada sistem operasi Windows hal ini dapat dilakukan dengan membuka aplikasi SageMath 9.x Notebook. URL *default* dari Jupyter notebook ketika dijalankan adalah `http://localhost:8888/tree`.

#### 1.1.1 Pembuatan *file* Notebook Baru

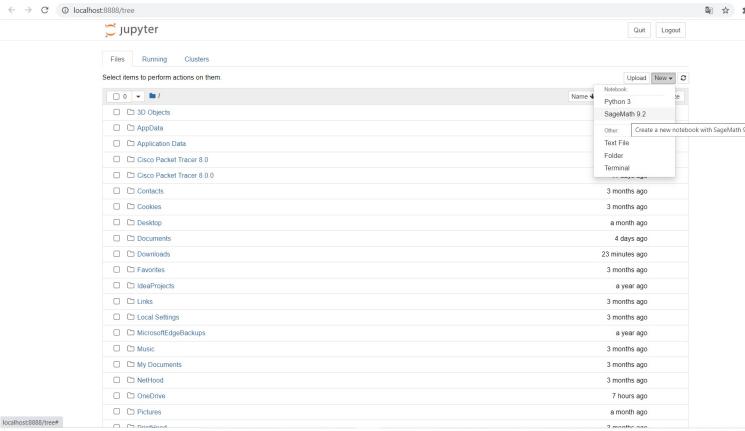
Pembuatan *file* notebook baru dapat diakses melalui *dashboard* yang ditunjukkan pada Gambar 1.1 ataupun menggunakan opsi menu *New Notebook* pada notebook aktif yang ditunjukkan pada Gambar 1.2. *File .ipynb* atau notebook yang baru ini secara *default* akan disimpan pada direktori yang sama dengan notebook yang aktif (biasanya pada direktori *Users/namauser*).

#### 1.1.2 Antarmuka Notebook

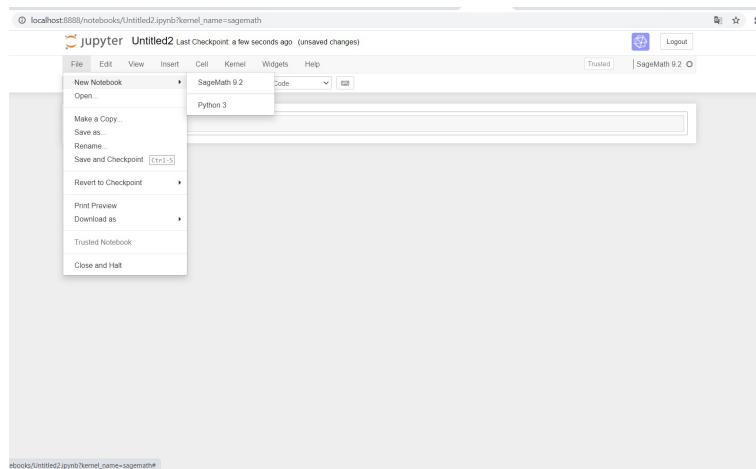
Terdapat beberapa fitur dan bagian-bagian pada sebuah notebook, yaitu nama notebook, menu bar, toolbar, dan *code cell* seperti yang ditunjukkan pada Gambar 1.3. Nama notebook juga merupakan representasi nama *file .ipynb* yang disimpan. Dengan mengeklik nama ini,

---

<sup>1</sup>Lihat Subtopik 0.2 untuk penjelasan detail.



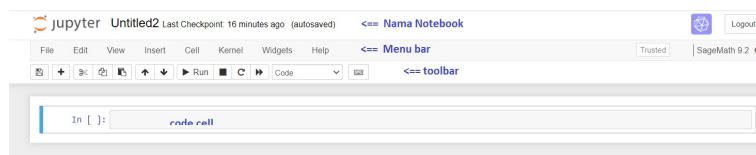
Gambar 1.1: Cara membuat *file* baru pada *dashboard* Jupyter notebook.



Gambar 1.2: Cara membuat *file* baru pada Jupyter notebook aktif.

pengguna bisa mengubah nama *file* menjadi nama lain yang diinginkannya. Pengubahan nama notebook otomatis mengubah nama *file* yang disimpan (yang bersesuaian dengan notebook tersebut).

Menu bar menampilkan berbagai opsi yang dapat digunakan untuk melakukan komputasi maupun dokumentasi notebook, toolbar menyediakan fitur untuk menampilkan operasi-operasi yang paling sering digunakan atau diakses di notebook, sedangkan *code cell* merupakan tempat untuk menulis kode program atau perintah pada SageMath. Sebuah *code cell* dapat berisi lebih dari satu perintah. Untuk mengeksekusi sebuah perintah pada suatu *code cell* kita dapat menekan kombinasi Ctrl+Enter ataupun Shift+Enter, atau mengeklik tombol *Play* pada toolbar. Kita juga dapat menjalankan perintah yang ada pada suatu *code cell* dengan memilih menu *Cell* pada menu bar, lalu kemudian mengeklik perintah *Run Cells*.



Gambar 1.3: Bagian-bagian notebook pada Jupyter notebook yang digunakan pada SageMath.

## 1.2 Perkenalan sintaks dasar Python

SageMath merupakan sebuah *Computer Algebra System* (CAS) atau Sistem Aljabar Komputasional yang menggunakan Python pada antarmuka penggunanya. Oleh karena itu pemahaman mengenai beberapa sintaks dasar pada Python sangat membantu dalam pemakaian SageMath. Secara khusus SageMath (sejak SageMath versi 8.9 pada tahun 2019) menggunakan Python 3. Pembaca yang baru pertama kali memakai Python dapat membaca tutorial singkat berikut.

### 1.2.1 Penugasan, Perbandingan, dan Operasi Dasar Aritmetika

Penugasan atau pemberian nilai (*assignment*) pada SageMath dijalankan dengan perintah `=`. Sebagaimana bahasa pemrograman pada umumnya, ada lima operator perbandingan aritmetika yang dapat kita gunakan, yaitu operator sama dengan, lebih kecil, lebih besar, lebih kecil sama dengan, dan lebih besar sama dengan. Kelima operator tersebut secara berurutan direpresentasikan dengan simbol `==`, `<`, `>`, `<=`, dan `>=`. Misalkan kita mengetikkan skrip berikut pada sebuah *code cell* di SageMath:

```
a=3  
b=5  
print('a==b', a==b)  
print('a<b', a<b)  
print('a<b', a<b)  
a=b  
print('a<=b', a<=b)  
print('a>=b', a>=b)
```

Sintaks `a == b` merupakan perbandingan nilai numerik antara variabel `a` dan `b`, sedangkan sintaks `a=b` merupakan penugasan nilai pada variabel `b` ke dalam variabel `a`. Kita dapat memperoleh hasil kalkulasi dengan menekan kombinasi `Ctrl+Enter` atau `Shift+Enter` pada *keyboard* sehingga diperoleh hasil berikut:

```
a==b False  
a<b True  
a<b True  
a<=b True  
a>=b True
```

Python 3 merupakan bahasa pemrograman yang menggunakan *dynamic typing*. Ini berarti pengecekan tipe data dilakukan oleh Python 3 dan pengguna tidak perlu mendefinisikan secara eksplisit tipe data dari suatu variabel. Oleh karena itu tipe data juga tidak perlu didefinisikan secara eksplisit pada SageMath oleh pengguna, meskipun hal ini juga tetap dapat dilakukan (dan terkadang diperlukan). Tipe data akan disesuaikan dengan isi dari variabel itu sendiri. Titik menjadi pembeda antara bilangan bulat (*integer*) dan bilangan real (*float*). Selain itu, **tipe data *integer* maupun *float* di Python dan SageMath memiliki makna yang berbeda**. Misalkan kita menuliskan skrip berikut:

```
print(type(5))  
print(type(5.0))  
print(type('d'))  
print(type(True))
```

Dengan menekan kombinasi `Ctrl+Enter` pada *keyboard* kita memperoleh hasil berikut:

```
<class 'sage.rings.integer.Integer'>  
<class 'sage.rings.real_mpfr.RealLiteral'>  
<class 'str'>  
<class 'bool'>
```

---

Secara *default* SageMath akan memperlakukan sebuah bilangan bulat sebagai elemen dari ring  $\mathbb{Z}$ . Hal ini akan dijelaskan pada Topik 2. Pada SageMath kita dapat mengetikkan tanda tanya ? setelah sintaks tertentu. Tanda tanya tersebut diartikan sebagai perintah untuk menampilkan bantuan (*help*) terkait deskripsi mengenai sintaks tertentu yang kita tuliskan. Misalnya kita menuliskan perintah berikut:

```
bool?
```

Dengan menekan kombinasi **Ctrl+Enter** pada *keyboard* kita memperoleh hasil berikut:

```
Init signature: bool(self, /, *args, **kwargs)
```

```
Docstring:
```

```
bool(x) -> bool
```

Returns True when the argument x is true, False otherwise.  
The builtins True and False are the only two instances  
of the class bool.

The class bool is a subclass of the class int, and cannot  
be subclassed.

Init docstring: Initialize self. See help(type(self)) for  
accurate signature.

```
File:
```

```
Type: type
```

```
Subclasses:
```

Notasi atau sintaks dari operasi aritmetika dalam SageMath ditunjukkan pada Tabel 1.1. Hasil pemangkatan dan perkalian akan bertipe *integer* (bilangan bulat) ketika kedua *operand* bertipe *integer*. Hasil pemangkatan dan perkalian akan bertipe *float* (bilangan real) ketika setidaknya ada salah satu *operand* yang bertipe *float*.

Tabel 1.1: Notasi atau sintaks operasi aritmetika pada SageMath.

No.	Sintaks	Arti
1	+ dan –	Penjumlahan dan pengurangan.
2	/	Pembagian dalam <i>float</i> .
3	//	Pembagian dalam <i>integer</i> (operasi div).
4	%	Modulo atau sisa hasil bagi.
5	** dan ^	Pemangkatan.

Berbeda dengan Python, SageMath juga mendukung operator  $\wedge$  untuk operasi pemangkatan. Secara umum hampir semua operasi aritmetika yang dapat dijalankan dalam Python juga dapat dijalankan pada SageMath (meskipun dengan sedikit penyesuaian), namun tidak sebaliknya.

SageMath juga mengakomodasi proses *input* dan *output* nilai atau isi variabel. Penerimaan nilai masukan dari pengguna dilakukan dengan sintaks `input()`. Perintah ini akan menunggu pengguna memberikan masukan yang diakhiri dengan penekanan tombol *Enter*. Salah satu hal penting yang perlu diketahui adalah **semua data yang dimasukkan pengguna akan dibaca sebagai string**. Oleh karena itu sering kali kita perlu mengubah masukan pengguna ke dalam tipe yang sesuai dengan algoritma yang kita terapkan. Misalkan kita menuliskan skrip berikut:

```
a=input("masukkan sebuah bilangan= ")  
print(type(a))
```

Apabila kita memasukkan nilai 5 pada kotak dialog *input* yang muncul, maka SageMath akan memberikan keluaran berikut:

```
masukkan sebuah bilangan= 5  
<class 'str'>
```

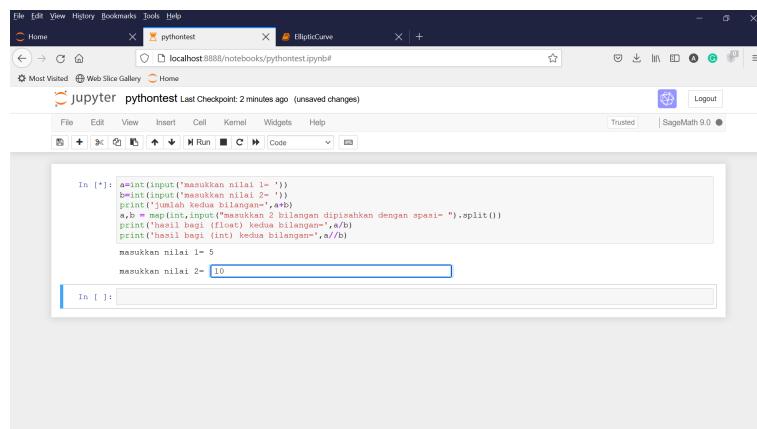
Karena nilai masukan pengguna disimpan dalam bentuk string, sedangkan bisa jadi nilai tersebut diproses sebagai angka (atau yang lain), maka terkadang kita perlu mengubah string tersebut ke dalam tipe data yang lain menggunakan *type casting*. Misalkan kita ingin membuat program sederhana yang menerima dua bilangan bulat  $a$  dan  $b$  serta kita ingin menghitung nilai dari  $a + b$ ,  $a/b$ , dan  $a//b$ , maka kita dapat menggunakan skrip berikut:

```
a=int(input('masukkan nilai 1= '))
b=int(input('masukkan nilai 2= '))
print('jumlah kedua bilangan=',a+b)
a,b = map(int,input("masukkan 2 bilangan dipisahkan dengan spasi= ").split())
print('hasil bagi (float) kedua bilangan=',a/b)
print('hasil bagi (int) kedua bilangan=',a//b)
```

*Method .split* digunakan untuk menerima masukan lebih dari satu yang secara *default* pemisahnya adalah spasi. Untuk memperoleh hasilnya, kita dapat menekan kombinasi tombol *Ctrl+Enter* pada *keyboard* sehingga dapat diperoleh interaksi berikut:

```
masukkan nilai 1= 5
masukkan nilai 2= 10
jumlah kedua bilangan= 15
masukkan 2 bilangan dipisahkan dengan spasi= 3 4
hasil bagi (float) kedua bilangan= 0.75
hasil bagi (int) kedua bilangan= 0
```

Pada interaksi ini bilangan pertama yang kita masukan adalah 5 dan bilangan kedua yang kita masukan adalah 10. Kemudian dua bilangan yang kita masukan dan dipisahkan dengan spasi adalah 3 dan 4. Tampilan ketika Jupyter notebook menerima masukan dari pengguna dapat dilihat pada Gambar 1.4.



Gambar 1.4: Ilustrasi interaksi *input* dari pengguna pada Jupyter notebook.

**Latihan 1.2.1** Buatlah sebuah program dengan ketentuan sebagai berikut. Program menerima masukan berupa empat bilangan asli (bilangan bulat positif)  $a$ ,  $b$ ,  $c$ , dan  $d$  yang dipisahkan dengan spasi. Keluaran dari program adalah *True* apabila hasil dari  $\frac{a}{b}$  sama dengan  $\frac{c}{d}$  atau *False* jika hasilnya berbeda. Untuk lebih jelasnya, perhatikan contoh pada Tabel 1.2.

Tabel 1.2: Contoh hasil dari Latihan 1.2.1.

<i>Input</i>	<i>Output</i>
2 1 4 2	True
2 1 4 3	False

### 1.2.2 Pengkondisian

Program komputer terdiri dari banyak aksi yang dikerjakan secara berurutan atau runut. Setiap aksi berawal dari kondisi tertentu atau *trigger* sebagai pembangkitnya. Kondisi diartikan sebagai ekspresi yang jika dievaluasi bernilai True atau False, sedangkan aksi adalah instruksi yang dijalankan berdasarkan nilai kondisi yang dihasilkan.

Pengkondisian atau percabangan memiliki dua skenario utama. Skenario pertama adalah ketika kasus yang ditinjau hanya memiliki sebuah kondisi seperti pada contoh berikut:

```
if 1==1:  
    print('aksi 1')  
else:  
    print('aksi 2')
```

Jika kondisi bernilai benar maka aksi 1 (yaitu memberikan keluaran 'aksi 1') akan dijalankan, akan tetapi jika kondisi salah maka aksi 2 (yaitu memberikan keluaran 'aksi 2') akan dijalankan. Untuk memperoleh hasilnya, kita dapat menekan tombol **Ctrl+Enter** pada *keyboard* sehingga diperoleh hasil berikut:

aksi 1

Perhatikan bahwa karena ekspresi `1 == 1` benar, maka string `aksi 1` dikeluarkan oleh SageMath. Skenario kedua dijalankan ketika terdapat lebih dari satu kondisi alternatif. Ilustrasi pada Algoritma 1 menunjukkan percabangan dengan dua kondisi yang terdefinisi dan sebuah kondisi untuk menampung sisanya (yang tidak memenuhi dua kondisi terdefinisi sebelumnya).

**Algoritma 1** Pengkondisian dengan lebih dari satu kondisi alternatif.

- ```
1. if kondisi1 then
2.   aksi1                                     // kondisi1 benar
3. else if kondisi2 then
4.   aksi2                                     // kondisi1 salah, kondisi2 benar
5. else
6.   aksi3                                     // kondisi1 salah, kondisi2 salah
7. end if
```

Kondisi pada Algoritma 1 dapat dituliskan pada Python dengan format berikut:

```
if kondisi1:  
    aksil # kondisi1 benar  
elif:  
    aksi2 # kondisi1 salah, kondisi2 benar  
else:  
    aksi3 # kondisi1 salah, kondisi2 salah
```

Sebagai contoh kongkret misalkan kita memiliki skrip berikut:

```
if 1==1 and 5>10:  
    print('aksi 1')  
elif 5>=5.01 or 4<4  
    print('aksi 2')  
else:  
    print('aksi 3')
```

Untuk memperoleh hasilnya, kita dapat menekan tombol **Ctrl+Enter** pada *keyboard* sehingga diperoleh:

aks 3

Perhatikan bahwa ekspresi `1 == 1` and `5>10` maupun `5>=5.01` or `4<4` keduanya bernilai salah.

---

**Latihan 1.2.2** Buatlah program kalkulator sederhana yang menerima masukan dua buah bilangan real dan sebuah karakter yang melambangkan operator yang digunakan. Operator yang dikenali adalah +, -, \*, dan /. Format tampilan yang diharapkan adalah sebagai berikut: operand operator operand = hasil. Untuk lebih jelasnya, perhatikan contoh di Tabel 1.3.

Tabel 1.3: Contoh hasil dari Latihan 1.2.2.

| <i>Input</i> | <i>Output</i>      |
|--------------|--------------------|
| 10 20 +      | 10.0 + 20.0 = 30.0 |

### 1.2.3 Perulangan

Sebagaimana ada pada Python 3, terdapat dua jenis perulangan yang dikenali oleh SageMath, yaitu perintah `while` dan `for`. Fungsi perintah `while` hampir mirip dengan perintah pengkondisian `if`, yaitu aksi akan dijalankan ketika kondisi bernilai `True`. Perbedaannya adalah perintah `if` paling banyak hanya akan menjalankan aksi sebanyak satu kali, sedangkan perintah `while` akan menjalankan aksi berulang-ulang selama kondisi bernilai `True`. Misalkan kita mengetikkan skrip berikut pada SageMath:

```
i=1
while (i<10):
    print(i)
    i+=1
print(i,'tidak dicetak didalam loop')
```

Program selalu memeriksa kondisi terlebih dahulu sebelum menjalankan aksi dari `while`, begitu kondisi bernilai `False`, maka program akan keluar dari perintah `while` dan melanjutkan ke perintah berikutnya. Untuk memperoleh hasil dari skrip sebelumnya, kita dapat menekan tombol `Ctrl+Enter` pada *keyboard* sehingga diperoleh:

```
1
2
3
4
5
6
7
8
9
10 tidak dicetak didalam loop
```

Pada skrip sebelumnya perintah `print(i)` dijalankan hanya ketika nilai  $i < 10$ . Perintah `for` digunakan untuk perulangan sekuensial yang diketahui nilai batas bawah dan batas atasnya (dinyatakan dalam `range`) atau kumpulan dari objek yang akan diiterasi (dinyatakan dalam `list`). Misalkan kita memiliki skrip berikut:

```
for i in range (1,10):
    print(i)
print(i,'tidak dicetak didalam loop')
list1 = ['sate','bakso','geprek']
for i in list1:
    print(i)
print(i,'adalah nama-nama makanan')
```

Untuk memperoleh hasil dari skrip sebelumnya, kita dapat menekan tombol `Ctrl+Enter` pada *keyboard* sehingga diperoleh:

---

```

1
2
3
4
5
6
7
8
9
9 tidak dicetak didalam loop
sate
bakso
geprek
geprek adalah nama-nama makanan

```

Meskipun sama-sama digunakan untuk perulangan, perintah `while` dan `for` memiliki bentuk yang berbeda. Bentuk `while` mengulang berdasarkan kondisi, sementara perintah `for` mengulang berdasarkan *item* di dalam sebuah `range` atau *list* (kumpulan objek).

**Catatan 1.2.3** *Kita perlu berhati-hati dalam menggunakan `range(a, b)` pada Python untuk nilai bilangan bulat  $a$  dan  $b$ . Pada Python nilai `range(a, b)` mencakup semua bilangan bulat antara  $a$  dan  $b-1$  secara inklusif, termasuk  $a$  dan  $b-1$  sendiri. Perhatikan bahwa `range(a, b)` tidak mencakup nilai  $b$ .*

**Latihan 1.2.4** Buatlah program sederhana yang menerima masukan bilangan ganjil dan akan berhenti menerima masukan jika pengguna memberikan bilangan genap. Kemudian tampilkan penjumlahan semua bilangan ganjil yang telah dimasukan. Untuk lebih jelasnya perhatikan contoh pada Tabel 1.4.

Tabel 1.4: Contoh hasil dari Latihan 1.2.4.

| <i>Input</i> | <i>Output</i> |
|--------------|---------------|
| 1            | 20            |
| 3            |               |
| 5            |               |
| 11           |               |
| 2            |               |

## 1.2.4 *List* dan *Tuple*

Tipe data terstruktur diartikan sebagai tipe data yang dapat menyimpan banyak nilai dan memiliki struktur tertentu. Tujuan adanya tipe data terstruktur ini adalah untuk mempermudah pengorganisasian, penyimpanan, dan manipulasi sejumlah data tertentu. Terdapat empat tipe data terstruktur *built-in* yang sifatnya primitif di SageMath (yang berbasis pada Python), yaitu *list*, *tuple*, *set*, dan *dictionary*. Pada tutorial ini hanya akan dibahas dua tipe data terstruktur yang sering digunakan, yaitu *list* dan *tuple*.

### *List*

*List* adalah tipe data terstruktur yang dapat menyimpan elemen atau objek dengan tipe data yang berbeda-beda. *List* memiliki indeks untuk menunjukkan posisi dengan nilai terurut secara sekuensial. *List* menyerupai *array* pada bahasa pemrograman seperti C dan C++, akan tetapi

---

*array* hanya bisa menyimpan elemen dengan tipe data yang sama. Elemen-elemen *list* juga boleh redundan (dua indeks berbeda menyimpan nilai yang sama). Selain itu, antara *list* dengan elemen/objeknya bersifat *mutable*, yang memiliki dua ciri utama yaitu *list* dapat diubah (ditambah maupun dikurangi) dan elemen dari *list* bisa dimodifikasi atau diubah. Pendefinisian dan fungsi pada *list* dapat ditunjukkan pada Tabel 1.5.

Tabel 1.5: Perintah dasar untuk *list* pada SageMath.

| No. | Sintaks                                                                                                                       | Keterangan                                                                                                             |
|-----|-------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| 1   | $\langle nm\_list \rangle = [\langle isi \rangle]$<br>atau<br>$\langle nm\_list \rangle = \text{list}([\langle isi \rangle])$ | membuat <i>list</i>                                                                                                    |
| 2   | $\langle nm\_list \rangle[\langle idx \rangle]$                                                                               | mengakses isi atau elemen dari <i>list</i>                                                                             |
| 3   | $\langle nm\_list \rangle.\text{index}(\langle isi \rangle)$                                                                  | mencari nilai indeks jika diketahui isi atau nilainya                                                                  |
| 4   | $\langle isi \rangle \text{ in } \langle nm\_list \rangle$                                                                    | cek keanggotaan <i>list</i> ( <i>output boolean</i> )                                                                  |
| 5   | <b>for</b> $\langle var \rangle$ <b>in</b> $\langle nm\_list \rangle$                                                         | iterasi isi atau elemen <i>list</i>                                                                                    |
| 6   | $\langle nm\_list \rangle[\langle start \rangle : \langle end \rangle]$                                                       | memotong isi <i>list</i> mulai indeks $\langle start \rangle$ hingga $\langle end \rangle - 1$                         |
| 7   | $\langle nm\_list \rangle.\text{append}(\langle isi \rangle)$                                                                 | menambah sebuah elemen di akhir <i>list</i>                                                                            |
| 8   | $\langle nm\_list \rangle.\text{extend}([\langle isi \rangle])$                                                               | menambah beberapa elemen <i>iterable</i> di akhir list                                                                 |
| 9   | $\langle nm\_list \rangle.\text{insert}(\langle idx \rangle, \langle isi \rangle)$                                            | menambah sebuah elemen di posisi tertentu                                                                              |
| 10  | $\langle nm\_list \rangle[\langle idx \rangle] = \langle isibaru \rangle$                                                     | mengubah isi elemen <i>list</i>                                                                                        |
| 11  | $\langle nm\_list \rangle.\text{remove}(\langle isi \rangle)$                                                                 | menghapus elemen yang bernilai $\langle isi \rangle$                                                                   |
| 12  | $\langle nm\_list \rangle.\text{pop}(\langle idx \rangle)$                                                                    | menghapus elemen yang berada pada $\langle idx \rangle$ <i>default</i> ,<br>menghapus elemen terakhir dari <i>list</i> |
| 13  | $\langle nm\_list \rangle.\text{clear}()$                                                                                     | menghapus seluruh isi <i>list</i>                                                                                      |

Berikut contoh penggunaan *list* pada SageMath:

```
list_a = ['a',1,2,3,'b']
print('list_a[0]=' ,list_a[0])
print('list_a.index(2)' ,list_a.index(2))
print('b' in list_a)
for i in list_a:
    print(i)
print('list_a[1:3]' ,list_a[1:3])
list_a.append(19)
list_a.extend([20,21,22])
print(list_a)
list_a.insert(1,0)
print('list_a.insert(1,0)' ,list_a)
list_a[list_a.index('b')]=18
print("list_a[list_a.index('b')]=18 menjadi" ,list_a)
list_a.remove('a')
print("list_a.remove('a')" ,list_a)
print('list_a.pop()' ,list_a.pop())
print('jadi isi list_a sekarang' ,list_a)
list_a.clear()
print('list_a.clear()' ,list_a)
```

Untuk memperoleh hasilnya, kita dapat menekan tombol Ctrl+Enter pada *keyboard* sehingga diperoleh:

---

```

list_a[0]= a
list_a.index(2)= 2
True
a
1
2
3
b
list_a[1:3]= [1, 2]
['a', 1, 2, 3, 'b', 19, 20, 21, 22]
list_a.insert(1,0)= ['a', 0, 1, 2, 3, 'b', 19, 20, 21, 22]
list_a[list_a.index('b')]=18 menjadi
['a', 0, 1, 2, 3, 18, 19, 20, 21, 22]
list_a.remove('a')= [0, 1, 2, 3, 18, 19, 20, 21, 22]
list_a.pop()= 22
jadi isi list_a sekarang [0, 1, 2, 3, 18, 19, 20, 21]
list_a.clear()= []

```

**Latihan 1.2.5** Buatlah sebuah program yang memenerima dua buah masukan himpunan  $A$  dan  $B$ . Keluarkan banyaknya semua anggota pada  $A \cap B$  (irisan dari  $A$  dan  $B$ , atau elemen-elemen yang muncul pada himpunan  $A$  maupun  $B$ ). Untuk lebih jelasnya perhatikan contoh pada Tabel 1.6.

Tabel 1.6: Contoh hasil dari Latihan 1.2.5.

| Input       | Output |
|-------------|--------|
| 2 6 8 10    | 2 6    |
| 1 2 3 4 5 6 |        |

### ***Tuple***

*Tuple* (atau tupel) adalah tipe data terstruktur yang dapat menyimpan elemen atau objek dengan tipe data yang berbeda-beda. *Tuple* juga memiliki indeks untuk menunjukkan posisi dengan nilai terurut secara sekuensial. *Tuple* serupa dengan *list*, namun *tuple* bersifat *immutable*. Elemen/objek dari suatu *tuple* tidak dapat diubah (ditambah maupun dikurangi) kecuali jika elemen *tuple* bersifat *mutable* (misalnya bila elemen *tuple* adalah sebuah *list*). Elemen *tuple* boleh redundan (dua indeks berbeda dapat menyimpan nilai yang sama). Pendefinisian dan fungsi pada *tuple* ditunjukkan pada Tabel 1.7.

Tabel 1.7: Perintah dasar untuk *tuple* pada SageMath.

| No. | Sintaks                                                                                                                        | Keterangan                                                                                      |
|-----|--------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| 1   | $\langle nm\_tuple \rangle = (\langle isi \rangle)$<br>atau<br>$\langle nm\_tuple \rangle = \text{tuple}(\langle isi \rangle)$ | membuat <i>tuple</i>                                                                            |
| 2   | $\langle nm\_tuple \rangle[\langle idx \rangle]$                                                                               | mengakses isi atau elemen dari <i>tuple</i>                                                     |
| 3   | $\langle nm\_tuple \rangle.\text{index}(\langle isi \rangle)$                                                                  | mencari nilai indeks jika diketahui isi atau nilainya                                           |
| 4   | $\langle isi \rangle \text{ in } \langle nm\_tuple \rangle$                                                                    | cek keanggotaan <i>tuple</i> ( <i>output boolean</i> )                                          |
| 5   | <b>for</b> $\langle var \rangle$ <b>in</b> $\langle nm\_tuple \rangle$                                                         | iterasi isi atau elemen <i>tuple</i>                                                            |
| 6   | $\langle nm\_tuple \rangle[\langle start \rangle : \langle end \rangle]$                                                       | memotong isi <i>tuple</i> mulai indeks $\langle start \rangle$ hingga $\langle end \rangle - 1$ |
| 7   | $\langle nm\_tuple \rangle.\text{count}(\langle isi \rangle)$                                                                  | menghitung banyaknya elemen pada <i>tuple</i>                                                   |

---

**Latihan 1.2.6** Buatlah sebuah program yang menerima masukan sebuah *tuple* yang berisi bilangan bulat positif dan sebuah bilangan bulat  $x$ . Hitunglah ada berapa banyak elemen *tuple* yang habis dibagi  $x$ . Untuk lebih jelasnya perhatikan contoh pada Tabel 1.8.

Tabel 1.8: Contoh hasil dari Latihan 1.2.6.

| <i>Input</i>             | <i>Output</i> |
|--------------------------|---------------|
| 1 2 3 4 5 6 7 10 15<br>2 | 4             |

## 1.2.5 Fungsi

Fungsi berguna untuk menyederhanakan atau membagi-bagi permasalahan menjadi bagian lebih kecil (atau modular) yang lebih sederhana. Fungsi bertujuan untuk memudahkan perbaikan kode atau meningkatkan skalabilitas jika kode akan dikembangkan. Kode program yang besar akan dikelompokkan berdasarkan tugas atau tujuan dari kode tersebut. Selain itu, fungsi juga berguna untuk mengurangi penulisan kode yang sama berulang kali. Fungsi pada SageMath terdiri dari dua jenis yaitu fungsi yang dibuat oleh *programmer (user-defined function)* dan fungsi yang sudah tersedia pada bahasa pemrograman (dalam hal ini SageMath) sehingga sudah siap digunakan *programmer (built-in function)*. Pada tutorial ini akan dibahas lebih detail mengenai *user-defined function*. Perhatikan contoh skrip berikut:

```
def tambah(a,b):  
    """  
        fungsi ini merupakan penjumlahan  
        antara bilangan pertama dan kedua  
    """  
    return(a+b)  
help(tambah)  
print('tambah(5,3)=',tambah(5,3))
```

Untuk memperoleh hasilnya, kita dapat menekan tombol Ctrl+Enter pada *keyboard* sehingga diperoleh:

```
Help on function tambah in module __main__:  
  
tambah(a, b)  
    fungsi ini merupakan penjumlahan  
    antara bilangan pertama dan kedua  
  
tambah(5, 3) = 8
```

*Docstring* yang diapit dengan tanda petik adalah penjelasan atau abstraksi kegunaan dari fungsi yang dibuat *programmer*. Tujuan adanya *docstring* adalah memberikan informasi singkat mengenai cara kerja fungsi sehingga memudahkan *programmer* lain yang akan memakai atau memodifikasi fungsi tersebut.

Penggunaan atau pemanggilan fungsi tidak lepas dari ruang lingkup variabel. Sebagaimana bahasa pemrograman terstruktur seperti C, C++, dan Pascal, pada Python kita juga mengenal istilah variabel lokal dan variabel global. Pendefinisian jenis-jenis variabel ini dilakukan untuk menghindari kesalahan pemanggilan variabel. Variabel global adalah variabel yang didefinisikan di program utama atau di luar fungsi (di luar *subprogram*) sehingga ruang lingkup variabel global lebih luas dan dapat dikenali oleh seluruh bagian program termasuk fungsi (*subprogram*) di dalamnya. Variabel lokal merupakan variabel yang didefinisikan di dalam fungsi (di dalam *subprogram*) sehingga ruang lingkup variabel ini hanya di dalam fungsi yang mendefinisikan-

---

nya saja dan tidak dapat dikenali/dipanggil/diakses di luar fungsi (di luar *subprogram*). Hal ini dapat diartikan bahwa variabel lokal tidak dapat dikenali program utama. Untuk memperjelas misalkan kita memiliki contoh skrip berikut:

```
t=5
def volume(a,b):
    """
        fungsi ini merupakan perhitungan volume jika
        tinggi didapatkan dari variabel global
    """
    hasil=t*a*b
    return(hasil)
t=8
help(volume)
print("volume=",volume(3,4))
```

Skrip sebelumnya memberikan keluaran berikut:

```
Help on function volume in module __main__:

volume(a, b)
    fungsi ini merupakan perhitungan volume jika
    tinggi didapatkan dari variabel global

volume= 96
```

Fungsi yang telah kita buat menghitung volume dari sebuah balok yang panjang dan lebar alasnya adalah  $a$  dan  $b$  serta memiliki tinggi  $t$ . Volume balok yang dimaksud memenuhi persamaan  $t \cdot a \cdot b$ . Dalam skrip yang kita buat, nilai  $t$  merupakan variabel global sedangkan nilai  $a$  dan  $b$  adalah variabel lokal yang hanya ditinjau di dalam fungsi `volume(a, b)`.

**Latihan 1.2.7** *Caesar cipher* adalah salah satu metode enkripsi klasik sederhana yang memanfaatkan pergeseran karakter. Urutan karakter pada pesan atau *plaintext* digeser sebanyak nilai kunci yang digunakan (ke kanan jika kunci tersebut bilangan bulat positif). Buatlah sebuah fungsi untuk meng-*encode* pesan dengan menggeser setiap karakter dari teks. Nilai dari pesan dan kunci merupakan masukan dari pengguna (asumsi: semua pesan ditulis menggunakan huruf kecil). Contoh masukan dan keluaran program ditunjukkan pada Tabel 1.9.

Tabel 1.9: Contoh hasil dari Latihan 1.2.7.

| <i>Input</i> | <i>Output</i> |
|--------------|---------------|
| saya         | ucac          |
| 2            |               |

## 1.2.6 Beberapa Operasi Simbolis Sederhana pada SageMath

Salah satu fitur yang membedakan SageMath dengan Python 3 biasa adalah kemampuan SageMath untuk menyelesaikan komputasi-komputasi matematis secara simbolis. SageMath—sebagaimana sistem aljabar komputasional yang lain seperti Maple, Mathematica, dan Magma—dapat dipakai untuk menyelesaikan komputasi-komputasi simbolis yang dituliskan dalam bentuk ekspresi matematis.

SageMath dapat digunakan untuk menyelesaikan persamaan maupun pertidaksamaan. Misalkan kita ingin mencari solusi dari ekspresi-ekspresi matematika berikut:

$$1. \quad 3x - 2 = 5$$

---

$$2. \ 3x - 2 > 5$$

$$3. \ 2x - 5 \leq 17$$

Kita dapat menuliskan skrip berikut pada SageMath:

```
print(solve(3*x-2 == 5, x))
print(solve(3*x-2 > 5, x))
print(solve(2*x-5 <= 17, x))
```

SageMath memberikan keluaran berikut:

```
[  
x == (7/3)  
]  
[ [x > (7/3)] ]  
[ [x <= 11] ]
```

Untuk mempermudah kita dalam membaca solusi matematis yang dikeluarkan oleh SageMath, kita dapat mengganti fungsi `print(...)` dengan fungsi `show(...)`. Misalkan kita mengetikkan perintah berikut:

```
show(solve(3*x-2 == 5, x))
show(solve(3*x-2 > 5, x))
show(solve(2*x-5 <= 17, x))
```

Dengan menekan kombinasi `Ctrl+Enter` SageMath akan menampilkan keluaran sebagaimana pada Gambar 1.5.

```
In [2]: show(solve(3*x-2 == 5, x))
show(solve(3*x-2 > 5, x))
show(solve(2*x-5 <= 17, x))

[x = (7/3)]
[[x > (7/3)]]
[[x <= 11]]
```

Gambar 1.5: Contoh penggunaan fungsi `show(...)` untuk menyelesaikan ekspresi matematika pada SageMath.

SageMath hampir selalu memberikan solusi ekspresi matematika dalam format *list* karena sebuah ekspresi matematika dapat memiliki lebih dari satu solusi. Misalkan kita ingin menyelesaikan persamaan-persamaan berikut menggunakan SageMath:

$$1. \ x^2 + x = 6$$

$$2. \ 2x^2 - x + 1 = 0$$

$$3. \ \exp(x) - x = 0$$

$$4. \ \cos(x) - \exp(x) = 0$$

Kita dapat menuliskan skrip berikut pada SageMath:

```
show(solve(x^2 + x == 6, x))
show(solve(2*x^2 - x + 1 == 0, x))
show(solve(exp(x) - x == 0, x))
show(solve(cos(x) - exp(x) == 0, x))
```

Keluaran dari skrip ini dapat dilihat pada Gambar 1.6. SageMath selalu berusaha untuk memberikan solusi ekspresi matematika dalam format *floating point*. Jika hal ini tidak dapat dilakukan, maka SageMath akan memberikan solusi dalam bentuk simbolis. Contohnya solusi

---

```
In [6]: show(solve(x^2 + x == 6, x))
show(solve(2*x^2 - x + 1 == 0, x))
show(solve(exp(x) - x == 0, x))
show(solve(cos(x) - exp(x) == 0, x))

[x = (-3), x = 2]
\left[ x = -\frac{1}{4}i\sqrt{7} + \frac{1}{4}, x = \frac{1}{4}i\sqrt{7} + \frac{1}{4} \right]
[x = e^x]
[cos(x) = e^x]
```

Gambar 1.6: Contoh penggunaan fungsi `show(...)` untuk menyelesaikan ekspresi matematika pada SageMath. Perhatikan bahwa solusi dari suatu ekspresi matematika dapat berupa sebuah persamaan.

dari  $\exp(x) - x = 0$  adalah  $\exp(x) = x$  dan solusi dari  $\cos(x) - \exp(x) = 0$  adalah  $\cos(x) = \exp(x)$ .

SageMath dapat digunakan untuk menyelesaikan persamaan maupun sistem persamaan yang meninjau lebih dari satu variabel. Dalam hal ini kita perlu mendefinisikan variabel yang digunakan terlebih dulu. Misalkan kita akan menyelesaikan sistem pertidaksamaan dua variabel berikut

$$\begin{aligned} x - y &\geq 2, \\ x + y &\leq 2. \end{aligned} \tag{1.1}$$

Kita dapat menyelesaikan sistem pertidaksamaan (1.1) menggunakan skrip berikut:

```
x, y = var('x y')
print(solve([ x-y >=2, x+y <=3], x,y))
```

Perintah `x, y = var('x y')` kita gunakan untuk menyatakan bahwa variabel yang ditinjau untuk ekspresi matematika kita adalah  $x$  dan  $y$ . SageMath memberikan keluaran berikut:

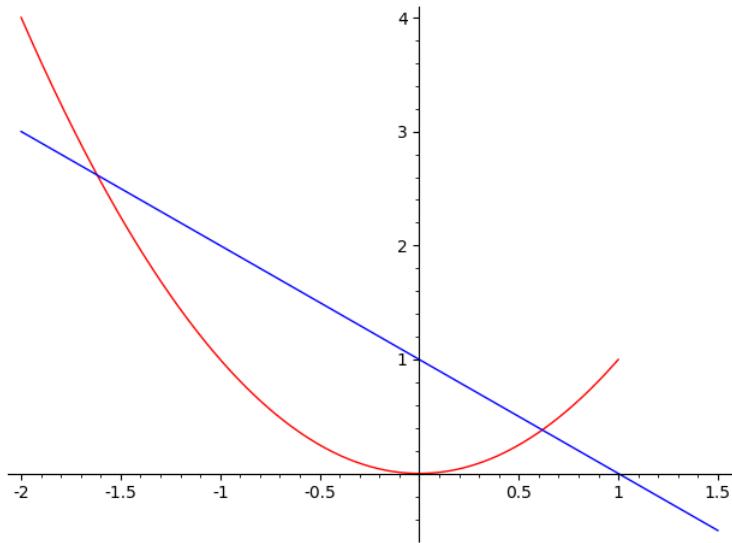
```
[]
[x == (5/2), y == (1/2)],
[x == -y + 3, y < (1/2)],
[x == y + 2, y < (1/2)],
[y + 2 < x, x < -y + 3, y < (1/2)]
]
```

Pada SageMath kita juga dapat membuat grafik dua dimensi maupun tiga dimensi yang mewakili satu atau lebih ekspresi matematika. Misalkan kita memiliki persamaan  $f(x) = x^2$  dan  $g(y) = -y + 1$ . Misalkan kita ingin menggambarkan grafik untuk  $f(x)$  untuk  $-2 \leq x \leq 1$  dan  $g(y)$  untuk  $-2 \leq y \leq 1.5$ . Kita dapat menggambarkan grafik untuk  $f(x)$  dan  $g(x)$  yang dimaksud dalam satu bidang menggunakan fungsi `plot(...)` sebagaimana dicontohkan pada skrip berikut:

```
x , y = var('x y')
fplot = plot(x^2, (x,-2,1), color='red')
gplot = plot(-y+1, (y,-2,1.5), color='blue')
show(fplot+gplot)
```

Pada skrip ini grafik fungsi  $f(x)$  diilustrasikan dengan warna merah (*red*) sedangkan grafik fungsi  $g(y)$  digambarkan dengan warna biru (*blue*). Hasil dari skrip ini adalah Gambar 1.7.

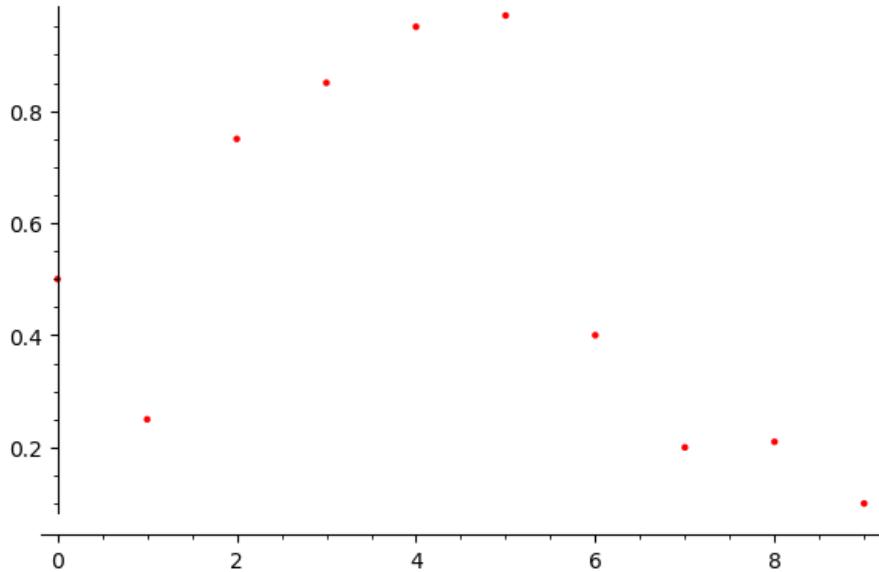
Kita juga dapat memakai SageMath untuk menggambarkan grafik yang mewakili data yang sifatnya diskret. Misalkan kita memiliki sebuah data yang disajikan dalam sebuah *list* `L = [0.50, 0.25, 0.75, 0.85, 0.95, 0.97, 0.40, 0.20, 0.21, 0.10]`. Kita dapat menggunakan fungsi `list_plot(...)` sebagaimana dicontohkan pada skrip berikut untuk menggambarkan representasi grafik dari data yang dijelaskan pada *list* `L`:



Gambar 1.7: Hasil penggambaran grafik dari fungsi  $f(x) = x^2$  untuk  $-2 \leq x \leq 1$  dan  $g(y) = -y + 1$  untuk  $-2 \leq y \leq 1.5$ .

```
L = [0.50, 0.25, 0.75, 0.85, 0.95, 0.97, 0.40, 0.20, 0.21, 0.10]
list_plot(L, color='red')
```

Hasil dari skrip ini adalah grafik pada Gambar 1.8.



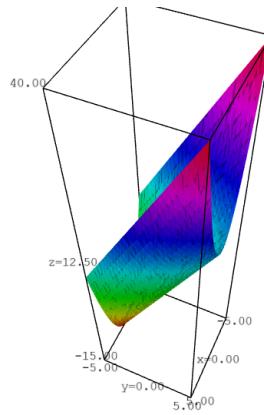
Gambar 1.8: Hasil penggambaran grafik dari data pada list

```
L = [0.50, 0.25, 0.75, 0.85, 0.95, 0.97, 0.40, 0.20, 0.21, 0.10].
```

SageMath juga dapat digunakan untuk membuat grafik dalam ruang tiga dimensi. Misalkan kita memiliki fungsi  $f(x, y) = x^2 + 3y$  dan kita ingin mengetahui representasi grafis dari fungsi  $f(x, y)$  yang dimaksud untuk rentang  $-5 \leq x, y \leq 5$ , maka kita dapat memakai fungsi `plot3d(...)` sebagaimana dijelaskan pada skrip berikut:

```
x , y = var('x y')
f = x^2 + 3*y
plot3d(f, (-5,5), (-5,5), adaptive = True)
```

Keluaran dari skrip ini adalah grafik pada Gambar 1.9.



Gambar 1.9: Hasil penggambaran grafik dari fungsi  $f(x, y) = x^2 + 3y$  untuk  $-5 \leq x, y \leq 5$ .

### Latihan 1.2.8

1. Gunakan SageMath untuk menyelesaikan sistem pertidaksamaan berikut:

$$\begin{aligned}x^2 + y^2 &\leq 100 \\3x + 4y &\geq 12\end{aligned}$$

2. Gunakan SageMath untuk menggambarkan grafik dari fungsi-fungsi berikut:

- (a)  $f(x) = x^2 - 100$  untuk  $-10 \leq x \leq 10$ .
- (b)  $f(x, y) = x^2 + y^2$  untuk  $-10 \leq x, y \leq 10$ .

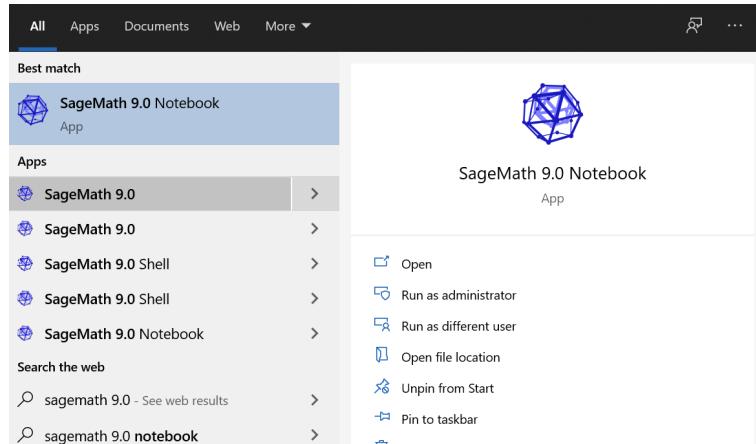
# Topik 2

## Dasar-dasar Aljabar pada SageMath

Kriptografi modern adalah topik yang sangat luas yang meliputi kajian pada ilmu komputer dan matematika. Beberapa kajian dalam matematika seperti teori bilangan, aljabar abstrak, dan aljabar linier dipakai dalam kriptografi. Pada bagian ini akan dijelaskan mengenai dasar-dasar penggunaan SageMath untuk komputasi yang terkait dengan aljabar, baik aljabar abstrak maupun aljabar linier. Penjelasan pada bagian ini dirangkum dari beberapa referensi, yaitu SAGEMath *reference documentation* untuk kajian *Basic Rings and Fields* dan *Linear Algebra* [1] serta referensi lain seperti yang disusun oleh R. A. Beezer [9] maupun T. W. Judson [10].

Sebelum memulai pembahasan materi, pengguna perlu mengetahui bahwa SageMath dapat dijalankan melalui *console* (terkadang juga disebut *shell*, *interpreter*, atau terminal) maupun Jupyter notebook. Ketika pengguna membuat skrip yang cukup panjang, penggunaan Jupyter notebook lebih disarankan dibandingkan dengan penggunaan *console/shell/interpreter/terminal*.

Pada sistem operasi Windows, penggunaan Jupyter notebook dapat dilakukan dengan mengetikkan SageMath x.y pada *start menu* untuk SageMath versi x.y sebagaimana diilustrasikan pada Gambar 2.1. Pengguna dapat memilih *SageMath x.y Notebook* untuk menjalankan Jupyter notebook.

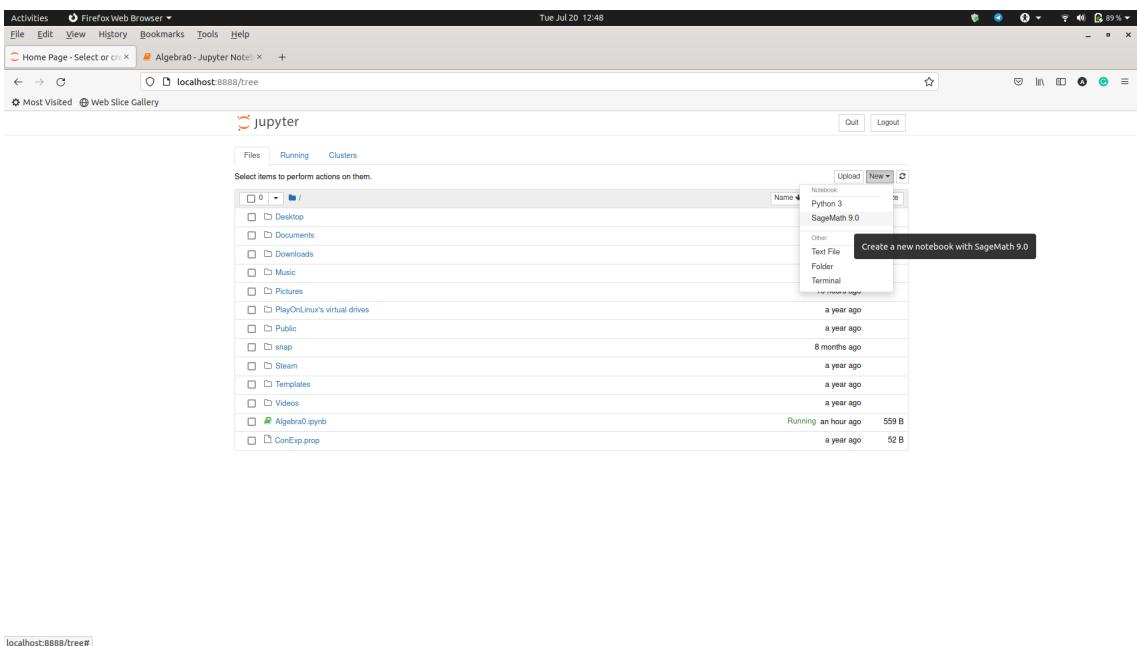


Gambar 2.1: Cara memulai penggunaan SageMath pada sistem operasi Windows.

Cara mengoperasikan SageMath pada *shell* atau *console* untuk sistem operasi Windows, Linux Ubuntu, maupun macOS serupa dan cukup sederhana. Pada penggunaan *shell*, pengguna perlu mengetikkan *sage* sebelum memulai interaksi dengan *interpreter* SageMath. Ilustrasi penggunaan *shell* dan *console* pada sistem operasi Windows dapat dilihat pada Gambar 2.2.

Pada Jupyter notebook, pengguna memulai SageMath dengan memilih *New*, lalu *Notebook*, lalu SageMath x.y (untuk SageMath versi x.y) sebagaimana pada Gambar 2.3. Pengguna dapat mengubah nama *file*, menyimpannya ke dalam *file* baru, maupun mengunduhnya dalam berbagai jenis format (seperti *.html*, *.tex*, *.ipynb*, *.pdf*, maupun *.py*).

Gambar 2.2: Penggunaan SageMath pada *shell* (gambar bagian kiri) dan *console* (gambar bagian kanan) pada sistem operasi Windows.



Gambar 2.3: Salah satu cara membuat *file* SageMath baru pada Jupyter notebook.

## 2.1 Operasi-operasi Dasar pada Bilangan Bulat

Beberapa sifat-sifat aljabar dari suatu objek atau struktur aljabar serupa dengan sifat-sifat yang dimiliki bilangan bulat. Pada bagian ini akan dijelaskan mengenai beberapa operasi pada SageMath untuk bilangan bulat (*integers*). Kita akan menotasikan himpunan bilangan bulat dengan  $\mathbb{Z}$ .

### 2.1.1 Teorema Pembagian

Dalam teori bilangan elementer kita mengetahui teorema pembagian sebagaimana dijelaskan pada Teorema 2.1.1.

**Teorema 2.1.1** *Misalkan  $a$  dan  $b$  adalah dua bilangan bulat, maka terdapat bilangan bulat  $q$  dan  $r$  dengan  $0 \leq |r| < |b|$  dengan sifat*

$$a = b \cdot q + r.$$

*Dalam hal ini kita menuliskan  $q = a \text{ div } b$  dan  $r = a \text{ mod } b$ .*

---

**Contoh 2.1.2** Kita memiliki:

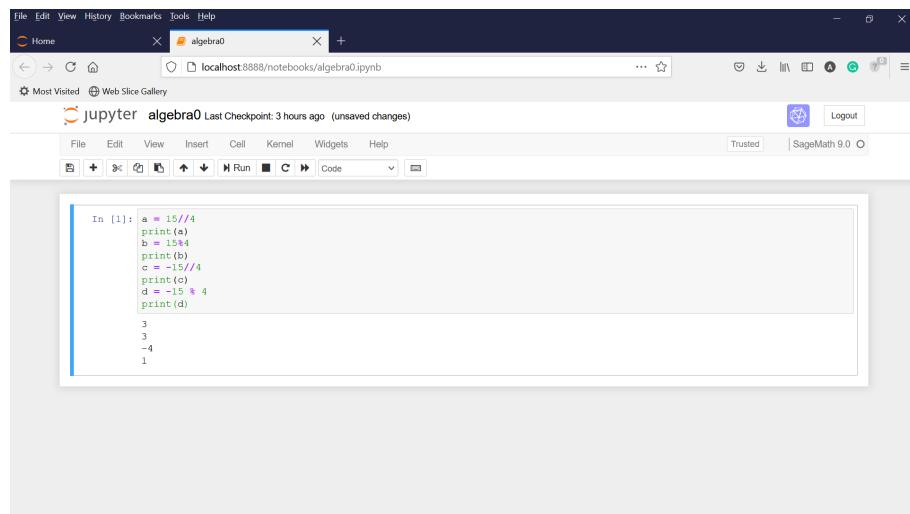
1.  $15 \text{ div } 4 = 3$  dan  $15 \text{ mod } 4 = 3$  karena  $15 = 4 \cdot 3 + 3$
2.  $-15 \text{ div } 4 = -4$  dan  $-15 \text{ mod } 4 = 1$  karena  $-15 = 4 \cdot (-4) + 1$

Pada SageMath operator div dinyatakan dengan // sedangkan operator mod dinyatakan dengan % (sebagaimana operator div dan mod pada Python 3):

```
a = 15//4
print(a)
b = 15%4
print(b)
c = -15//4
print(c)
d = -15 % 4
print(d)
```

Untuk memperoleh hasil kalkulasi, kita dapat menekan kombinasi tombol Ctrl+Enter atau kombinasi tombol Shift+Enter pada keyboard sehingga diperoleh hasil seperti pada Gambar 2.4.

**Catatan 2.1.3** Ada sedikit perbedaan jika kita menekan Ctrl+Enter dan Shift+Enter. Jika Ctrl+Enter ditekan, maka Jupyter notebook akan mengeksekusi perintah yang terdapat pada suatu sel, namun tidak berpindah ke sel berikutnya. Namun jika Shift+Enter ditekan, selain mengeksekusi perintah yang ada pada suatu sel kurSOR juga akan berpindah ke sel di bawahnya.



Gambar 2.4: Operasi div dan mod pada Jupyter notebook. Kita memiliki  $15 \text{ div } 4 = 3$ ,  $15 \text{ mod } 4 = 3$ ,  $-15 \text{ div } 4 = -4$ ,  $-15 \text{ mod } 4 = 1$ .

Kita juga dapat memperoleh nilai kalkulasi div dan mod secara simultan dengan perintah berikut:

```
(a,b) = 15.quo_rem(4)
print(a,b)
(c,d) = (-15).quo_rem(4)
print(c,d)
```

Dengan menekan Ctrl+Enter SageMath akan memberikan jawaban berikut:

```
3 3
-4 1
```

---

Pencarian dari nilai  $a \text{ div } b$  dan  $a \text{ mod } b$  juga dapat dilakukan secara simultan dengan fungsi `divmod(a, b)`, contohnya adalah sebagai berikut:

```
a = divmod(15, 4); print(a)
b = divmod(-15, 4); print(b)
```

Fungsi `divmod(a, b)` akan memberikan pasangan  $(q, r)$  dengan  $q = a \text{ div } b$  dan  $r = a \text{ mod } b$ . Hasil dari skrip sebelumnya adalah:

```
(3, 3)
(-4, 1)
```

**Latihan 2.1.4** Apa yang terjadi bila kita menuliskan perintah `-15.quo_rem(4)` pada SageMath?

**Latihan 2.1.5** Gunakan SageMath untuk menghitung nilai-nilai berikut dan verifikasi kebenarannya berdasarkan Teorema 2.1.1.

1.  $15 \text{ div } -4$
2.  $15 \text{ mod } -4$
3.  $-15 \text{ div } -4$
4.  $-15 \text{ mod } -4$

Selain konsep div dan mod, SageMath juga mengakomodasi pengecekan keterbagian pada bilangan bulat berdasarkan Definisi 2.1.6.

**Definisi 2.1.6** Misalkan  $a$  dan  $b$  adalah dua bilangan bulat,  $a$  dikatakan habis bagi  $b$ , ditulis dengan  $a|b$ , apabila  $k \cdot a = b$  untuk suatu bilangan bulat  $k$ . Notasi  $a \nmid b$  menyatakan bahwa  $a$  tidak habis bagi  $b$ .

Ketika  $a$  habis bagi  $b$  maka kita juga mengatakan bahwa  $b$  habis dibagi  $a$ . Pada SageMath kita dapat memeriksa apakah  $a|b$  dipenuhi menggunakan sintaks `a.divides(b)`. Sebagai contoh, kita dapat memeriksa apakah  $-23$  membagi  $92$  menggunakan `(-23).divides(92)`.

**Latihan 2.1.7** Menggunakan Definisi 2.1.6 dan sintaks yang telah dijelaskan, gunakan SageMath untuk memeriksa ekspresi-ekspresi berikut:

1.  $-23|92$
2.  $26|-130$
3.  $-3|0$
4.  $0|-3$
5.  $0|0$

## 2.1.2 Faktor Persekutuan Terbesar dan Kelipatan Persekutuan Terkecil

Diberikan dua bilangan bulat yang salah satunya tidak nol, maka kita dapat mendefinisikan faktor persekutuan terbesar (FPB) atau *greatest common divisor* (gcd) dari kedua bilangan tersebut sebagai bilangan terbesar yang habis membagi kedua bilangan yang ditinjau. Dengan perkataan lain, sebuah bilangan bulat  $d$  dikatakan sebagai gcd dari  $a$  dan  $b$  apabila  $d$  adalah bilangan bulat terbesar yang habis membagi  $a$  maupun  $b$ .

---

Kita mengenal algoritma Euclid sebagai salah satu algoritma yang dapat digunakan untuk menentukan gcd dari dua bilangan bulat (salah satunya tidak nol) secara efisien. Pada SageMath kita mendefinisikan  $\text{gcd}(0, 0) = 0$ . Ingat kembali bahwa algoritma Euclid menggunakan sifat penting berikut:

1.  $\text{gcd}(a, 0) = |a|$  untuk setiap  $a \in \mathbb{Z}$  yang tidak nol.
2.  $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$ .

Pada SageMath, gcd dari dua bilangan bulat dapat dicari menggunakan fungsi `gcd` sebagai berikut:

```
a = gcd(1030,1070); print(a)
b = gcd(1024,4444); print(b)
c = gcd(1234,1324); print(c)
```

Dengan menekan Ctrl+Enter pada Jupyter notebook, maka kita memperoleh jawaban berikut:

```
10
4
2
```

Selain perhitungan faktor persekutuan terbesar, SageMath juga dapat digunakan untuk menghitung kelipatan persekutuan terkecil dari dua bilangan bulat positif. Ingat kembali bahwa sebuah bilangan bulat positif  $c$  dikatakan sebagai kelipatan persekutuan terkecil (KPK) atau *least common multiple* (lcm) dari dua bilangan bulat positif  $a$  dan  $b$  apabila  $c$  adalah bilangan terkecil yang habis dibagi oleh  $a$  dan  $b$ . Pada SageMath, nilai  $\text{lcm}(a, b)$  ekuivalen dengan kelipatan persekutuan terkecil dari  $|a|$  dan  $|b|$ . Khusus untuk nilai  $a = b = 0$  SageMath akan memberikan  $\text{lcm}(0, 0) = 0$ . Ini berarti lcm dapat dihitung untuk setiap bilangan bulat  $a$  dan  $b$ . Dalam teori bilangan kita juga mengetahui hubungan  $\text{lcm}(a, b) = \frac{ab}{\text{gcd}(a,b)}$  ketika  $a$  dan  $b$  keduanya positif.

Contoh perhitungan lcm pada SageMath adalah sebagai berikut:

```
x = lcm(1030,1070); print(x)
y = lcm(1024,4444); print(y)
z = lcm(1234,1324); print(z)
u = lcm(3,-4); print(u)
v = lcm(-3,-4); print(v)
w = lcm(0,0); print(w)
```

Dengan menekan Ctrl+Enter kita memperoleh jawaban berikut:

```
110210
1137664
816908
12
12
0
```

Secara matematis, kita dapat menghitung gcd maupun lcm untuk tiga bilangan atau lebih. Diberikan  $n$  bilangan bulat  $a_1, a_2, \dots, a_n$ , maka:

1.  $\text{gcd}(a_1, a_2, \dots, a_n)$  adalah bilangan bulat terbesar yang habis bagi semua bilangan  $a_1, a_2, \dots, a_n$ ,
2.  $\text{lcm}(a_1, a_2, \dots, a_n)$  adalah bilangan bulat terkecil yang merupakan kelipatan positif dari  $a_1, a_2, \dots, a_n$ .

Secara matematis, kita memiliki:

$$\text{gcd}(a_1, a_2, a_3) = \text{gcd}(\text{gcd}(a_1, a_2), a_3), \quad (2.1)$$

$$\text{lcm}(a_1, a_2, a_3) = \text{lcm}(\text{lcm}(a_1, a_2), a_3). \quad (2.2)$$

---

Persamaan (2.1) dan (2.2) memberikan ide metode rekursif untuk menentukan gcd maupun lcm dari sebuah *list* yang memuat satu atau lebih bilangan dengan cara berikut:

1. Jika *list*  $L$  hanya memuat satu bilangan saja, misalkan  $L = [a_1]$ , maka  $\text{gcd}(L) = \text{lcm}(L) = |a_1|$ .
2. Jika *list*  $L$  memuat dua atau lebih bilangan, maka  $\text{gcd}(L) = \text{gcd}(L[0], \text{gcd}(L[1 :]))$  dan  $\text{lcm}(L) = \text{lcm}(L[0], \text{lcm}(L[1 :]))$ , dengan  $L[1 :]$  menyatakan *list* yang diperoleh dari *list*  $L$  dengan membuang elemen pertamanya (elemen ke-0).

### Latihan 2.1.8

1. Lengkapi fungsi `gcdlist` dan `lcmlist` berikut untuk menghitung gcd dan lcm dari sebuah *list* yang minimal memuat satu anggota.

```
def gcdlist(L):
    if len(L) == 1: return abs(L[0])
    else: return ... <isi bagian ini>
def lcmlist(L):
    if len(L) == 1: return abs(L[0])
    else: return ... <isi bagian ini>
```

2. Gunakan hasil pada 1 untuk menghitung nilai-nilai berikut:

- (a)  $\text{gcd}(240, 360, 600)$ .
- (b)  $\text{lcm}(240, 360, 600)$ .
- (c)  $\text{gcd}(510, 340, 323, 255, 357, 153, 187, 119, 391, 527)$ .
- (d)  $\text{lcm}(510, 340, 323, 255, 357, 153, 187, 119, 391, 527)$ .

Pada kajian teori bilangan elementer kita juga mengetahui identitas Bézout yang dijelaskan pada Teorema 2.1.9.

**Teorema 2.1.9 (Identitas Bézout)** *Misalkan  $a$  dan  $b$  adalah dua bilangan bulat, maka terdapat dua bilangan bulat  $s$  dan  $t$  yang memenuhi*

$$s \cdot a + t \cdot b = \text{gcd}(a, b).$$

Pada Teorema 2.1.9 nilai  $s$  dan  $t$  selanjutnya dikatakan sebagai koefisien Bézout dan tidak tunggal. Sebagai contoh kita memiliki

$$\begin{aligned}\text{gcd}(6, 14) &= 2 = (-2) \cdot 6 + (1) \cdot 14 \\ &= (5) \cdot 6 + (-2) \cdot 14.\end{aligned}$$

Nilai dari koefisien Bézout dapat diperoleh melalui algoritma Euclid yang diperluas (*extended Euclid's algorithm*). Pada SageMath nilai koefisien Bézout dari dua bilangan bulat  $a$  dan  $b$  dapat diperoleh melalui perintah `xgcd(a, b)`, contohnya adalah sebagai berikut:

```
a = xgcd(6,14); print(a)
b = xgcd(633,331); print(b)
c = xgcd(97,13); print(c)
d = xgcd(0,0); print(d)
```

Jika kita mengeksekusi perintah dengan menekan `Ctrl+Enter` maka kita memperoleh jawaban berikut:

```
(2, -2, 1)
(1, -137, 262)
(1, -2, 15)
(0, 0, 0)
```

Perintah `xgcd(a, b)` memberikan keluaran berupa tripel  $(d, s, t)$  yang memenuhi  $s \cdot a + t \cdot b = d = \gcd(a, b)$ . Kita dapat mengambil nilai individu dari  $s$ ,  $t$ , maupun  $\gcd(a, b)$  dengan memperlakukan keluaran `xgcd(a, b)` sebagai suatu tripel. Perhatikan contoh berikut:

```
bezout = xgcd(18, 27); print(bezout)
gcd = bezout[0]; print(gcd)
s = bezout[1]; print(s)
t = bezout[2]; print(t)
```

Keluaran dari skrip di atas adalah:

```
(9, -1, 1)
9
-1
1
```

Perhatikan bahwa  $\gcd(18, 27) = 9 = -1 \cdot 18 + 1 \cdot 27$ .

### 2.1.3 Bilangan Prima dan Faktorisasi Prima

Bilangan prima merupakan salah satu objek esensial dalam kriptografi, khususnya kriptografi asimetrik. Biasanya bilangan prima hanya dikaji pada domain bilangan bulat positif saja. Dalam hal ini, suatu bilangan bulat positif  $p > 1$  dikatakan sebagai prima apabila faktornya (bilangan yang habis bagi  $p$ ) hanyalah 1 dan  $p$  saja. Untuk memeriksa apakah suatu bilangan bulat  $n$  merupakan bilangan prima atau bukan kita dapat menggunakan method `.is_prime()` pada SageMath sebagaimana contoh berikut:

```
a = 3.is_prime(); print(a)
b = 17.is_prime(); print(b)
c = 2003.is_prime(); print(c)
d = 2011.is_prime(); print(d)
e = 2019.is_prime(); print(e)
f = 2021.is_prime(); print(f)
g = (-3).is_prime(); print(g)
h = (-17).is_prime(); print(h)
```

Dengan menjalankan skrip yang telah dibuat, SageMath memberikan keluaran berikut:

```
True
True
True
True
False
False
False
False
```

Dari keluaran yang dihasilkan kita melihat bahwa:

1. 3, 17, 2003, dan 2017 adalah bilangan-bilangan prima.
2.  $-3, -17, 2019$ , dan 2021 bukan bilangan prima. Perhatikan bahwa method `.is_prime()` hanya meninjau domain bilangan bulat positif saja, sehingga  $-3$  dan  $-17$  tidak dianggap sebagai bilangan prima.

---

**Latihan 2.1.10** Kita melihat bahwa *method* `.is_prime()` hanya meninjau domain himpunan bilangan bulat positif. Misalkan kita menggunakan definisi berikut: sebuah bilangan bulat  $p$  dikatakan prima apabila faktor (pembagi) dari  $p$  hanyalah  $\pm 1$  dan  $\pm p$ . Buatlah sebuah fungsi `prima(n)` yang memberikan keluaran `True` jika dan hanya jika  $n$  adalah bilangan prima berdasarkan definisi ini. Sebagai contoh, jika kita menjalankan fungsi `prima(n)` untuk beberapa nilai  $n$  berikut:

```
a = prima(3); print(a)
b = prima(17); print(b)
c = prima(2003); print(c)
d = prima(2011); print(d)
e = prima(2019); print(e)
f = prima(2021); print(f)
g = prima(-3); print(g)
h = prima(-17); print(h)
```

maka kita akan memperoleh keluaran berikut:

```
True
True
True
True
False
False
True
True
```

Kita juga dapat membangkitkan bilangan prima pada selang  $[2, n]$  secara acak pada SageMath menggunakan perintah `random_prime(n, proof=True)` atau `random_prime(n, proof=False)`. Ketika kita menggunakan parameter `proof=True` maka bilangan yang dibangkitkan sudah pasti merupakan bilangan prima, namun jika kita menggunakan parameter `proof=False` maka ada kemungkinan—meskipun sangat kecil—bahwa bilangan yang dibangkitkan bukan bilangan prima. Hal ini karena perintah `random_prime` menggunakan algoritma probabilistik untuk membangkitkan bilangan prima.<sup>1</sup> Meskipun demikian eksekusi dari perintah `random_prime(n, proof=True)` lebih lambat bila dibandingkan dengan eksekusi dari perintah `random_prime(n, proof=False)`.

Misalkan kita menuliskan perintah berikut untuk membangkitkan 10 bilangan prima berbeda antara 2 sampai dengan  $10^{15}$  yang dilabeli dengan  $a, b, \dots, j$ :

```
a = random_prime(10^15, proof=False); print(a); print(a.is_prime())
b = random_prime(10^15, proof=False); print(b); print(b.is_prime())
c = random_prime(10^15, proof=False); print(c); print(c.is_prime())
d = random_prime(10^15, proof=False); print(d); print(d.is_prime())
e = random_prime(10^15, proof=False); print(e); print(e.is_prime())
f = random_prime(10^15, proof=False); print(f); print(f.is_prime())
g = random_prime(10^15, proof=False); print(g); print(g.is_prime())
h = random_prime(10^15, proof=False); print(h); print(h.is_prime())
i = random_prime(10^15, proof=False); print(i); print(i.is_prime())
j = random_prime(10^15, proof=False); print(j); print(j.is_prime())
```

Jika kita menjalankan skrip di atas, maka SageMath secara acak akan membangkitkan bilangan prima secara probabilistik. Hasil keluaran dari suatu eksekusi dapat berbeda dengan hasil keluaran eksekusi lainnya, meskipun skrip yang digunakan sama. Bilangan yang dihasilkan mungkin saja bukan bilangan prima, namun kemungkinannya sangat kecil. Jika kita menginginkan bilangan yang dibangkitkan adalah pasti bilangan prima, maka kita dapat mengganti parameter `proof=False` dengan parameter `proof=True`. Selain itu eksekusi perintah dari

---

<sup>1</sup>Penjelasan lebih jauh dapat dilihat di [2, Subbab 3.4]

---

`random_prime(n, proof=False)` (maupun `random_prime(n, proof=True)`) akan memberikan hasil yang berbeda-beda untuk setiap panggilan yang dilakukan.

Untuk membangkitkan bilangan prima pada selang  $[a, b - 1]$  kita dapat menggunakan perintah `prime_range(a, b)` pada SageMath. Keluaran dari perintah ini adalah sebuah *list* yang berisi bilangan-bilangan prima pada selang  $[a, b - 1]$ . Jika  $a > b - 1$  atau tidak ada bilangan prima pada selang yang dimaksud maka SageMath akan memberikan sebuah *list* kosong. Namun metode ini memiliki keterbatasan karena tidak dapat digunakan untuk memberikan bilangan prima yang nilainya lebih dari 436 273 290. Untuk mengakalinya, kita dapat membuat skrip berikut:

```
def prime_list(a,b) :  
    L = []  
    for i in range(a,b+1):  
        if Integer(i).is_prime() == True: L.append(Integer(i))  
    return L
```

Perlu diingat bahwa tipe data bilangan bulat (*integer*) secara *default* pada Python 3 biasa dan tipe data bilangan bulat secara *default* pada SageMath berbeda. Tipe data bilangan bulat pada Python 3 secara *default* adalah '`int`' sedangkan tipe data bilangan bulat pada SageMath secara *default* adalah '`sage.rings.integer.Integer`'. Mengingat enumerasi pada *for loop* dilakukan dalam Python 3 sedangkan *method .is\_prime()* mengharuskan tipe data bilangan bulat pada SageMath, maka kita perlu mengonversi bilangan bulat *i* pada Python 3 menjadi bilangan bulat `Integer(i)` pada SageMath. Hal ini dapat dilakukan cukup mudah dengan mendefinisikan `Integer(i)` untuk setiap bilangan bulat *i* yang disimpan dalam format tipe data Python 3.

Untuk setiap bilangan bulat *n*, kita dapat menentukan bilangan prima terkecil yang lebih besar dari *n* dengan perintah `next_prime(n)`. Kemudian untuk setiap bilangan bulat *n*, kita juga dapat menentukan bilangan prima terbesar yang lebih kecil dari *n* dengan perintah `previous_prime(n)`. Hal ini dicontohkan pada skrip berikut:

```
a = next_prime(1000); print(a)  
b = previous_prime(1000); print(b)
```

Hasil eksekusi dari skrip ini adalah:

```
1009  
997
```

Selain memeriksa apakah sebuah bilangan bulat bersifat prima atau tidak serta membangkitkan bilangan-bilangan prima pada suatu interval tertentu, pada SageMath kita juga dapat melakukan faktorisasi prima menggunakan *method .factor()*. *Method* ini memberikan faktorisasi prima sesuai dengan Teorema Fundamental Aritmetika pada Teorema 2.1.11.

**Teorema 2.1.11 (Teorema Fundamental Aritmetika)** *Misalkan  $n$  adalah sebuah bilangan bulat positif, maka  $n$  dapat ditulis sebagai  $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$  dengan  $p_1, p_2, \dots, p_k$  adalah  $k \geq 1$  bilangan prima yang berbeda-beda dan  $a_1, a_2, \dots, a_k$  adalah  $k$  bilangan bulat nonnegatif.*

Misalkan kita mengetikkan perintah berikut pada suatu sel di SageMath:

```
print(2021.factor())  
print(3000.factor())  
print(3011.factor())
```

SageMath akan memberikan keluaran berikut:

```
43 * 47  
2^3 * 3 * 5^3  
3011
```

Dari hasil ini kita memperoleh faktorisasi prima berikut:

- 
1.  $2021 = 43 \times 47$
  2.  $3000 = 2^3 \times 3 \times 5^3$
  3.  $3011 = 3011$

Pada contoh ini 3011 adalah bilangan prima. Lebih jauh, SageMath menyimpan faktorisasi prima dari suatu bilangan bulat dalam format *list* yang setiap elemennya adalah pasangan terurut dengan komponen pertamanya adalah bilangan prima dan komponen keduanya adalah pangkat yang bersesuaian dengan bilangan prima tersebut. Hal ini berarti jika suatu bilangan prima memiliki faktorisasi prima  $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$ , maka SageMath menyimpan faktorisasi prima tersebut dalam *list* dengan bentuk  $[(p_1, a_1), (p_2, a_2), \dots, (p_k, a_k)]$ . Perhatikan contoh berikut:

```
factors = 3000.factor()
print(factors)
print(list(factors))
suku_satu = factors[0]; print(suku_satu)
suku_dua = factors[1]; print(suku_dua)
suku_tiga = factors[2]; print(suku_tiga)
prima_satu = factors[0][0]; print(prima_satu)
pangkat_satu = factors[0][1]; print(pangkat_satu)
```

Keluaran dari skrip ini adalah:

```
2^3 * 3 * 5^3
[(2, 3), (3, 1), (5, 3)]
(2, 3)
(3, 1)
(5, 3)
2
3
```

### Latihan 2.1.12

1. Gunakan SageMath untuk mencari bilangan-bilangan prima pada rentang  $[10^{18} - 1000, 10^{18}]$ .
2. Gunakan metode yang telah dijelaskan untuk membangkitkan dua bilangan prima yang terdiri atas 8 digit. Misalkan bilangan pertama kita sebut sebagai  $p$  dan bilangan kedua kita sebut sebagai  $q$ .
3. Dari hasil pada 2 untuk mencari  $s$  dan  $t$  yang memenuhi  $s \cdot p + t \cdot q = \gcd(p, q)$ .
4. Carilah faktorisasi prima dari 4 598 037 234.
5. Gunakan hasil pada 4 untuk menuliskan semua faktor prima dari 4 598 037 234.

## 2.2 Grup

Grup merupakan salah satu struktur aljabar yang sangat penting dalam kriptografi modern. Pada subtopik ini kita akan meninjau kembali definisi grup dan beberapa contohnya. Kita juga akan meninjau representasi dari grup permutasi pada SageMath yang berguna dalam penerapan sandi transposisi (atau sandi permutasi) yang akan dibahas pada Topik 3.

## 2.2.1 Definisi Grup dan Contohnya

Misalkan  $G$  adalah sebuah himpunan tak kosong yang dilengkapi dengan operasi biner  $* : G \times G \rightarrow G$ . Dalam hal ini operasi  $*$  memetakan pasangan terurut  $(a, b)$  ke sebuah elemen  $(a * b) \in G$ . Definisi grup  $(G, *)$  dijelaskan pada Definisi 2.2.1.

**Definisi 2.2.1 (Definisi Grup)** *Sebuah struktur  $(G, *)$  dengan  $G$  adalah himpunan tak kosong yang dilengkapi dengan operasi biner  $* : G \times G \rightarrow G$  dikatakan sebagai suatu grup apabila memenuhi ketiga aksioma berikut:*

1. operasi  $*$  bersifat asosiatif, yaitu  $(a * b) * c = a * (b * c)$  untuk setiap  $a, b, c \in G$ ,
2. terdapat elemen  $e \in G$  yang disebut elemen identitas dengan sifat  $a * e = e * a = a$  untuk setiap  $a \in G$ , dan
3. untuk setiap  $a \in G$  terdapat  $a^{-1} \in G$  (disebut sebagai invers dari  $a$ ) sehingga  $a * a^{-1} = a^{-1} * a = e$ , dengan  $e$  adalah elemen identitas.

Jika sebuah grup  $(G, *)$  memiliki sifat  $a * b = b * a$  untuk setiap  $a, b \in G$  (atau sifat komutatif) maka  $G$  dikatakan grup Abelian (grup komutatif).

**Contoh 2.2.2** Misalkan kita memiliki himpunan  $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$  yang dilengkapi dengan operasi  $+_5$  dengan definisi  $a +_5 b = (a + b) \bmod 5$  untuk setiap  $a, b \in \mathbb{Z}_5$ . Sebagai contoh  $3 +_5 4 = (3 + 4) \bmod 5 = 2$ . Kita dapat membangun tabel penjumlahan  $+_5$  pada  $\mathbb{Z}_5$  sebagai berikut:

| $+_5$ | 0 | 1 | 2 | 3 | 4 |
|-------|---|---|---|---|---|
| 0     | 0 | 1 | 2 | 3 | 4 |
| 1     | 1 | 2 | 3 | 4 | 0 |
| 2     | 2 | 3 | 4 | 0 | 1 |
| 3     | 3 | 4 | 0 | 1 | 2 |
| 4     | 4 | 0 | 1 | 2 | 3 |

Pada contoh ini  $(\mathbb{Z}_5, +_5)$  adalah contoh grup komutatif.

Pada SageMath kita dapat membangun tabel penjumlahan pada  $\mathbb{Z}_5$  dengan operasi  $+_5$  yang telah dijelaskan dengan perintah berikut:

```
Z5 = Integers(5); print(Z5)
Z5.addition_table(names='elements')
```

Sebenarnya  $\mathbb{Z}_5$  juga dapat ditinjau sebagai ring (atau gelanggang) bilangan bulat dalam modulo 5 jika kita juga meninjau operasi perkalian pada  $\mathbb{Z}_5$ . Hal ini akan dijelaskan pada kajian berikutnya pada Subtopik 2.3. Dengan menjalankan perintah yang telah ditulis pada SageMath, kita akan memperoleh keluaran berikut:

```
Ring of integers modulo 5

+ 0 1 2 3 4
+-----+
0| 0 1 2 3 4
1| 1 2 3 4 0
2| 2 3 4 0 1
3| 3 4 0 1 2
4| 4 0 1 2 3
```

Salah satu contoh struktur matematika yang bukan merupakan grup adalah  $(\mathbb{Z}_5, \times_5)$  dengan  $\times_5$  adalah operasi yang didefinisikan sebagai berikut:  $a \times_5 b = (a \times_5 b) \bmod 5$  untuk setiap

---

$a, b \in \mathbb{Z}_5$ . Hal ini karena 0 tidak memiliki invers terhadap operasi  $\times_5$ , yaitu, tidak terdapat  $0^{-1}$  sehingga  $0 \times 0^{-1} = 1$  (kita dapat membuktikan bahwa 1 adalah elemen identitas pada struktur  $(\mathbb{Z}_5, \times_5)$  karena  $a \times_5 1 = 1 \times_5 a = a$  untuk setiap  $a \in \mathbb{Z}_5$ ). Kita dapat mengonstruksi tabel perkalian atas struktur  $(\mathbb{Z}_5, \times_5)$  dengan SageMath menggunakan perintah berikut:

```
Z5 = Integers(5); print(Z5)
Z5.multiplication_table(names='elements')
```

SageMath memberikan keluaran berikut:

```
Ring of integers modulo 5

*   0 1 2 3 4
+-----
0| 0 0 0 0 0
1| 0 1 2 3 4
2| 0 2 4 1 3
3| 0 3 1 4 2
4| 0 4 3 2 1
```

Perhatikan bahwa jika kita mendefinisikan struktur  $(\mathbb{Z}_5^*, \times_5)$  dengan  $\mathbb{Z}_5^* = \mathbb{Z}_5 \setminus 0$  maka kita akan memperoleh struktur grup komutatif.

## 2.2.2 Grup Permutasi (Grup Simetri)

Pada subtopik ini akan dibahas mengenai grup permutasi (terkadang juga disebut grup simetri) dan penerapan komputasinya pada SageMath. Pada kriptografi grup permutasi berguna dalam mengkaji sandi atau sistemkripto yang terkait dengan permutasi, seperti sandi transposisi (*transposition cipher*). Kajian teori pada tutorial ini disarikan dari [11, 12] sedangkan penerapannya pada SageMath dirangkum dari [9, Bab 5]. Sebuah permutasi pada himpunan  $S$  adalah pemetaan yang injektif dari  $S$  ke  $S$ . Secara khusus kita akan meninjau himpunan  $S$  yang memuat tepat  $n$  anggota, yaitu  $\{1, 2, \dots, n\}$ , yang selanjutnya dinotasikan dengan  $S_n$ . Grup permutasi dengan  $n$  elemen adalah struktur aljabar  $(S_n, \circ)$  dengan  $\circ$  adalah operasi komposisi pada  $S_n$ . Elemen-elemen pada  $S_n$  dapat dinyatakan dalam bentuk dua baris. Pada baris pertama kita menuliskan  $1, 2, \dots, n$  dan pada baris kedua kita menuliskan permutasi dari elemen-elemen pada baris pertama. Misalkan  $\phi \in S_n$ , maka  $\phi$  ditulis sebagai:

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ \phi(1) & \phi(2) & \cdots & \phi(n) \end{pmatrix}$$

Misalkan kita memiliki  $\phi \in S_4$  dengan definisi berikut:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix} \tag{2.3}$$

Ini berarti  $\phi \in S_4$  adalah pemetaan dengan definisi  $\phi(1) = 2$ ,  $\phi(2) = 4$ ,  $\phi(3) = 3$ , dan  $\phi(4) = 1$ , atau dapat juga ditulis  $1 \mapsto 2$ ,  $2 \mapsto 4$ ,  $3 \mapsto 3$ , dan  $4 \mapsto 1$ . Elemen-elemen pada  $S_n$  juga dapat ditulis dalam notasi siklus (*cycle notation*). Dalam hal ini kita menuliskan permutasi (2.3) sebagai  $(1 \ 2 \ 4)$  karena permutasi  $\phi$  dapat ditulis sebagai  $1 \mapsto 2 \mapsto 4$  dan  $3 \mapsto 3$ . Mengingat  $3 \mapsto 3$  adalah suatu permutasi identitas maka kita tidak perlu menuliskannya. Bentuk  $(1 \ 2 \ 4)$  juga dapat ditulis sebagai  $(2 \ 4 \ 1)$  maupun  $(4 \ 1 \ 2)$ .

Invers dari suatu permutasi  $\phi \in S_n$  adalah  $\phi^{-1} \in S_n$  dengan sifat  $\phi^{-1}(i) = j$  jika dan hanya jika  $\phi(j) = i$ . Sebagai contoh, untuk  $\phi \in S_4$  yang dijelaskan pada (2.3) kita memperoleh

---

$\phi^{-1}(1) = 4, \phi^{-1}(2) = 1, \phi^{-1}(3) = 3$ , dan  $\phi^{-1}(4) = 2$ , sehingga kita memiliki:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}$$

yang dapat ditulis dalam notasi siklus sebagai  $(1\ 4\ 2)$ ,  $(4\ 2\ 1)$ , ataupun  $(2\ 1\ 4)$ . Perhatikan bahwa kita cukup “membalik” notasi siklus dari  $\phi$  untuk memperoleh notasi siklus dari  $\phi^{-1}$ , sebagai contoh  $(1\ 2\ 4)^{-1} = (4\ 2\ 1)$ .

Operasi  $\circ$  pada  $S_n$  dilakukan dengan aturan komposisi fungsi seperti biasanya, yaitu dari kanan ke kiri.<sup>2</sup> Misalkan kita memiliki dua permutasi  $\alpha$  dan  $\beta$  dengan definisi berikut:  $\alpha = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix}$ ,  $\beta = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$ , maka kita memiliki:

$$\alpha \circ \beta = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix} \quad (2.4)$$

sedangkan

$$\beta \circ \alpha = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 3 & 1 \end{pmatrix}. \quad (2.5)$$

Operasi pada (2.4) dapat ditulis dalam notasi siklus sebagai  $\alpha \circ \beta = (1\ 2\ 4\ 3) \circ (1\ 3)(2\ 4) = (1)(2\ 3)(4) = (2\ 3)$ , sedangkan operasi pada (2.5) dapat ditulis dalam notasi siklus sebagai  $\beta \circ \alpha = (1\ 3)(2\ 4) \circ (1\ 2\ 4\ 3) = (14)(2)(3) = (1\ 4)$ . Dari hasil ini kita melihat bahwa operasi  $\circ$  pada  $S_n$  tidak bersifat komutatif.

Grup permutasi  $(S_n, \circ)$  memuat  $n!$  anggota sebagaimana dijelaskan pada [12, Teorema 6.2]. Sebagai contoh, elemen-elemen dari  $S_6$  adalah:

1.  $\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} = (1)(2)(3) = (1) = (2) = (3)$  yang merupakan permutasi identitas
2.  $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} = (1\ 2\ 3)$
3.  $\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} = (1\ 3\ 2)$
4.  $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} = (1\ 2)$
5.  $\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} = (1\ 3)$
6.  $\begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} = (2\ 3)$

Pada SageMath kita dapat mengonstruksi grup  $S_n$  dengan perintah `SymmetricGroup(n)`. Misalkan kita ingin mengonstruksi grup  $S_3$  pada SageMath, maka kita menuliskan perintah berikut:

```
S3 = SymmetricGroup(3)
print(S3)
for x in S3:
    print(x)
```

---

<sup>2</sup>Beberapa referensi memberikan definisi komposisi yang berbeda, yaitu komposisi dilakukan dari kiri ke kanan.

---

Keluaran dari perintah ini adalah:

```
Symmetric group of order 3! as a permutation group
()
(1, 3, 2)
(1, 2, 3)
(2, 3)
(1, 3)
(1, 2)
```

Pada SageMath permutasi identitas ditulis dengan () sedangkan permutasi lain dalam notasi siklus dapat dinyatakan dalam bentuk tupel, misalnya  $(1\ 3\ 2)$  ditulis  $(1, 3, 2)$ . Kita memiliki beberapa alternatif dalam menyajikan sebuah elemen  $\phi \in S_n$  pada SageMath. Sebagai contoh perhatikan skrip berikut:

```
alpha1 = S3("(1,2,3)"); print(alpha1)
alpha2 = S3([(1,2,3)]); print(alpha2)
alpha3 = S3([2,3,1]); print(alpha3)
```

Pada skrip tersebut kita menuliskan elemen  $\alpha = (1\ 2\ 3) = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \in S_3$  dalam tiga cara, yaitu dalam format  $S3("(1,2,3)")$ ,  $S3([(1,2,3)])$ , dan  $S3([2,3,1])$ . Pada format terakhir kita menyajikan  $\alpha \in S_3$  dalam bentuk *list* yang komponen-komponennya secara terurut adalah  $\alpha(i)$  untuk  $1 \leq i \leq 3$ . Bentuk ini juga dikenal dengan istilah *bottom row format*. Untuk semua format yang digunakan, SageMath memberikan keluaran berikut:

```
(1, 2, 3)
(1, 2, 3)
(1, 2, 3)
```

Hasil ini berarti representasi keluaran yang berupa suatu elemen grup permutasi pada SageMath diberikan dalam format tupel sebagaimana yang telah kita lihat pada keluaran skrip sebelumnya.

Operasi  $\circ$  pada  $S_n$  direpresentasikan pada SageMath dengan operator  $*$ . Misalkan kita ingin menghitung nilai dari  $\alpha \circ \beta$  yang dijelaskan di (2.4) dan  $\beta \circ \alpha$  yang dijelaskan di (2.5). Kita dapat menuliskan skrip berikut:

```
S4 = SymmetricGroup(4)
print(S4)
alpha = S4([(1,2,4,3)]); print(alpha)
beta = S4([(1,3),(2,4)]); print(beta)
alphabeta = alpha * beta; print(alphabeta)
betaalpha = beta * alpha; print(betaalpha)
```

Eksekusi skrip ini memberikan hasil dari  $\alpha \circ \beta$  dan  $\beta \circ \alpha$  yang dinyatakan dalam notasi tupel:

```
Symmetric group of order 4! as a permutation group
(1, 2, 4, 3)
(1, 3) (2, 4)
(1, 4)
(2, 3)
```

Misalkan kita mendefinisikan  $\alpha^n = \underbrace{\alpha \circ \alpha \circ \cdots \circ \alpha}_{n \text{ suku}}$ . Sebagai contoh jika  $\alpha = (1\ 2\ 4\ 3)$ , maka kita memiliki  $\alpha^3 = (1\ 3\ 4\ 2)$  dan  $\alpha^4 = (1)(2)(3)(4)$ . Lebih lanjut, operasi invers pada  $S_n$  dapat dilakukan dengan sintaks  $\dots^{-1}$  pada SageMath. Kalkulasi-kalkulasi ini dapat dilakukan menggunakan skrip berikut:

```

alpha3 = alpha^3; print(alpha3)
alpha4 = alpha^4; print(alpha4)
alphainv = alpha^-1; print(alphainv)

```

SageMath memberikan hasil kalkulasi berikut:

```

(1, 3, 4, 2)
()
(1, 3, 4, 2)

```

Ingat kembali bahwa  $()$  berarti elemen identitas pada  $S_n$ . Kita juga dapat memakai SageMath untuk membangun tabel operasi  $\circ$  pada grup permutasi tertentu. Misalkan kita ingin membangun tabel operasi  $\circ$  pada  $S_3$ , maka kita dapat memakai skrip berikut:

```

S3 = SymmetricGroup(3)
print(S3)
S3.cayley_table(names='elements')

```

SageMath akan memberikan keluaran berikut:

```
Symmetric group of order 3! as a permutation group
```

| *         | ( )       | (2, 3)    | (1, 2)    | (1, 2, 3) | (1, 3, 2) | (1, 3)    |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| ( )       | ( )       | (2, 3)    | (1, 2)    | (1, 2, 3) | (1, 3, 2) | (1, 3)    |
| (2, 3)    | (2, 3)    | ( )       | (1, 2, 3) | (1, 2)    | (1, 3)    | (1, 3, 2) |
| (1, 2)    | (1, 2)    | (1, 3, 2) | ( )       | (1, 3)    | (2, 3)    | (1, 2, 3) |
| (1, 2, 3) | (1, 2, 3) | (1, 3)    | (2, 3)    | (1, 3, 2) | ( )       | (1, 2)    |
| (1, 3, 2) | (1, 3, 2) | (1, 2)    | (1, 3)    | ( )       | (1, 2, 3) | (2, 3)    |
| (1, 3)    | (1, 3)    | (1, 2, 3) | (1, 3, 2) | (2, 3)    | (1, 2)    | ( )       |

### Latihan 2.2.3

1. Gunakan SageMath untuk menghitung hasil dari operasi-operasi berikut pada  $S_4$ :

- (a)  $(1\ 3)(2\ 4) \circ (1\ 3\ 2\ 4)$
- (b)  $(1\ 3\ 2\ 4) \circ (1\ 3) \circ (2\ 4)$
- (c)  $[(1\ 3)(2\ 4) \circ (1\ 3\ 2\ 4)] \circ (2\ 3)(1\ 4)$
- (d)  $((1\ 3)(2\ 4) \circ [(1\ 3\ 2\ 4) \circ (2\ 3)(1\ 4)])$

2. Sebuah bilangan bulat  $q$  dikatakan sebagai orde dari sebuah elemen  $\alpha \in S_n$  apabila  $q$  adalah bilangan bulat positif terkecil yang memenuhi  $\alpha^q = (1)$ , dengan  $(1) \in S_n$  menyatakan elemen identitas. Tentukan orde dari setiap elemen pada  $S_4$ .

### 2.2.3 Grup Siklis

Misalkan  $G$  adalah sebuah grup. Suatu himpunan tak kosong  $H \subseteq G$  dikatakan sebagai subgrup dari  $G$  apabila  $H$  membentuk suatu grup. Jika  $H$  adalah subgrup dari  $G$ , maka  $H$  dan  $G$  memiliki elemen identitas yang sama. Lebih lanjut untuk memeriksa apakah suatu himpunan tak kosong  $H \subseteq G$  yang dilengkapi dengan operasi  $*$  merupakan suatu subgrup atau bukan kita dapat memeriksa dua sifat berikut: (1) untuk setiap  $a, b \in H$  maka  $a * b \in H$  dan (2) untuk setiap  $a \in H$  maka  $a^{-1} \in H$  [12, Teorema 7.1]. Dari sini kita juga dapat memiliki fakta pada Teorema 2.2.4.

**Teorema 2.2.4 (Teorema 4.3 pada [10])** Misalkan  $G$  adalah sebuah grup dan  $g \in G$  adalah sembarang elemen, maka himpunan

$$\langle g \rangle = \{g^k : k \in \mathbb{Z}\}$$

---

adalah subgrup dari  $G$ . Lebih lanjut  $\langle g \rangle$  merupakan subgrup terkecil yang memuat  $g$ . Kita juga mengatakan bahwa  $\langle g \rangle$  adalah subgrup siklis yang dibangkitkan oleh  $g$ .

Apabila  $G$  adalah sebuah grup dan terdapat  $g \in G$  dengan sifat  $\langle g \rangle = G$  maka  $G$  dinamakan sebagai grup siklis. Dalam hal ini  $g$  dikatakan sebagai pembangkit (*generator*) dari  $G$ .

### Contoh 2.2.5

1. Pada  $(\mathbb{Z}_5, +)$  kita memiliki  $\langle 1 \rangle = \mathbb{Z}_5$ . Jadi  $(\mathbb{Z}_5, +)$  adalah grup siklis dengan pembangkit 1. Kita memiliki  $1 + 1 = 2$ ,  $1 + 1 + 1 = 3$ ,  $1 + 1 + 1 + 1 = 4$ , dan  $1 + 1 + 1 + 1 + 1 = 0$ .

2. Misalkan  $\mathbb{Z}_5^* = \mathbb{Z}_5 \setminus \{0\}$ . Pada  $(\mathbb{Z}_5^*, \times)$  kita memiliki  $\langle 2 \rangle = \mathbb{Z}_5^*$ . Jadi  $(\mathbb{Z}_5^*, \times)$  adalah grup siklis dengan pembangkit 2. Kita memiliki  $2 \times 2 = 4$ ,  $2 \times 2 \times 2 = 3$ , dan  $2 \times 2 \times 2 \times 2 = 1$ .

Kita dapat membuktikan dengan mudah bahwa grup siklis juga merupakan grup Abelian. Pada SageMath, grup siklis dengan  $n$  elemen secara abstrak dapat dibangkitkan dengan sintaks  $G < g > = \text{AbelianGroup}([n])$  seperti pada skrip berikut:

```
G.<g> = AbelianGroup([5])
print(G)
print(G.list())
```

Skrip ini membangkitkan grup siklis  $\langle g \rangle = \{1, g, g^2, g^3, g^4\}$ . Keluaran dari skrip ini adalah:

```
Multiplicative Abelian group isomorphic to C5
(1, g, g^2, g^3, g^4)
```

Pada keluaran SageMath C5 berarti grup siklis dengan 5 anggota.

## 2.3 Dasar-dasar Ring dan *Field* pada SageMath

Ring dan *field* adalah contoh struktur aljabar yang dilengkapi dengan dua operasi. Pada sub-topik ini kita akan mengkaji dasar-dasar ring dan *field* pada SageMath yang mencakup bahasan tentang ring  $\mathbb{Z}_m$  dan *field*  $\mathbb{F}_p$  untuk suatu bilangan prima  $p$ .

### 2.3.1 Definisi Ring dan *Field* serta Beberapa Contohnya

Suatu ring (atau terkadang disebut gelanggang [13]) adalah struktur matematika  $(R, +, \cdot)$  dengan  $R$  adalah himpunan tak kosong yang memenuhi sifat:

1.  $(R, +)$  merupakan grup Abelian dengan elemen identitas 0.
2.  $(R, \cdot)$  memenuhi sifat asosiatif, yaitu  $(a \cdot b) \cdot c = a \cdot (b \cdot c)$  untuk setiap  $a, b, c \in R$ . Lebih lanjut kita menulis  $a \cdot b$  sebagai  $ab$  untuk meringkas penulisan.
3. Untuk setiap  $a, b, c \in R$  berlaku sifat distributif berikut:

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

Jika sebuah ring dilengkapi dengan elemen  $1 \neq 0$  yang merupakan identitas perkalian, yang memenuhi  $a \cdot 1 = 1 \cdot a = a$  untuk setiap  $a \in R$ , maka  $R$  disebut ring dengan elemen satuan (*ring with unity*). Sebuah ring dikatakan sebagai ring komutatif apabila  $a \cdot b = b \cdot a$  untuk setiap  $a, b \in R$ . Sebuah ring komutatif dengan elemen satuan dikatakan sebagai daerah integral (*integral domain*<sup>3</sup>) apabila kondisi  $a \cdot b = 0$  memberikan  $a = 0$  atau  $b = 0$ . Sebuah elemen

<sup>3</sup>Integral domain berarti sebuah domain yang sifat-sifatnya menyerupai himpunan bilangan bulat (*integer*).

---

tak nol  $a \in R$  dikatakan sebagai *unit* apabila terdapat  $a^{-1} \in R$  yang unik untuk setiap  $a$  dan memenuhi  $a \cdot a^{-1} = a^{-1} \cdot a = 1$ . Sebuah ring dikatakan ring pembagian (*division ring*) apabila setiap elemen tak nolnya merupakan unit. Sebuah ring pembagian yang komutatif selanjutnya dinamakan sebagai *field* (atau terkadang disebut lapangan [13]). Untuk memperjelas definisi-definisi yang telah diberikan, perhatikan Contoh 2.3.1.

### Contoh 2.3.1

1. *Struktur  $(\mathbb{Z}, +, \cdot)$  merupakan suatu ring. Lebih lanjut  $(\mathbb{Z}, +, \cdot)$  juga merupakan ring komutatif dan suatu daerah integral, karena solusi dari  $ab = 0$  adalah  $a = 0$  atau  $b = 0$ . Namun struktur ini bukan ring pembagian dan bukan pula suatu field.*
2. *Struktur  $(\mathbb{Z}_n, +, \cdot)$  merupakan suatu ring. Struktur ini juga merupakan ring komutatif. Untuk  $n > 1$  yang bukan prima struktur ini bukan daerah integral. Misalnya pada struktur  $(\mathbb{Z}_{12}, +, \cdot)$  salah satu dari solusi  $ab = 0$  adalah  $a = 3$  dan  $b = 4$ . Kita juga dapat menunjukkan dengan mudah bahwa  $(\mathbb{Z}_{12}, +, \cdot)$  bukan suatu ring pembagian (misalnya tidak ada  $3^{-1}$  sehingga  $3 \cdot 3^{-1} = 3^{-1} \cdot 3 = 1$ ) dan bukan pula suatu field. Untuk  $n > 1$  yang prima kita dapat membuktikan bahwa  $(\mathbb{Z}_n, +, \cdot)$  adalah daerah integral. Lebih lanjut kita dapat membuktikan bahwa  $(\mathbb{Z}_p, +, \cdot)$  juga merupakan field untuk setiap bilangan prima  $p > 1$ .*
3. *Struktur  $(\mathbb{Q}, +, \cdot)$ ,  $(\mathbb{R}, +, \cdot)$ , dan  $(\mathbb{C}, +, \cdot)$  adalah ring. Ketiganya adalah ring komutatif, daerah integral, ring pembagian, dan juga field.*

### 2.3.2 Ring $\mathbb{Z}$ dan $\mathbb{Z}_m$ pada SageMath

Ada setidaknya tiga cara berbeda untuk mendefinisikan ring  $\mathbb{Z}$  pada SageMath, yaitu menggunakan sintaks `IntegerRing()`, `Integers()`, dan `ZZ`. Perhatikan skrip berikut:

```
Ring1 = IntegerRing()
print(Ring1)
print(type(Ring1))
Ring2 = Integers()
print(Ring2)
print(type(Ring2))
Ring3 = ZZ
print(Ring3)
print(type(Ring3))
```

Jika kita menjalankan skrip ini pada SageMath maka kita akan memperoleh:

```
Integer Ring
<class 'sage.rings.integer_ring.IntegerRing_class'>
Integer Ring
<class 'sage.rings.integer_ring.IntegerRing_class'>
Integer Ring
<class 'sage.rings.integer_ring.IntegerRing_class'>
```

Pada SageMath bilangan bulat secara *default* memiliki tipe `sage.rings.integer.Integer`. Hal ini berbeda dengan tipe `int` pada Python 3. SageMath memperlakukan 10, 10.0, dan 20/2 dalam tipe yang berbeda-beda. Kita dapat mengujinya dengan skrip berikut:

```
a = 10
print(type(a))
b = 10.0
print(type(b))
c = 20/2
print(type(c))
```

Keluaran dari skrip tersebut adalah:

```
<class 'sage.rings.integer.Integer'>
<class 'sage.rings.real_mpfr.RealLiteral'>
<class 'sage.rings.rational.Rational'>
```

Secara matematis, ini berarti nilai 10 diperlakukan sebagai elemen  $\mathbb{Z}$ , 10.0 diperlakukan sebagai elemen  $\mathbb{R}$ , dan 20/2 diperlakukan sebagai elemen  $\mathbb{Q}$ . Pada SageMath operator == akan menguji kesamaan numerik. Hal ini dapat dilakukan meskipun tipe data yang digunakan berbeda-beda. Misalkan kita memiliki skrip berikut yang merupakan lanjutan dari skrip sebelumnya:

```
print(a == b)
print(b == c)
print(a == c)
print(type(a) == type(b))
print(type(b) == type(c))
print(type(a) == type(c))
```

Keluaran dari skrip ini adalah:

```
True
True
True
False
False
False
```

Pada SageMath kita dapat mengonversi string, bilangan real, bilangan rasional, bilangan heksadesimal, maupun bilangan oktal ke dalam bilangan bulat menggunakan fungsi Integer(...). Bilangan heksadesimal berbentuk string yang diawali oleh 0x sedangkan bilangan oktal berbentuk string yang diawali oleh 0o. Misalkan kita memiliki skrip berikut:

```
a = Integer('10'); print(type(a))
b = Integer(10.0); print(type(b))
c = Integer(20/2); print(type(c))
d = Integer('0x10'); print(type(d))
e = Integer('0o10'); print(type(e))
print(a);
print(b);
print(c);
print(d);
print(e);
```

Keluaran dari skrip ini adalah:

```

<class 'sage.rings.integer.Integer'>
<class 'sage.rings.integer.Integer'>
<class 'sage.rings.integer.Integer'>
<class 'sage.rings.integer.Integer'>
<class 'sage.rings.integer.Integer'>
10
10
10
16
8

```

Perhatikan bahwa bilangan 10 dalam notasi heksadesimal sama dengan 16 dalam notasi desimal, sedangkan bilangan 10 dalam dalam notasi oktal sama dengan 8 dalam notasi desimal. Dalam teori bilangan, kita memiliki Teorema 2.3.2 yang menjelaskan mengenai representasi dari suatu bilangan bulat positif  $n$  dalam suatu basis  $b > 1$ .

**Teorema 2.3.2 (Teorema 1 pada Subbab 4.2.2 di [14])** Misalkan  $b > 1$  adalah sebuah bilangan bulat, maka setiap bilangan bulat  $n > 1$  dapat dinyatakan secara tunggal (unik) sebagai

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

untuk suatu bilangan bulat nonnegatif  $k$  serta bilangan-bilangan bulat nonnegatif  $a_0, a_1, \dots, a_k$  yang kurang dari  $b$  dengan sifat  $a_k \neq 0$ .

Nilai dari  $a_k a_{k-1} \dots a_1 a_0$  pada Teorema 2.3.2 selanjutnya dikatakan sebagai representasi dari  $n$  dalam basis  $b$ . Untuk memperjelas biasanya kita menuliskan hal ini sebagai  $(a_k a_{k-1} \dots a_1 a_0)_b$ . Sebagai contoh representasi dari bilangan desimal 14 dalam basis 2, 3, dan 4, berturut-turut adalah  $(1110)_2$ ,  $(112)_3$ , dan  $(32)_4$ . Pada SageMath kita dapat mengonversi sebuah bilangan desimal  $n$  ke dalam representasinya dalam basis  $b$  menggunakan method .digits (base=b). Keluaran dari perintah n.digits (base=b) adalah sebuah list  $[a_0, a_1, \dots, a_{k-1}, a_k]$  dengan sifat  $(a_k a_{k-1} \dots a_1 a_0)_b = n$ . Perhatikan contoh skrip berikut:

```

a = 14.digits(base=2); print(a)
b = 14.digits(base=3); print(b)
c = 14.digits(base=4); print(c)

```

Keluaran dari skrip ini adalah:

```

[0, 1, 1, 1]
[2, 1, 1]
[2, 3]

```

Untuk mengubah list  $[a_0, a_1, \dots, a_{k-1}, a_k]$  menjadi sebuah bilangan desimal  $n$  (basis 10) maka kita dapat menggunakan sintaks Integers ( $[a_0, a_1, \dots, a_{k-1}, a_k], b$ ). Perhatikan contoh berikut:

```

desimal_a = Integer([0, 1, 1, 1], base=2); print(desimal_a)
desimal_b = Integer([2, 1, 1], base=3); print(desimal_b)
desimal_c = Integer([2, 3], base=4); print(desimal_c)

```

Keluaran dari skrip ini adalah bilangan bulat 14. Perhatikan bahwa SageMath merepresentasikan bilangan bulat dalam basis  $b$  dalam bentuk list  $[a_0, a_1, \dots, a_{k-1}, a_k]$  dan bukan list  $[a_k, a_{k-1}, \dots, a_0, a_1]$ .

Dalam teori bilangan dan aljabar kita mengetahui ring  $\mathbb{Z}_m$  sebagai himpunan yang memuat  $m$  bilangan bulat  $0, 1, \dots, m - 1$  yang dilengkapi dengan operasi penjumlahan dan perkalian dalam modulo  $m$ . Sebuah ring  $\mathbb{Z}_m$  di SageMath dapat didefinisikan dengan fungsi Integers(m) atau IntegerModRing(m). Pada SageMath Integers(0) memberikan ring  $\mathbb{Z}$  dan Integers(m) memberikan ring bilangan bulat modulo  $m$ . Perhatikan skrip berikut:

---

```

z0 = Integers(0); print(z0)
z1 = Integers(1); print(z1)
z2 = Integers(2); print(z2)
z3 = Integers(3); print(z3)
z4 = Integers(4); print(z4)

```

Keluaran dari skrip ini adalah:

```

Integer Ring
Ring of integers modulo 1
Ring of integers modulo 2
Ring of integers modulo 3
Ring of integers modulo 4

```

Operasi aritmetika pada  $\mathbb{Z}_m$  dapat kita lakukan dengan mendefinisikan *instances* yang merupakan elemen-elemen dari  $\mathbb{Z}_m$ . Misalkan kita mendefinisikan suatu ring  $R = \mathbb{Z}_m$ , maka setiap elemen pada  $\mathbb{Z}_m$  dapat kita definisikan pada SageMath menggunakan sintaks  $R(\dots)$ , di sini  $(\dots)$  dapat diisi dengan *instance* apa pun dan SageMath akan mengonversinya ke dalam elemen pada  $\mathbb{Z}_m$ . Operasi-operasi aritmetika juga dapat dilakukan secara natural menggunakan sintaks operasi aritmetika yang telah dijelaskan sebelumnya. Skrip berikut menjelaskan beberapa operasi aritmetika pada  $\mathbb{Z}_{10}$ :

```

z10 = Integers(10); print(z10)
a = z10(14); print(a)
b = z10(17); print(b)
c = a + b; print(c)
d = a - b; print(d)
e = a * d; print(e)
f = a^4; print(f)

```

Hasil komputasi dari skrip ini adalah:

```

Ring of integers modulo 10
4
7
1
7
8
6

```

Perhatikan bahwa  $4 \equiv 14 \pmod{10}$  dan  $7 \equiv 17 \pmod{10}$ .<sup>4</sup> Operasi penjumlahan, pengurangan, perkalian, dan perpangkatan dilakukan pada  $\mathbb{Z}_{10}$  atau dilakukan modulo 10. Misalnya  $14+17 \equiv 1 \pmod{10}$  dan  $14 \cdot 17 \equiv 8 \pmod{10}$ . Dari teori bilangan, kita mengetahui bahwa sebuah elemen  $a \in \mathbb{Z}_m$  memiliki invers perkalian (bilangan  $a^{-1}$  yang memenuhi  $a \cdot a^{-1} = a^{-1} \cdot a = 1$ ) jika dan hanya jika  $\gcd(a, m) = 1$ . Kita dapat menentukan invers pada  $\mathbb{Z}_m$  dengan cukup mudah pada SageMath. Perhatikan skrip berikut:

```

z10 = Integers(10); print(z10)
a = z10(27); print(a)
ainv = a^-1; print(ainv)

```

Keluaran dari skrip ini adalah:

```

Ring of integers modulo 10
7
3

```

Perhatikan bahwa  $27 \cdot 3 \equiv 1 \pmod{10}$ , dengan perkataan lain invers perkalian dari 27 di  $\mathbb{Z}_{10}$  adalah 3. Perhatikan pula bahwa  $7 \equiv 27 \pmod{10}$ .

---

<sup>4</sup>Notasi  $a \equiv b \pmod{n}$  berarti  $n|(a - b)$ .

---

Sebagaimana telah dijelaskan sebelumnya, secara aljabar kita mengetahui bahwa sebuah elemen  $a \in \mathbb{Z}_m$  memiliki invers perkalian di  $\mathbb{Z}_m$  jika dan hanya jika  $\gcd(a, m) = 1$ . Akibatnya banyaknya elemen berbeda pada  $\mathbb{Z}_m$  yang memiliki invers sama dengan banyaknya anggota himpunan  $\{0 \leq a \leq m - 1 : \gcd(a, m) = 1\}$ . Hal ini dinamakan dengan fungsi phi Euler (*Euler's phi function*) atau fungsi totient Euler (*Euler's totient function*) dan secara formal didefinisikan sebagai

$$\phi(m) = |\{0 \leq a \leq m - 1 : \gcd(a, m) = 1\}|. \quad (2.6)$$

Sebagai contoh, berdasarkan penjelasan pada (2.6) kita memiliki  $\phi(24) = 8$  (diperoleh dari  $\{1, 5, 7, 11, 13, 17, 19, 23\}$ ) dan  $\phi(7) = 6$  (diperoleh dari  $\{1, 2, \dots, 6\}$ ). Untuk setiap bilangan prima  $p$  kita memiliki  $\phi(p) = p - 1$ . Kita juga dapat membuktikan (dengan memanfaatkan Teorema Sisa Tiongkok yang akan dibahas pada Teorema 2.3.4) bahwa  $\phi(m, n) = \phi(m)\phi(n)$  apabila  $\gcd(m, n) = 1$ . Akibatnya jika  $m = \prod_{i=1}^n p_i^{e_i}$  dengan  $p_i$  untuk  $1 \leq i \leq n$  adalah bilangan-bilangan prima yang berbeda dan  $e_i$  untuk  $1 \leq i \leq n$  adalah bilangan-bilangan bulat positif, maka kita memperoleh

$$\phi(m) = \prod_{i=1}^n (p_i^{e_i} - p_i^{e_i-1}). \quad (2.7)$$

Pada SageMath perhitungan fungsi Euler phi dari sebuah bilangan bulat  $m$  dapat dilakukan menggunakan perintah `euler_phi(m)`.

**Latihan 2.3.3** Gunakan SageMath untuk melakukan komputasi berikut:

```

z10 = Integers(10); print(z10)
a = z10(6); print(a)
ainv = a^-1; print(ainv)

```

Apa keluaran dari skrip ini? Jelaskan keluaran yang dihasilkan oleh SageMath.

SageMath juga dapat digunakan untuk menyelesaikan kongruensi linier sebagaimana dijelaskan pada [8]. Ingat kembali bahwa sebuah kongruensi linier berbentuk  $ax \equiv b \pmod{m}$ . Kongruensi linier ini ekuivalen dengan persamaan  $ax = b$  di  $\mathbb{Z}_m$ . Pada SageMath solusi dari kongruensi linier ini dapat dicari menggunakan perintah `solve_mod(a*x == b, m)`. Misalkan kita memiliki skrip berikut:

```

a = solve_mod(8*x == 12, 21); print(a)
b = solve_mod(9*x == 12, 21); print(b)
c = solve_mod(9*x == 13, 21); print(c)

```

Baris pertama, kedua, dan ketiga dari skrip ini secara berurutan merupakan representasi kongruensi linier berikut:

1.  $8x \equiv 12 \pmod{21}$
2.  $9x \equiv 12 \pmod{21}$
3.  $9x \equiv 13 \pmod{21}$

Ingat kembali bahwa ekspresi  $p \equiv q \pmod{m}$  ekuivalen dengan  $m|(p - q)$ . Menggunakan SageMath, kita memperoleh solusi berikut:

```

[(12,)]
[(6,), (13,), (20,)]
[]

```

Perintah `solve_mod(a*x == b, m)` memberikan semua nilai  $x$  (jika ada) yang memenuhi kongruensi linier  $ax \equiv b \pmod{m}$  dalam notasi *list*. Lebih jauh nilai  $x$  yang diberikan

adalah semua  $x$  yang ada pada rentang  $[0, m - 1]$  atau merupakan elemen  $\mathbb{Z}_m$ . Dari hasil jawaban SageMath, kita memiliki

1. Solusi dari  $8x \equiv 12 \pmod{21}$  adalah  $x \equiv 12 \pmod{21}$ .
2. Solusi dari  $9x \equiv 13 \pmod{21}$  adalah  $x \equiv 6 \pmod{21}$ ,  $x \equiv 13 \pmod{21}$ , dan  $x \equiv 20 \pmod{21}$ .
3. Tidak ada solusi dari  $9x \equiv 13 \pmod{21}$ .

Ingat kembali bahwa kongruensi linier  $ax \equiv b \pmod{m}$  memiliki solusi jika dan hanya jika  $\gcd(a, m) | b$ . Lebih jauh, ketika solusi dari kongruensi linier ini ada, maka ada tepat  $d = \gcd(a, m)$  solusi berbeda untuk kongruensi linier tersebut.

SageMath juga dapat digunakan untuk menentukan solusi kongruensi linier untuk lebih dari satu variabel, sistem kongruensi linier, maupun kongruensi nonlinier. Misalkan kita ingin mencari seluruh pasangan  $(x, y)$  dengan  $x, y \in \mathbb{Z}_m$  yang memenuhi persamaan Diophantine  $ax + by = c$ , maka kita dapat memakai perintah `solve_mod(a*x+b*y==c, m)`. Sebagai contoh untuk mencari seluruh pasangan  $(x, y)$  yang memenuhi  $3x + 5y = 4$  di  $\mathbb{Z}_7$  (yang setara dengan kongruensi linier  $3x + 5y \equiv 4 \pmod{7}$ ) kita dapat menggunakan skrip berikut:

```
x, y = var('x, y')
a = solve_mod(3*x + 5*y == 4, 7); print(a)
```

SageMath akan memberikan keluaran berikut:

```
[ (0, 5), (1, 3), (2, 1), (3, 6), (4, 4), (5, 2), (6, 0) ]
```

Penulisan `x, y = var('x, y')` penting agar SageMath mengenal variabel apa saja yang akan dicari nilainya. Variabel ini dapat diganti menggunakan variabel yang lain. Kemudian misalkan kita ingin menyelesaikan sistem persamaan linier dua variabel pada  $\mathbb{Z}_m$  yang berbentuk  $ax + by = u$  atau ekuivalen dengan  $px + qy \equiv u \pmod{m}$ , maka kita dapat menggunakan perintah `solve_mod([a*x+b*y = u, p*x+q*y = v], m)`. Misalkan kita ingin mencari seluruh pasangan  $x, y$  dengan  $x, y \in \mathbb{Z}_{21}$  yang memenuhi  $\begin{cases} 9x + 2y = 2 \\ 3x + 2y = 11 \end{cases}$  maka kita dapat menuliskan skrip berikut:

```
x, y = var('x, y')
solve_mod( [9*x + 2*y == 2, 3*x + 2*y == 11], 21)
```

SageMath akan memberikan jawaban berikut:

```
[ (9, 13), (16, 13), (2, 13) ]
```

SageMath juga dapat dipakai untuk menyelesaikan kongruensi nonliner. Misalkan kita ingin mencari semua solusi dari kongruensi nonliner  $2x^2 + 3y^2 = 5$  di  $\mathbb{Z}_7$ , maka kita dapat menggunakan skrip berikut:

```
x, y = var('x,y')
a = solve_mod([2*x^2+3*y^2 == 5], 7); print(a)
```

Keluaran dari skrip ini adalah:

```
[ (0, 2), (0, 5), (1, 1), (1, 6), (6, 1), (6, 6) ]
```

SageMath juga dapat menyelesaikan sistem kongruensi linier yang diberikan dalam modulus yang berbeda-beda jika modulus-modulus tersebut bersifat relatif prima satu sama lain.<sup>5</sup> Pada teori bilangan dan aljabar kita mengenal Teorema Sisa Tiongkok (*Chinese Remainder Theorem*) yang dijelaskan pada Teorema 2.3.4 yang terkait dengan masalah ini.

<sup>5</sup> Sekumpulan bilangan-bilangan bulat  $a_1, a_2, \dots, a_n$  dikatakan saling relatif prima satu sama lain apabila  $\gcd(a_i, a_j) = 1$  untuk setiap  $i \neq j$ .

---

**Teorema 2.3.4 (Teorema Sisa Tiongkok (Chinese Remainder Theorem), Teorema 16.43 pada [10])**  
Misalkan  $n_1, n_2, \dots, n_k$  adalah  $k$  bilangan bulat positif yang memenuhi  $\gcd(n_i, n_j) = 1$  untuk setiap  $1 \leq i < j \leq n$ , maka untuk setiap bilangan bulat  $a_1, a_2, \dots, a_n$  sistem kongruensi linier

$$\begin{aligned}x &\equiv a_1 \pmod{n_1} \\x &\equiv a_2 \pmod{n_2} \\\vdots \\x &\equiv a_k \pmod{n_k}\end{aligned}$$

memiliki solusi tunggal dalam modulo  $n_1 n_2 \cdots n_k$ .

Misalkan kita ingin mencari nilai  $x$  yang memenuhi sistem kongruensi linier berikut:

$$\begin{aligned}x &\equiv 3 \pmod{4} \\x &\equiv 4 \pmod{5} \\x &\equiv 1 \pmod{9} \\x &\equiv 5 \pmod{7}\end{aligned}\tag{2.8}$$

Kita dapat menyelesaikan sistem kongruensi (2.8) menggunakan SageMath dengan perintah berikut:

```
crt([3, 4, 1, 5], [4, 5, 9, 7])
```

SageMath akan memberikan keluaran 19 yang berarti solusi dari sistem kongruensi linier (2.8) adalah  $x \equiv 19 \pmod{1260}$ . Dalam hal ini 1260 diperoleh dari  $4 \cdot 5 \cdot 9 \cdot 7$ .

### 2.3.3 Prime Field $\mathbb{Z}_p$ atau $\mathbb{F}_p$ pada SageMath

Sebuah *field* merupakan suatu ring komutatif dengan elemen satuan yang setiap elemen tak nolnya merupakan unit (memiliki invers perkalian). Dari teori bilangan dan aljabar kita mengetahui bahwa ring  $\mathbb{Z}_p$  untuk setiap bilangan prima  $p > 1$  merupakan suatu *field*. Lebih jauh, *field* seperti ini dinamakan sebagai *prime field*. Ada beberapa notasi matematis untuk *prime field* dengan  $p$  elemen, yaitu  $\mathbb{Z}_p$ ,  $\mathbb{F}_p$ , dan  $GF(p)$ . Konstruksi dari *field* ini dapat dilakukan dengan beberapa cara berikut pada SageMath:

```
F1 = Integers(7); print(F1)
F2 = GF(7); print(F2)
print(F1.is_field())
print(F2.is_field())
print(F1 == F2)
```

*Method .is\_field()* digunakan untuk memeriksa apakah suatu struktur matematika merupakan *field* atau bukan. SageMath memberikan keluaran berikut dari skrip sebelumnya:

```
Ring of integers modulo 7
Finite Field of size 7
True
True
False
```

Perhatikan bahwa meskipun  $\mathbb{Z}_7$  dan  $GF(7)$  secara aljabar keduanya isomorfik (keduanya merupakan *field*), namun SageMath menganggap keduanya berbeda. Bila kita mengetikkan perintah berikut pada SageMath:

---

```

A = [x for x in F1]; print(A)
B = [x for x in F2]; print(B)
print(A == B)

```

maka kita memperoleh keluaran berikut:

```

[0, 1, 2, 3, 4, 5, 6]
[0, 1, 2, 3, 4, 5, 6]
True

```

Ini berarti meskipun secara struktur SageMath menganggap  $\mathbb{Z}_7$  dan  $GF(7)$  berbeda, elemen-elemen pembentuk keduanya adalah sama. Dari aljabar kita juga mengetahui bahwa sebuah *field hingga (finite field)* haruslah memiliki elemen sebanyak  $p^k$  untuk suatu bilangan prima  $p > 1$  dan bilangan bulat positif  $k$ . Akibatnya perintah  $GF(n)$  pada SageMath hanya dapat dijalankan apabila  $n = p^k$  untuk suatu bilangan prima  $p > 1$  dan bilangan bulat  $k > 1$ . Kajian mengenai *field  $GF(p^k)$*  akan dibahas pada Subtopik 2.5.

**Latihan 2.3.5** Gunakan SageMath untuk menjawab pertanyaan-pertanyaan berikut:

1. Gunakan SageMath untuk memeriksa apakah 11 memiliki invers perkalian pada  $\mathbb{Z}_{1001}$ . Dengan perkataan lain, apakah ada  $11^{-1} \in \mathbb{Z}_{1001}$  yang memenuhi  $11 \cdot 11^{-1} = 11^{-1} \cdot 11 = 1$  pada  $\mathbb{Z}_{1001}$ . Berikan analisis singkat terkait keluaran yang diberikan SageMath.
2. Gunakan SageMath untuk mencari semua nilai  $x \in \mathbb{Z}_{1001}$  yang memenuhi  $11x = 968$ . Apa kaitan dari solusi persamaan ini dengan invers dari 11 pada  $\mathbb{Z}_{1001}$ ?
3. Gunakan SageMath untuk membantu pencarian nilai  $x$  dengan  $0 \leq x \leq 168$  yang memenuhi

$$\begin{aligned} 4x &\equiv 2 \pmod{6} \\ 5x &\equiv 3 \pmod{7} \\ 7x &\equiv 5 \pmod{8} \end{aligned}$$

## 2.4 Polinom atas Ring $\mathbb{Z}_m$ dan Polinom atas Field $\mathbb{F}_p$ pada SageMath

Misalkan  $R$  adalah sebuah ring. Kita dapat membentuk suatu polinom dengan koefisien-koefisien yang diambil dari  $R$ . Polinom dengan koefisien atas  $R$  dalam variabel  $x$  dengan derajat (*degree*)  $n$  untuk suatu bilangan bulat  $n \geq 0$  secara umum berbentuk

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n,$$

dengan  $a_0, a_1, a_2, \dots, a_n \in R$ . Derajat dari suatu polinom  $f(x)$  selanjutnya dinotasikan dengan  $\deg(f(x))$ . Himpunan seluruh polinom dengan koefisien-koefisien yang diambil dari  $R$  ini selanjutnya dinotasikan dengan  $R[x]$ . Kita dapat membuktikan dengan mudah bahwa  $(R[x], +, \cdot)$  membentuk suatu ring (di sini  $+$  dan  $\cdot$  secara berurutan adalah operasi penjumlahan dan perkalian pada  $R$ ). Ring  $R[x]$  selanjutnya juga dinamakan sebagai ring polinom univariat dalam  $x$  atas  $R$ . Penjumlahan dan perkalian polinom pada  $R[x]$  dilakukan secara natural sebagaimana penjumlahan dan perkalian polinom pada umumnya. Misalkan kita memiliki polinom  $f_1(x) = (1 + 3x)^2$  dan  $f_2(x) = (3 + x)^2$  pada  $\mathbb{Z}[x]$ , kita dapat melakukan komputasi  $f_1(x) + f_2(x)$ ,  $f_1(x) - f_2(x)$ , dan  $f_1(x) \cdot f_2(x)$  dengan skrip berikut:

---

```
Z.<x> = PolynomialRing(Integer()); print(Z)
f1 = (1+3*x)^2; print(f1)
f2 = (3+x)^3; print(f2)
print(f1+f2)
print(f1-f2)
print(f1*f2)
```

Keluaran dari SageMath adalah:

```
Univariate Polynomial Ring in x over Integer Ring
9*x^2 + 6*x + 1
x^3 + 9*x^2 + 27*x + 27
x^3 + 18*x^2 + 33*x + 28
-x^3 - 21*x - 26
9*x^5 + 87*x^4 + 298*x^3 + 414*x^2 + 189*x + 27
```

Untuk mengonversi keluaran SageMath ke dalam format LaTeX kita dapat menambahkan fungsi `latex(...)` pada ekspresi yang akan diubah ke dalam format LaTeX. Misalnya untuk mengubah ekspresi  $f_1 = 9x^2 + 6x + 1$  ke dalam format LaTeX kita dapat menggunakan perintah `print(latex(f1))`. Hasil dari perintah ini adalah  $9x^2 + 6x + 1$ .

Perhatikan bahwa SageMath secara otomatis menuliskan polinom dalam bentuk  $a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0$ . Dalam komputasi yang dilakukan SageMath kita memperoleh  $f_1(x) = 9x^2 + 6x + 1$ ,  $f_2(x) = x^3 + 9x^2 + 27x + 27$ ,  $f_1(x) + f_2(x) = x^3 + 18x^2 + 33x + 28$ ,  $f_1(x) - f_2(x) = -x^3 - 21x - 26$ , dan  $f_1(x) \cdot f_2(x) = 9x^5 + 87x^4 + 298x^3 + 414x^2 + 189x + 27$ .

Kalkulasi juga dapat dilakukan untuk polinom yang koefisien-koefisiennya diambil dari  $\mathbb{Z}_m$  maupun  $GF(p)$ . Misalkan kita memiliki  $g_1(x) = x^2 + 3 \in GF(7)[x]$  dan  $g_2(x) = x^3 + x + 4 \in GF(7)[x]$ , maka kita dapat menghitung  $g_1(x) + g_2(x)$ ,  $g_1(x) - g_2(x)$ ,  $g_1(x) \cdot g_2(x)$ , dan  $(g_1(x))^4$  dengan skrip berikut:

```
R.<x> = PolynomialRing(GF(7)); print(R)
g1 = x^2 + 3; print(g1)
g2 = x^3 + x + 4; print(g2)
print(g1+g2)
print(g1-g2)
print(g1*g2)
print(g1^4)
```

SageMath memberikan keluaran berikut:

```
Univariate Polynomial Ring in x over Finite Field of size 7
x^2 + 3
x^3 + x + 4
x^3 + x^2 + x
6*x^3 + x^2 + 6*x + 6
x^5 + 4*x^3 + 4*x^2 + 3*x + 5
x^8 + 5*x^6 + 5*x^4 + 3*x^2 + 4
```

Perhatikan bahwa seluruh kalkulasi penjumlahan dan perkalian dilakukan di  $GF(7)$  (dalam modulo 7). SageMath merepresentasikan polinom pada  $\mathbb{Z}_m$  maupun  $GF(p)$  dengan menghindari koefisien “negatif”. Sebagai contoh polinom  $f(x) = x^2 - 4x - 3 \in GF(7)[x]$  direpresentasikan sebagai  $f(x) = x^2 + 3x + 4$ , hal ini karena kita memiliki  $-4 = 3$  dan  $-3 = 4$  pada  $GF(7)$ . Perhatikan skrip berikut:

```
R.<x> = PolynomialRing(GF(7)); print(R)
f= x^2-4*x-3; print(f)
```

Keluaran dari skrip ini adalah:

```
Univariate Polynomial Ring in x over Finite Field of size 7
x^2 + 3*x + 4
```

## 2.4.1 Teorema Pembagian pada Ring Polinom

Selain operasi penjumlahan, pengurangan, dan perkalian, kita juga dapat melakukan operasi pembagian, mod, dan div pada polinom yang koefisien-koefisiennya merupakan anggota suatu daerah integral. Lebih lanjut, pada ring polinom  $\mathbb{F}_p[x]$  yang koefisien-koefisiennya diambil dari *prime field*  $\mathbb{F}_p$  operasi mod dan div dilakukan berdasarkan Teorema 2.4.1 yang merupakan perumuman dari Teorema 2.1.1.

**Teorema 2.4.1** Misalkan  $f(x)$  dan  $g(x)$  adalah dua polinom pada  $\mathbb{F}_p[x]$  dengan  $g(x) \neq 0$ , maka terdapat polinom  $q(x)$  dan  $r(x)$  di  $\mathbb{F}_p[x]$  yang memenuhi

$$f(x) = g(x)q(x) + r(x),$$

dengan  $r(x) = 0$  atau  $\deg(r(x)) < \deg(g(x))$ . Lebih jauh, kita menuliskan  $q(x) = f(x) \operatorname{div} g(x)$  dan  $r(x) = f(x) \operatorname{mod} g(x)$ .

Misalkan  $f(x), g(x) \in \mathbb{F}_p[x]$ , operasi  $f(x)/g(x)$ ,  $f(x) \operatorname{div} g(x)$ ,  $f(x) \operatorname{mod} g(x)$  pada SageMath secara berturut-turut dilakukan dengan operator  $/$ ,  $//$ , dan  $\%$ . Misalkan kita memiliki  $f(x) = x^4 + 6x^3 \in \mathbb{F}_7[x]$  dan  $g(x) = x^2 + 5x + 1 \in \mathbb{F}_7[x]$ , kita dapat melakukan kalkulasi  $f(x)/g(x)$ ,  $f(x) \operatorname{div} g(x)$ , dan  $f(x) \operatorname{mod} g(x)$  menggunakan skrip berikut:

```
R.<x> = PolynomialRing(GF(7)); print(R)
f = x^4 + 6*x^3; print(f)
g = x^2 + 5*x + 1; print(g)
print(f/g)
print(f//g)
print(f % g)
```

Hasil dari skrip ini pada SageMath adalah:

```
Univariate Polynomial Ring in x over Finite Field of size 7
x^4 + 6*x^3
x^2 + 5*x + 1
x^3/(x + 6)
x^2 + x + 1
x + 6
```

Dari keluaran ini kita memiliki  $f(x)/g(x) = \frac{x^3}{x+6}$ ,  $f(x) \operatorname{div} g(x) = x^2 + x + 1$ , dan  $f(x) \operatorname{mod} g(x) = x + 6$ . Perlu diingat bahwa  $f(x)/g(x)$  belum tentu merupakan elemen dari  $\mathbb{F}_7[x]$ . Pada SageMath jika kita memiliki suatu ekspresi matematis  $a \in A$ , kita dapat mengetahui struktur himpunan  $A$  dengan sintaks `parent(...)`. Misalnya dari skrip sebelumnya kita dapat membuat skrip lanjutan berikut:

```
print(parent(f/g))
print(parent(f//g))
print(parent(f%g))
```

Keluaran dari skrip SageMath adalah:

```
Fraction Field of Univariate Polynomial Ring in x over Finite Field of size 7
Univariate Polynomial Ring in x over Finite Field of size 7
Univariate Polynomial Ring in x over Finite Field of size 7
```

Teori mengenai *field of fractions* atau *fraction field* (terkadang juga disebut lapangan hasil bagi) dapat dilihat pada [10, Bab 18]. Sebagaimana telah dijelaskan pada Subtopik 2.1.1 mengenai `method .divides(...)` untuk bilangan bulat, kita juga dapat menggunakan `method` tersebut untuk polinom pada  $\mathbb{F}_p[x]$ . Definisi keterbagian pada  $\mathbb{F}_p[x]$  berikut merupakan perumuman dari Definisi 2.1.6.

**Definisi 2.4.2** Misalkan  $f(x), g(x) \in \mathbb{F}_p[x]$ , maka  $f(x)$  dikatakan *habis membagi*  $g(x)$ , ditulis dengan  $f(x)|g(x)$ , apabila terdapat  $h(x) \in \mathbb{F}_p[x]$  dengan sifat  $f(x) \cdot h(x) = g(x)$ . Notasi

---

$f(x) \nmid g(x)$  menyatakan bahwa  $f(x)$  tidak habis membagi  $g(x)$ .

Dalam aljabar, apabila  $f(x), g(x) \in \mathbb{R}[x]$  dan  $f(x)|g(x)$ , maka kita juga mengatakan  $f(x)$  sebagai faktor dari  $g(x)$ . Kita dapat memeriksa hubungan keterbagian untuk polinom pada  $\mathbb{F}_p[x]$  menggunakan SageMath. Misalkan kita memiliki  $f(x) = x^3 + 2x^2 + 2x + 2 \in \mathbb{F}_7[x]$  dan  $g(x) = x^2 + 3x + 5 \in \mathbb{F}_7[x]$ , kita dapat memeriksa apakah  $f(x)|g(x)$  maupun  $g(x)|f(x)$  dipenuhi menggunakan skrip berikut:

```
F.<x> = PolynomialRing(GF(7)); print(F)
f = x^3 + 2*x^2 + 2*x + 2; print(f)
g = x^2 + 3*x + 5; print(g)
print(f.divides(g))
print(g.divides(f))
print(g/f)
print(g//f)
print(g%f)
print(f/g)
print(f//g)
print(f%g)
```

Hasil eksekusi dari SageMath memberikan keluaran berikut:

```
Univariate Polynomial Ring in x over Finite Field of size 7
x^3 + 2*x^2 + 2*x + 2
x^2 + 3*x + 5
False
True
1/(x + 6)
0
x^2 + 3*x + 5
x + 6
x + 6
0
```

Perhatikan bahwa  $f(x) \nmid g(x)$  namun  $g(x)|f(x)$ . Kita memiliki  $g(x) \text{ div } f(x) = 0$  dan  $g(x) \text{ mod } f(x) = g(x)$ . Hasil komputasi SageMath juga memberikan  $f(x)/g(x) = f(x) \text{ div } g(x) = x+6$  dan  $f(x) \text{ mod } g(x) = 0$ . Pada kondisi  $g(x)|f(x)$  kita mengatakan bahwa  $g(x)$  adalah faktor dari  $f(x)$ . Kajian mengenai faktorisasi dari polinom akan dibahas pada Subtopik 2.4.3.

**Latihan 2.4.3** Pada SageMath derajat dari suatu polinom  $f(x) \in \mathbb{F}_p[x]$  dapat diketahui dengan method `.degree()`. Misalnya untuk mengetahui derajat dari polinom  $f(x) = 3x^3 + 4x + 1 \in \mathbb{F}_5[x]$  kita dapat menggunakan perintah `(3*x^3+4*x+1).degree()`. SageMath akan memberikan keluaran 3.<sup>6</sup> Pada soal ini Anda diminta untuk mencari semua polinom berderajat 1 dan 2 pada  $\mathbb{F}_7[x]$  yang habis membagi  $5x^3 + x^2 + x + 3 \in \mathbb{F}_7[x]$ .

## 2.4.2 gcd dan lcm pada Ring Polinom

Ketika kita meninjau ring polinom  $\mathbb{F}_p[x]$  maka kita dapat menghitung gcd dan lcm dari sembarang dua polinom  $f(x), g(x) \in \mathbb{F}_p[x]$ . Konsep gcd maupun lcm untuk polinom pada dasarnya adalah perumuman dari konsep gcd dan lcm untuk bilangan bulat yang dibahas pada Subtopik 2.1.2. Sebuah polinom  $d(x) \in \mathbb{F}_p[x]$  dikatakan sebagai faktor persekutuan terbesar (gcd) dari polinom  $f(x), g(x) \in \mathbb{F}_p[x]$  apabila  $d(x)$  adalah polinom dengan derajat terbesar dengan sifat

<sup>6</sup>SageMath mendefinisikan derajat dari setiap konstanta, atau polinom dengan bentuk  $a_0$  dengan  $a_0 \in R$ , sebagai polinom berderajat -1.

$d(x)|f(x)$  dan  $d(x)|g(x)$ . Selanjutnya sebuah polinom  $c(x) \in \mathbb{F}_p[x]$  dikatakan sebagai kelipatan persekutuan terkecil ( $\text{lcm}$ ) dari polinom  $f(x), g(x) \in \mathbb{F}_p[x]$  apabila  $c(x)$  adalah polinom dengan derajat terkecil dengan sifat  $f(x)|c(x)$  dan  $g(x)|c(x)$ . SageMath juga mendefinisikan  $\text{gcd}(0, 0) = 0$  dan  $\text{lcm}(0, 0) = 0$  sebagaimana dilakukan pada bilangan bulat.

Misalkan kita memiliki polinom  $f(x) = 5x^3 + x^2 + x + 3 \in \mathbb{F}_7[x]$  dan  $g(x) = x^4 + 3x^3 + 4x + 5 \in \mathbb{F}_7[x]$ , maka kita dapat menentukan  $\text{gcd}(f(x), g(x))$  dan  $\text{lcm}(f(x), g(x))$  menggunakan skrip berikut:

```
F.<x> = PolynomialRing(GF(7)); print(F)
f = 5*x^3 + x^2 + x + 3; print(f)
g = x^4 + 3*x^3 + 4*x + 5; print(g)
d = gcd(f, g); print(d)
c = lcm(f, g); print(c)
```

SageMath akan memberikan keluaran berikut:

```
Univariate Polynomial Ring in x over Finite Field of size 7
5*x^3 + x^2 + x + 3
x^4 + 3*x^3 + 4*x + 5
x + 3
x^6 + 3*x^5 + 3*x^4 + 6*x^3 + 5*x^2 + 5*x + 1
```

Keluaran ini berarti:

- $\text{gcd}(5x^3 + x^2 + x + 3, x^4 + 3x^3 + 4x + 5) = x + 3$
- $\text{lcm}(5x^3 + x^2 + x + 3, x^4 + 3x^3 + 4x + 5) = x^6 + 3x^5 + 3x^4 + 6x^3 + 5x^2 + 5x + 1$

Identitas Bézout yang dijelaskan pada Teorema 2.1.9 juga dapat diperumum untuk ring polinom  $\mathbb{F}_p[x]$  dalam Teorema 2.4.4.

**Teorema 2.4.4** Misalkan  $f(x), g(x) \in \mathbb{F}_p[x]$ , maka terdapat dua polinom  $s(x), t(x) \in \mathbb{F}_p[x]$  yang memenuhi

$$s(x) \cdot f(x) + t(x) \cdot g(x) = \text{gcd}(f(x), g(x)).$$

Sebagaimana yang telah dijelaskan sebelumnya pada Subtopik 2.1.2 untuk kasus bilangan bulat, kita juga dapat menggunakan fungsi `xgcd(...., ...)` untuk dua polinom  $f(x), g(x) \in \mathbb{F}_p[x]$ . Keluaran dari fungsi ini adalah tripel  $(d(x), s(x), t(x))$  dengan  $s(x)$  dan  $t(x)$  memenuhi  $s(x) \cdot f(x) + t(x) \cdot g(x) = d(x) = \text{gcd}(f(x), g(x))$ . Misalkan kita ingin mencari dua polinom  $s(x), t(x) \in \mathbb{F}_7[x]$  yang memenuhi  $s(x) \cdot f(x) + t(x) \cdot g(x) = \text{gcd}(f(x), g(x))$  untuk  $f(x) = 5x^3 + x^2 + x + 3 \in \mathbb{F}_7[x]$  dan  $g(x) = x^4 + 3x^3 + 4x + 5 \in \mathbb{F}_7[x]$ , maka kita dapat menggunakan skrip berikut:

```
F.<x> = PolynomialRing(GF(7)); print(F)
f = 5*x^3 + x^2 + x + 3; print(f)
g = x^4 + 3*x^3 + 4*x + 5; print(g)
ans = xgcd(f, g); print(ans)
```

SageMath memberikan keluaran berikut:

```
Univariate Polynomial Ring in x over Finite Field of size 7
5*x^3 + x^2 + x + 3
x^4 + 3*x^3 + 4*x + 5
(x + 3, 5*x^2 + 2*x + 6, 3*x + 4)
```

Keluaran ini berarti nilai  $s(x), t(x)$ , dan  $d(x)$  pada persamaan  $s(x) \cdot f(x) + t(x) \cdot g(x) = d(x) = \text{gcd}(f(x), g(x))$  untuk  $f(x) = 5x^3 + x^2 + x + 3 \in \mathbb{F}_7[x]$  dan  $g(x) = x^4 + 3x^3 + 4x + 5 \in \mathbb{F}_7[x]$  adalah  $d(x) = x + 3$ ,  $s(x) = 5x^2 + 2x + 6$ , dan  $t(x) = 3x + 4$ .

**Latihan 2.4.5** Apakah modifikasi fungsi `gcdlist` dan `lcmlist` pada Soal 1 Latihan 2.1.8 juga dapat diterapkan pada ring polinom  $\mathbb{F}_p[x]$ ? Hitunglah nilai-nilai berikut menggunakan modifikasi yang sesuai:

- 
1.  $\gcd(f(x), g(x), h(x))$  dengan  $f(x) = x^3 + 3x^2 + 3x + 1 \in \mathbb{F}_5[x]$ ,  $g(x) = x^2 + 2x + 1 \in \mathbb{F}_5[x]$ , dan  $h(x) = 4x + 4 \in \mathbb{F}_5[x]$
  2.  $\text{lcm}(f(x), g(x), h(x))$  dengan  $f(x) = x^3 + 3x^2 + 3x + 1 \in \mathbb{F}_5[x]$ ,  $g(x) = x^2 + 2x + 1 \in \mathbb{F}_5[x]$ , dan  $h(x) = 4x + 4 \in \mathbb{F}_5[x]$

### 2.4.3 Akar, Polinom Tak Tereduksi, dan Faktorisasi Polinom

Misalkan  $f(x) \in R[x]$  adalah suatu polinom pada ring polinom  $R[x]$  dengan koefisien-koefisien yang diambil dari ring  $R$ . Sebuah elemen  $a \in R$  dikatakan sebagai akar (*root*) atau *zero* dari  $f(x) \in R[x]$  apabila  $f(a) = 0$ . Pada SageMath kita dapat menggunakan *method* `.roots()` untuk mencari semua akar dari polinom yang koefisien-koefisiennya didefinisikan pada ring tertentu. Misalkan kita meninjau polinom-polinom  $f(x) = x^2 + x - 6 \in \mathbb{Z}[x]$ ,  $g(x) = 2x^2 + 5x - 3 \in \mathbb{Z}[x]$ , dan  $h(x) = x^5 + 5x^4 - 5x^3 - 45x^2 + 108 \in \mathbb{Z}[x]$ . Kita dapat memakai skrip berikut untuk mencari akar-akar dari  $f(x)$ ,  $g(x)$ , dan  $h(x)$  pada ring  $\mathbb{Z}$ :

```
Z.<x> = PolynomialRing(Integer()); print(Z)
f = x^2-x-6; print(f)
g = 2*x^2+5*x-3; print(g)
h = x^5+5*x^4-5*x^3-45*x^2+108; print(h)
print(f.roots())
print(g.roots())
print(h.roots())
```

SageMath memberikan keluaran berikut:

```
Univariate Polynomial Ring in x over Integer Ring
x^2 - x - 6
2*x^2 + 5*x - 3
x^5 + 5*x^4 - 5*x^3 - 45*x^2 + 108
[(3, 1), (-2, 1)]
[(-3, 1)]
[(2, 2), (-3, 3)]
```

Keluaran dari *method* `f.roots()` untuk suatu polinom  $f(x) \in R[x]$  berbentuk *list* yang setiap komponennya adalah pasangan  $(a, m_a)$  dengan  $a$  adalah akar dari  $f(x)$  dan  $m_a$  adalah multiplisitas (banyaknya kemunculan) dari  $a$ . Pada contoh sebelumnya kita memiliki:

1. Akar-akar dari  $x^2 - x - 6$  pada  $\mathbb{Z}$  adalah 3 (dengan multiplisitas 1) dan  $-2$  (dengan multiplisitas 1). Perhatikan bahwa  $x^2 - x - 6 = (x - 3)(x + 2)$ .
2. Akar-akar dari  $2x^2 + 5x - 3$  pada  $\mathbb{Z}$  adalah  $-3$  (dengan multiplisitas 1). Perhatikan bahwa  $2x^2 + 5x - 3 = (2x - 1)(x + 3)$  sehingga solusi dari  $2x^2 + 5x - 3 = 0$  adalah  $x = \frac{1}{2}$  dan  $x = -3$ . Namun  $x = \frac{1}{2} \notin \mathbb{Z}$ , sehingga jika kita bekerja pada  $\mathbb{Z}[x]$  akar dari  $2x^2 + 5x - 3$  adalah  $x = -3$  saja.
3. Akar-akar dari  $x^5 + 5x^4 - 5x^3 - 45x^2 + 108$  pada  $\mathbb{Z}$  adalah 2 (dengan multiplisitas 2) dan  $-3$  (dengan multiplisitas 3). Perhatikan bahwa  $x^5 + 5x^4 - 5x^3 - 45x^2 + 108 = (x - 2)^2(x + 3)^3$ .

Pencarian akar juga dapat dilakukan pada polinom pada ring  $\mathbb{F}_p[x]$ . Misalkan kita ingin mencari akar-akar dari  $f(x) = x^2 + 1 \in \mathbb{F}_5[x]$ ,  $g(x) = x^3 + 3x^2 + 3x + 1 \in \mathbb{F}_5[x]$ , dan  $h(x) = x^4 + 1 \in \mathbb{F}_5[x]$ . Kita dapat menggunakan skrip SageMath berikut:

---

```
F5.<x> = PolynomialRing(GF(5)); print(F5)
f = x^2+1; print(f)
g = x^3+3*x^2+3*x+1; print(g)
h = x^4+1; print(h)
print(f.roots())
print(g.roots())
print(h.roots())
```

Keluaran dari skrip ini adalah:

```
Univariate Polynomial Ring in x over Finite Field of size 5
x^2 + 1
x^3 + 3*x^2 + 3*x + 1
x^4 + 1
[(3, 1), (2, 1)]
[(4, 3)]
[]
```

Perhatikan bahwa:

1. Pada  $\mathbb{F}_5[x]$  kita memiliki  $f(x) = x^2 + 1 = x^2 - 4 = (x - 2)(x + 2) = (x + 3)(x + 2)$ , sehingga akar-akar dari  $f(x) = x^2 + 1 \in \mathbb{F}_5[x]$  adalah  $-3 = 2$  dengan multiplisitas 1 dan  $-2 = 3$  dengan multiplisitas 1.
2. Pada  $\mathbb{F}_5[x]$  kita memiliki  $g(x) = x^3 + 3x^2 + 3x + 1 = (x + 1)^3$ , sehingga akar-akar dari  $g(x) = x^3 + 3x^2 + 3x + 1 \in \mathbb{F}_5[x]$  adalah  $-1 = 4$  dengan multiplisitas 3.
3. Pada  $\mathbb{F}_5[x]$  polinom  $h(x) = x^4 + 1 \in \mathbb{F}_5[x]$  tidak memiliki akar. Secara aljabar ini berarti tidak terdapat  $a \in \mathbb{F}_5$  sehingga  $h(a) = a^4 + 1 = 0$ .

Sebuah elemen  $p$  pada ring  $R$  dikatakan tak tereduksi (*irreducible*) apabila  $p$  bukan unit ( $p$  tidak memiliki invers multiplikatif) dan setiap ‘‘faktorisasi’’ dari  $p$  berbentuk  $p = r \cdot s$  dengan salah satu dari  $r$  atau  $s$  adalah unit. Sifat ‘‘tak tereduksi’’ merupakan perumuman dari sifat bilangan prima pada ring  $\mathbb{Z}$ . Sebuah polinom  $p(x) \in \mathbb{F}[x]$  dengan koefisien-koefisien pada *field*  $\mathbb{F}$  dikatakan tak tereduksi bila tidak ada polinom  $r(x), s(x) \in \mathbb{F}[x]$  dengan sifat  $0 < \deg(r(x)), \deg(s(x)) < \deg(p(x))$  yang memenuhi  $p(x) = r(x) \cdot s(x)$ . Pada SageMath kita dapat memeriksa apakah suatu polinom bersifat tereduksi atau tidak menggunakan *method* `.is_irreducible()`. Misalkan  $f(x) = x^2 + 1 \in \mathbb{F}_5[x]$ ,  $g(x) = x^2 + 3 \in \mathbb{F}_5[x]$ , dan  $h(x) = x^4 + 1 \in \mathbb{F}_5[x]$ , kita dapat memeriksa apakah polinom-polinom tersebut tereduksi atau tidak menggunakan skrip berikut:

```
F5.<x> = PolynomialRing(GF(5)); print(F5)
f = x^2+1; print(f)
g = x^2+3; print(g)
h = x^4+1; print(h)
print(f.is_irreducible())
print(g.is_irreducible())
print(h.is_irreducible())
```

Keluaran dari skrip ini adalah:

```
Univariate Polynomial Ring in x over Finite Field of size 5
x^2 + 1
x^2 + 3
x^4 + 1
False
True
False
```

---

Keluaran ini berarti  $f(x) = x^2 + 1 \in \mathbb{F}_5[x]$  dan polinom  $h(x) = x^4 + 1$  adalah polinom yang tereduksi, sedangkan polinom  $g(x) = x^2 + 3 \in \mathbb{F}_5[x]$  adalah polinom yang tak tereduksi. Secara aljabar perhatikan bahwa:

1. Pada  $\mathbb{F}_5[x]$  kita memiliki  $f(x) = x^2 + 1 = (x + 2)(x + 3)$ .
2. Pada  $\mathbb{F}_5[x]$  tidak terdapat polinom  $r(x)$  dan  $s(x)$  dengan  $0 < \deg(r(x)), \deg(s(x)) < \deg(g(x))$  yang memenuhi  $g(x) = x^2 + 3 = r(x) \cdot s(x)$ .
3. Pada  $\mathbb{F}_5[x]$  kita memiliki  $h(x) = x^4 + 1 = (x^2 + 2)(x^2 + 3)$ .

Perhatikan bahwa jika suatu polinom  $p(x) \in \mathbb{F}[x]$  tidak memiliki akar, maka belum tentu polinom tersebut tidak tereduksi. Dari contoh-contoh yang telah dijelaskan, kita melihat bahwa polinom  $h(x) = x^4 + 1 \in \mathbb{F}_5[x]$  tidak memiliki akar, namun  $h(x)$  tereduksi karena dapat ditulis sebagai  $(x^2 + 3)(x^2 + 3)$ . Akan tetapi, jika suatu polinom  $p(x) \in \mathbb{F}[x]$  memiliki akar, maka sudah pasti  $p(x)$  tereduksi pada  $\mathbb{F}[x]$ . Hal ini karena jika  $p(x)$  memiliki akar, maka terdapat  $x = a$  yang memenuhi  $p(a) = 0$ . Dari aljabar ini juga mengakibatkan bahwa  $x - a$  adalah salah satu faktor dari  $p(x)$ .

Berdasarkan sifat tak tereduksi yang telah dijelaskan, kita memiliki Teorema 2.4.6 yang merupakan perumuman dari Teorema Fundamental Aritmetika pada Teorema 2.1.11.

**Teorema 2.4.6 (Teorema Faktorisasi Polinom (Proposisi 2.49 pada [2]))** Misalkan  $\mathbb{F}$  adalah sebuah field, maka setiap polinom  $f(x) \in \mathbb{F}[x]$  dapat dinyatakan sebagai hasil kali polinom-polinom monik<sup>7</sup> tak tereduksi dalam cara yang tunggal. Ini berarti apabila:

$$f(x) = \alpha \cdot p_1(x) \cdot p_2(x) \dots p_m(x) \text{ dan}$$

$$f(x) = \beta \cdot q_1(x) \cdot q_2(x) \dots q_n(x),$$

dengan  $\alpha, \beta \in \mathbb{F}$  dan  $p_1(x), \dots, p_m(x), q_1(x), \dots, q_n(x)$  adalah polinom-polinom monik yang tak tereduksi pada  $\mathbb{F}[x]$ , kita dapat mengatur urutan dari  $q_1(x), \dots, q_n(x)$  dan memperoleh:  $\alpha = \beta$ ,  $m = n$ , dan  $p_i(x) = q_i(x)$  untuk setiap  $1 \leq i \leq m$ .

Faktorisasi untuk polinom pada  $R[x]$  pada SageMath dilakukan menggunakan fungsi `factor(...)` maupun `method .factor()`. Misalkan kita ingin memfaktorkan  $f(x) = x^2 + 1 \in \mathbb{F}_5[x]$ ,  $g(x) = x^2 + 3 \in \mathbb{F}_5[x]$ , dan  $h(x) = x^4 + 1 \in \mathbb{F}_5[x]$ , maka kita dapat menggunakan skrip berikut:

```
F5.<x> = PolynomialRing(GF(5)); print(F5)
f = x^2+1; print(f)
g = x^2+3; print(g)
h = x^4+1; print(h)
print(factor(f))
print(f.factor())
print(factor(g))
print(g.factor())
print(factor(h))
print(h.factor())
```

SageMath memberikan keluaran berikut:

---

<sup>7</sup>Sebuah polinom  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  dikatakan sebagai polinom monik apabila  $a_n = 1$ .

```

Univariate Polynomial Ring in x over Finite Field of size 5
x^2 + 1
x^2 + 3
x^4 + 1
(x + 2) * (x + 3)
(x + 2) * (x + 3)
x^2 + 3
x^2 + 3
(x^2 + 2) * (x^2 + 3)
(x^2 + 2) * (x^2 + 3)

```

Dari penjelasan sebelumnya kita mengetahui bahwa pada  $\mathbb{F}_5[x]$  kita memiliki  $f(x) = (x^2 + 1)$  dapat difaktorkan menjadi  $(x + 2)(x + 3)$ ,  $g(x) = (x^2 + 3)$  adalah polinom tak tereduksi, dan  $h(x) = (x^4 + 1)$  dapat difaktorkan menjadi  $(x^2 + 2)(x^2 + 3)$ .

### Latihan 2.4.7

1. Gunakan SageMath untuk mencari semua akar dari polinom-polinom berikut yang didefinisikan pada  $\mathbb{F}_7[x]$ :
  - (a)  $f(x) = x^3 - 3$
  - (b)  $g(x) = 3x^3 + x^2 + 2x + 1$
  - (c)  $h(x) = x^4 + x^2 + 1$
2. Gunakan SageMath untuk memfaktorkan semua polinom pada Soal 1.

## 2.5 Field $GF(p^k)$ atau $\mathbb{F}_{p^k}$

Pada Subtopik 2.3 kita melihat bahwa  $\mathbb{Z}_p$  adalah sebuah *field* hingga (*finite field*) dengan  $p$  anggota  $0, 1, \dots, p - 1$ . Dalam aljabar kita mengetahui bahwa  $\mathbb{Z}/m\mathbb{Z} \cong \mathbb{Z}_m$ . Kita juga mengetahui bahwa  $\mathbb{Z}/m\mathbb{Z} \cong \mathbb{Z}_m$  adalah sebuah *field* jika dan hanya jika  $m$  adalah bilangan prima. Melalui ide konstruksi yang serupa kita dapat membangun sebuah *field* hingga dengan  $p^k$  anggota dengan meninjau ring kuosien (disebut juga ring hasil bagi atau gelanggang hasil bagi)  $\mathbb{F}_p[x]/(f(x))$  untuk suatu polinom  $f(x) \in \mathbb{F}_p[x]$  yang berderajat  $k$ .

Ring kuosien  $\mathbb{F}_p[x]/(f(x))$  memuat seluruh polinom yang derajatnya tidak lebih dari  $\deg(f(x))$ . Misalkan operasi penjumlahan dan perkalian pada  $\mathbb{F}_p[x]/(f(x))$  secara berturut-turut dinotasikan dengan  $\oplus$  dan  $\odot$ , maka operasi-operasi ini didefinisikan sebagai berikut:

1. Untuk setiap  $p(x), q(x) \in \mathbb{F}_p[x]/(f(x))$  maka  $p(x) \oplus q(x) = (p(x) + q(x)) \text{ mod } f(x)$ , dengan  $p(x) + q(x)$  menyatakan penjumlahan polinom pada  $\mathbb{F}_p[x]$ .
2. Untuk setiap  $p(x), q(x) \in \mathbb{F}_p[x]/(f(x))$  maka  $p(x) \odot q(x) = (p(x) \cdot q(x)) \text{ mod } f(x)$ , dengan  $p(x) \cdot q(x)$  menyatakan perkalian polinom pada  $\mathbb{F}_p[x]$ .

Misalkan  $\deg(f(x)) = k$ , maka  $\mathbb{F}_p[x]/(f(x))$  memuat semua polinom dengan bentuk  $a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \dots + a_1x + a_0$  dengan  $a_0, a_1, \dots, a_{k-1} \in \mathbb{F}_p$ . Akibatnya ring kuosien  $\mathbb{F}_p[x]/(f(x))$  memuat  $p^k$  elemen berbeda. Dari aljabar abstrak kita juga mengetahui Teorema 2.5.1 berikut.

**Teorema 2.5.1 (Akibat 2.54 pada [2], Teorema 40.1 pada [12])** Misalkan  $\mathbb{F}$  adalah sebuah *field* dan  $f(x) \in \mathbb{F}[x]$ , maka ring kuosien  $\mathbb{F}[x]/(f(x))$  adalah sebuah *field* jika dan hanya jika  $f(x)$  tidak tereduksi di  $\mathbb{F}[x]$ .

---

Field  $\mathbb{F}_p[x]/(f(x))$  dengan  $\deg(f(x)) = k$  selanjutnya dinotasikan dengan  $GF(p^k)$  atau  $\mathbb{F}_{p^k}$ . Beberapa referensi menamakan field ini sebagai field kuosien atau lapangan hasil bagi. Berdasarkan teori dalam aljabar abstrak, dengan memperhatikan isomorfisme, hanya terdapat satu field  $GF(p^k)$  untuk setiap bilangan prima  $p > 1$  dan bilangan bulat  $k \geq 1$ .

Hasil matematis yang diperoleh di sini memberikan sebuah keuntungan dalam kriptografi. Alih-alih menggunakan field  $\mathbb{F}_p$  untuk bilangan prima  $p$  yang cukup besar, kita dapat memilih untuk bekerja pada field  $\mathbb{F}_{2^k}$  untuk nilai  $2^k$  yang mendekati nilai  $p$ . Salah satu alasannya karena operasi pada field  $\mathbb{F}_{2^k}$  dapat dilakukan dengan meninjau string biner yang merupakan representasi dari polinom pada  $\mathbb{F}_2[x]/(f(x))$  untuk suatu polinom tak tereduksi  $f(x)$  dengan  $\deg(f(x)) = k$ . Dalam kasus ini, setiap elemen  $p(x) \in \mathbb{F}_2[x]/(f(x))$  dituliskan sebagai polinom  $a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \dots + a_1x + a_0$  dengan  $a_0, a_1, \dots, a_{k-1} \in \{0, 1\}$ . Pada kondisi ini elemen dari  $\mathbb{F}_{2^k}$  cukup ditulis sebagai  $(a_{k-1}a_{k-2} \dots a_1a_0)$  dengan  $a_0, a_1, \dots, a_{k-1} \in \{0, 1\}$ . Representasi seperti ini mudah diterapkan dalam komputer, terutama untuk bahasa tingkat rendah.

Contoh kongkret dari konstruksi suatu field hingga  $\mathbb{F}_{p^k}$  dijelaskan sebagai berikut. Misalkan kita akan mengonstruksi  $\mathbb{F}_4 = \mathbb{F}_{2^2}$ . Pertama kita perlu mencari sebuah polinom tak tereduksi di  $\mathbb{F}_2[x]$  berderajat 2. Perhatikan bahwa ada empat polinom berderajat 2 pada  $\mathbb{F}_2[x]$ , yaitu:  $f_1(x) = x^2$ ,  $f_2(x) = x^2 + 1$ ,  $f_3(x) = x^2 + x$ , dan  $f_4(x) = x^2 + x + 1$ . Kita dapat memeriksa dengan mudah bahwa satu-satunya polinom yang tak tereduksi di antara polinom-polinom ini adalah  $f_4(x) = x^2 + x + 1$ .<sup>8</sup> Untuk selanjutnya kita akan menuliskan  $f_4(x)$  sebagai  $f(x)$ .

Selanjutnya elemen-elemen pada  $\mathbb{F}_{2^2}$  dapat direpresentasikan sebagai polinom pada  $\mathbb{F}_2[x]$  yang derajatnya tidak lebih dari 1. Elemen-elemen tersebut adalah 0, 1,  $x$ , dan  $x + 1$ . Operasi penjumlahan dan perkalian pada  $\mathbb{F}_{2^2}$  pada dasarnya adalah operasi penjumlahan dan perkalian polinom pada  $\mathbb{F}_2[x]$  modulo  $x^2 + x + 1$ . Sebagai contoh  $x \oplus (x + 1) = (x + x + 1) \bmod(x^2 + x + 1) = 1$  dan  $x \odot (x + 1) = (x^2 + x) \bmod(x^2 + x + 1) = 1$ . Kita dapat membangun tabel penjumlahan pada  $\mathbb{F}_4$  dan tabel perkalian pada  $\mathbb{F}_4$  sebagai berikut:

| $\oplus$ | 0       | 1       | $x$     | $x + 1$ | $\odot$ | 0 | 1       | $x$     | $x + 1$ |
|----------|---------|---------|---------|---------|---------|---|---------|---------|---------|
| 0        | 0       | 1       | $x$     | $x + 1$ | 0       | 0 | 0       | 0       | 0       |
| 1        | 1       | 0       | $x + 1$ | $x$     | 1       | 0 | 1       | $x$     | $x + 1$ |
| $x$      | $x$     | $x + 1$ | 0       | 1       | $x$     | 0 | $x$     | $x + 1$ | 1       |
| $x + 1$  | $x + 1$ | $x$     | 1       | 0       | $x + 1$ | 0 | $x + 1$ | 1       | $x$     |

Kemudian jika  $a^n = \underbrace{a \odot a \cdots \odot a}_{n \text{ suku}}$  maka kita memiliki  $x^1 = x$ ,  $x^2 = x + 1$ ,  $x^3 = 1$ . Ini

berarti himpunan semua elemen tak nol pada  $\mathbb{F}_{2^2}$  membentuk grup siklis dengan pembangkit  $x$ . Dari teori pada aljabar kita memiliki Teorema 2.5.2 untuk field hingga.

**Teorema 2.5.2 (Teorema 2.62 pada [2])** Misalkan  $\mathbb{F}_q$  adalah sebuah field hingga dengan  $q = p^k$  anggota (untuk suatu bilangan prima  $p > 1$  dan bilangan bulat  $k \geq 1$ ). Misalkan kita menuliskan  $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$ , dengan perkataan lain  $\mathbb{F}_q^*$  memuat semua elemen tak nol pada  $\mathbb{F}_q$ . Dalam hal ini pasti terdapat elemen  $g \in \mathbb{F}_q^*$  dengan sifat

$$\mathbb{F}_q^* = \{1, g, g^2, \dots, g^{q-2}\}.$$

Selanjutnya elemen  $g$  dikatakan sebagai pembangkit (generator) dari  $\mathbb{F}_q^*$ .

Pada  $\mathbb{F}_{2^2} = \mathbb{F}_2[x]/(x^2 + x + 1)$  yang telah dijelaskan sebelumnya, kita melihat bahwa  $x$  adalah pembangkit dari  $\mathbb{F}_{2^2}^*$ . Pada SageMath kita dapat mendefinisikan field hingga  $GF(q)$  dengan  $q = p^k$  untuk suatu bilangan prima  $p > 1$  dan bilangan bulat  $k \geq 1$  menggunakan

<sup>8</sup>Sebenarnya ada metode konstruktif untuk mencari semua polinom tak tereduksi dengan derajat tertentu pada  $\mathbb{F}_2[x]$ , namun hal ini tidak mudah.

fungsi  $\text{GF}(q)$  atau  $\text{GF}(q, x)$ . Ketika kita menggunakan fungsi  $\text{GF}(q, x)$  maka kita secara eksplisit menggunakan variabel  $x$  sebagai pembangkit dari  $\mathbb{F}_q^*$ . Perhatikan skrip berikut:

```
F = GF(4); print(F)
G = GF(4,x); print(G)
print( F == G)
```

Keluaran dari SageMath adalah:

```
Finite Field in z2 of size 2^2
Finite Field in x of size 2^2
False
```

Kita melihat bahwa pada SageMath  $\text{GF}(4)$  dan  $\text{GF}(4, x)$  diperlakukan sebagai struktur yang berbeda meskipun keduanya isomorfik. Jika kita tidak mendefinisikan pembangkit dari *field* hingga  $GF(p^k)$  yang kita tinjau (untuk kasus  $k > 1$ ) maka SageMath akan menggunakan variabel  $z_k$  sebagai pembangkit sebagaimana pada contoh sebelumnya. Kita dapat mengonstruksi tabel penjumlahan dan tabel perkalian pada  $GF(4)$  dengan pembangkit  $x$  menggunakan perintah-perintah berikut pada SageMath:

```
F4 = GF(4,x); print(F4)
print(F4.addition_table(names='elements'))
print(F4.multiplication_table(names='elements'))
```

Keluaran dari skrip ini adalah:

| Finite Field in x of size 2^2 |         |         |         |         |
|-------------------------------|---------|---------|---------|---------|
|                               | 0       | 1       | $x$     | $x + 1$ |
| +                             | 0       | 1       | $x$     | $x + 1$ |
| 0                             | 0       | 1       | $x$     | $x + 1$ |
| 1                             | 1       | 0       | $x + 1$ | $x$     |
| $x$                           | $x$     | $x + 1$ | 0       | 1       |
| $x + 1$                       | $x + 1$ | $x$     | 1       | 0       |

| *       |   |         |         |         |
|---------|---|---------|---------|---------|
|         | 0 | 1       | $x$     | $x + 1$ |
| *       | 0 | 1       | $x$     | $x + 1$ |
| 0       | 0 | 0       | 0       | 0       |
| 1       | 0 | 1       | $x$     | $x + 1$ |
| $x$     | 0 | $x$     | $x + 1$ | 1       |
| $x + 1$ | 0 | $x + 1$ | 1       | $x$     |

Pada  $\mathbb{F}_4$  kita dapat merepresentasikan elemen 0 sebagai (00), elemen 1 sebagai (01), elemen  $x$  sebagai (10), dan elemen  $x + 1$  sebagai (11).

**Latihan 2.5.3** Gunakan SageMath untuk mendefinisikan  $GF(8)$  dengan pembangkit  $x$ , kemudian buatlah tabel penjumlahan dan perkalian pada  $GF(8)$ .

## 2.6 Vektor dan Matriks atas Ring dan Field pada SageMath

Kita mengenal vektor dan matriks dalam kajian aljabar linier elementer. Salah satu referensi lebih jauh mengenai hal ini dapat dilihat pada [15]. Pada SageMath vektor maupun matriks dapat didefinisikan atas ring atau *field* tertentu.

### 2.6.1 Vektor, Modul, dan Ruang Vektor

Secara matematis, sebuah vektor atas suatu ring (atau *field*)  $R$  adalah sebuah  $n$ -tupel berbentuk  $(a_1, a_2, \dots, a_n)$  dengan  $a_1, a_2, \dots, a_n \in R$  untuk suatu bilangan bulat  $n$  yang merupakan

---

ukuran (atau dimensi) dari modul (atau ruang vektor) yang ditinjau.<sup>9</sup> Pada SageMath vektor direpresentasikan dalam format yang serupa dengan *list*. Kita dapat mendefinisikan vektor  $(a_1, a_2, \dots, a_n)$  menggunakan fungsi `vector([a_1, a_2, ..., a_n])`. Perhatikan contoh berikut:

```
v = vector([0,-1,0,-2])
w = vector([-1.0,-2.0,3,4])
print(v)
print(w)
print(parent(v))
print(parent(w))
print(type(v))
print(type(w))
```

SageMath memberikan keluaran berikut:

```
(0, -1, 0, -2)
(-1.00000000000000, -2.00000000000000, 3.00000000000000, 4.00000000000000)
Ambient free module of rank 4 over the principal ideal domain Integer Ring
Vector space of dimension 4 over Real Field with 53 bits of precision
<class 'sage.modules.vector_integer_dense.Vector_integer_dense'>
<class 'sage.modules.free_module_element.FreeModuleElement_generic_dense'>
```

Perhatikan bahwa secara *default*  $(0, -1, 0, -2)$  diperlakukan sebagai vektor pada modul  $\mathbb{Z}^4$  sedangkan  $(-1.0, -2.0, 3, 4)$  diperlakukan sebagai vektor pada  $\mathbb{R}^4$ . Secara struktur data vektor dan *list* pada SageMath memiliki kemiripan. Pada SageMath, indeks vektor dimulai dari 0 sebagaimana indeks untuk *list*. Tinjau contoh berikut:

```
v = vector([1,2,-3,4])
print(v[0])
print(v[1])
print(v[2])
print(v[3])
print(len(v))
```

SageMath memberikan keluaran:

```
1
2
-3
4
4
```

Panjang (*length*) dari suatu vektor adalah banyaknya komponen pada vektor tersebut. Vektor atas suatu modul atau *field*  $R$  tertentu dengan panjang  $n$  dapat didefinisikan menggunakan sintaks `vector(R, [a_1, a_2, ..., a_n])`. Operasi-operasi aritmetika standar seperti perkalian skalar dan penjumlahan dapat didefinisikan secara natural. Misalkan kita memiliki vektor  $\mathbf{u} = (1, 4, 1, 4) \in \mathbb{F}_5^4$  dan  $\mathbf{v} = (2, 3, 3, 2) \in \mathbb{F}_5^4$ , kita dapat menghitung  $3\mathbf{u} - 4\mathbf{v}$  menggunakan skrip berikut:

```
u = vector(GF(5), [1,4,1,4])
v = vector(GF(5), [2,3,3,2])
print(u)
print(parent(u))
print(v)
print(parent(v))
print(3*u - 4*v)
print(parent(3*u-4*v))
```

SageMath memberikan keluaran:

---

<sup>9</sup>Dalam aljabar, modul atas ring merupakan generalisasi dari ruang vektor atas *field*.

---

```
(1, 4, 1, 4)
Vector space of dimension 4 over Finite Field of size 5
(2, 3, 3, 2)
Vector space of dimension 4 over Finite Field of size 5
(0, 0, 1, 4)
Vector space of dimension 4 over Finite Field of size 5
```

Perhatikan bahwa operasi aritmetika dilakukan berdasarkan aturan pada  $\mathbb{F}_5$ . Selain perkalian skalar dan penjumlahan, kita juga dapat melakukan operasi hasil kali titik (*dot product*) antara dua vektor pada modul atau ruang vektor yang sama. Ingat kembali bahwa hasil kali titik dari dua vektor  $\mathbf{u} = (u_1, u_2, \dots, u_n)$  dan  $\mathbf{v} = (v_1, v_2, \dots, v_n)$  pada modul atau ruang vektor yang sama dinotasikan dengan  $\mathbf{u} \cdot \mathbf{v}$  dan didefinisikan sebagai

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^n u_i v_i = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n.$$

Operasi perkalian dan penjumlahan pada definisi hasil kali titik dilakukan pada ring atau *field* yang ditinjau. Operasi ini dilakukan dengan *method* `.dot_product`. Sebagai contoh kita dapat menghitung hasil kali titik antara  $\mathbf{u} = (0, 2, 1, 3) \in \mathbb{Z}_4^4$  dan  $\mathbf{v} = (1, 1, 3, 3) \in \mathbb{Z}_4^4$  menggunakan skrip berikut:

```
u = vector(Integers(4), [0, 2, 1, 3])
print(u)
v = vector(Integers(4), [1, 1, 3, 3])
print(v)
print(u.dot_product(v))
```

Keluaran dari skrip ini adalah:

```
(0, 2, 1, 3)
(1, 1, 3, 3)
2
```

Perhatikan bahwa pada  $\mathbb{Z}_4^4$  kita memiliki  $(0, 2, 1, 3) \cdot (1, 1, 3, 3) = 0 + 2 + 3 + 1 = 2$ . Pada SageMath pendefinisian sebuah modul atau ruang vektor atas suatu ring atau *field*  $R$  dengan ukuran atau dimensi  $n$  dapat dilakukan menggunakan sintaks  $R^n$ . Selanjutnya untuk memeriksa apakah sebuah vektor  $\mathbf{v}$  merupakan anggota modul atau ruang vektor  $V$  kita dapat menggunakan sintaks  $\mathbf{v} \text{ in } V$ . Perhatikan skrip berikut:

```
U = Integers(4)^4; print(U)
F4.<x> = GF(4)
V = F4^4; print(V)
u = vector([1, 2, 3, 4]); print(u)
print(u in U)
v = vector([x, x+1, 1, 1]); print(v)
print(v in V)
```

Pada skrip ini kita mendefinisikan  $U = \mathbb{Z}_4^4$  dan  $V = \mathbb{F}_4^4$ . Penulisan  $F4.<x> = GF(4)$  penting agar SageMath mengenali variabel yang digunakan pada representasi  $\mathbb{F}_4$ . Hasil SageMath menunjukkan bahwa  $(1, 2, 3, 4) \in \mathbb{Z}_4^4$  dan  $(x, x + 1, 1, 1) \in \mathbb{F}_4^4$  sebagaimana dijelaskan pada keluaran berikut:

```
Ambient free module of rank 4 over Ring of integers modulo 4
Vector space of dimension 4 over Finite Field in x of size 2^2
(1, 2, 3, 4)
True
(x, x + 1, 1, 1)
True
```

Pada SageMath meskipun vektor ditampilkan dalam bentuk tupel, vektor dengan  $n$  kompo-

---

nen direpresentasikan dalam bentuk matriks kolom ketika dikalikan dengan matriks dari sebelah kanan dan direpresentasikan dalam bentuk matriks baris ketika dikalikan dengan matriks dari sebelah kiri. Hal ini akan dibahas lebih lanjut pada Subtopik 2.6.2.

**Latihan 2.6.1** Vektor  $u$  dan  $v$  pada modul atau ruang vektor  $U$  dikatakan saling ortogonal apabila  $u \cdot v = 0$ . Carilah semua vektor pada  $\mathbb{Z}_8^5$  yang ortogonal dengan vektor  $u = (1, 2, 5, 6, 7)$ .

## 2.6.2 Matriks atas Ring dan Field Pada SageMath

Misalkan kita memiliki sebuah matriks  $A$  yang berukuran  $m \times n$  sebagai berikut:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \ddots & \dots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix} \quad (2.9)$$

Pada SageMath kita dapat mendefinisikan matriks tersebut dengan sintaks

```
A = matrix([[a1,1, a1,2, ..., a1,n], [a2,1, a2,2, ..., a2,n], ..., [am,1, am,2, ..., am,n]]).
```

Sebagai contoh, misalkan kita ingin mendefinisikan matriks:

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & -4 & -5 \end{bmatrix} \text{ dan } B = \begin{bmatrix} -1 & 2.3 & 0.5 \\ 0 & -4/5 & -5 \end{bmatrix}$$

pada SageMath, maka kita menggunakan skrip berikut:

```
A = matrix([[1,2,0], [0,-4,-5]]); print(A)
B = matrix([[[-1,2.3,0.5], [0,-4/5,-5]])); print(B)
```

SageMath menghasilkan keluaran berikut:

```
[ 1 2 0]
[ 0 -4 -5]
[ -1.00000000000000 2.30000000000000 0.500000000000000]
[ 0.00000000000000 -0.800000000000000 -5.00000000000000]
```

Melalui perintah `parent(A)` dan `parent(B)` kita dapat mengetahui struktur himpunan yang memuat  $A$  maupun  $B$  pada SageMath. Perintah ini memberikan keluaran berikut:

```
Full MatrixSpace of 2 by 3 dense matrices over Integer Ring
Full MatrixSpace of 2 by 3 dense matrices over Real Field with 53 bits of precision
```

Sebagaimana vektor, matriks direpresentasikan dalam struktur yang serupa dengan *list* pada SageMath. Matriks  $A$  yang berukuran  $m \times n$  (memiliki  $m$  baris dan  $n$  kolom) direpresentasikan dalam *list* dua dimensi dengan indeks baris dari 0 sampai dengan  $m - 1$  dan indeks kolom dari 0 hingga  $n - 1$ . Misalkan  $A$  adalah matriks  $2 \times 3$  yang telah dijelaskan sebelumnya dan kita menuliskan skrip berikut:

```
A = matrix([[1,2,0], [0,-4,-5]]);
print(A[0][0]);
print(A[0][1]);
print(A[0][2]);
print(A[1][0]);
print(A[1][1]);
print(A[1][2]);
```

Keluaran dari skrip ini adalah:

```

1
2
0
0
-4
-5

```

Ketika matriks pada persamaan (2.9) didefinisikan atas ring atau *field* tertentu—misalkan ring atau *field*  $R$ —maka kita dapat menggunakan sintaks berikut:

```
A = matrix(R, [[a1,1, a1,2, ..., a1,n], [a2,1, a2,2, ..., a2,n], ..., [am,1, am,2, ..., am,n]]),
```

sebagai contoh misalkan kita ingin mendefinisikan matriks  $A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -3 \\ 0 & 0 & -4 \end{bmatrix}$  dan  $B = \begin{bmatrix} 1 & -1 & 1 \\ 1 & -2 & 0 \\ -2 & 0 & 0 \end{bmatrix}$  atas  $\mathbb{F}_5$ , maka kita dapat menggunakan skrip berikut:

```

A = matrix(GF(5), [[1,2,3], [0,-2,-3], [0,0,-4]]); print(A)
B = matrix(GF(5), [[1,-1,1], [1,-2,0], [-2,0,0]]); print(B)

```

SageMath memberikan keluaran berikut:

```

[1 2 3]
[0 3 2]
[0 0 1]
[1 4 1]
[1 3 0]
[3 0 0]

```

Jika kita menggunakan Jupyter notebook pada SageMath, kita dapat memakai fungsi `show(...)` untuk mengonversi keluaran pada ke dalam bentuk formula matematis yang lebih mudah dibaca sebagai berikut:

```

A = matrix(GF(5), [[1,2,3], [0,-2,-3], [0,0,-4]]); show(A)
B = matrix(GF(5), [[1,-1,1], [1,-2,0], [-2,0,0]]); show(B)

```

Jupyter notebook akan memberikan keluaran sebagaimana pada Gambar 2.5.

```

In [1]: A = matrix(GF(5), [[1,2,3], [0,-2,-3], [0,0,-4]]); show(A)
B = matrix(GF(5), [[1,-1,1], [1,-2,0], [-2,0,0]]); show(B)

\left(\begin{array}{ccc} 1 & 2 & 3 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{array}\right)
\left(\begin{array}{ccc} 1 & 4 & 1 \\ 1 & 3 & 0 \\ 3 & 0 & 0 \end{array}\right)

```

Gambar 2.5: Tampilan hasil dari fungsi `show(A)` dan `show(B)` untuk  $A = \text{matrix}(\text{GF}(5), [[1,2,3], [0,-2,-3], [0,0,-4]])$  dan  $B = \text{matrix}(\text{GF}(5), [[1,-1,1], [1,-2,0], [-2,0,0]])$ .

Sebagaimana biasanya, SageMath otomatis mengonversi suatu bilangan “negatif” pada  $\mathbb{F}_5$  menjadi bilangan “positif” yang ekuivalen. Perkalian skalar, penjumlahan matriks, dan perkalian matriks pada SageMath dilakukan secara natural. Matriks identitas dengan ukuran  $n \times n$  didefinisikan dengan fungsi `identity_matrix(n)`. Berikut adalah beberapa perintah dasar kalkulasi matriks pada SageMath:

- Untuk setiap matriks persegi  $A$  kalkulasi *trace* dari  $A$ , yaitu `trace(A)`, dilakukan dengan sintaks `A.trace()`. Ingat bahwa `trace(A)` adalah jumlah komponen-komponen

---

diagonal dari  $\mathbf{A}$ .

2. Untuk setiap matriks  $\mathbf{A}$  kita dapat menentukan transpos dari  $\mathbf{A}$ , yaitu  $\mathbf{A}^T$ , dengan sintaks  $\mathbf{A}.transpose()$ .
3. Untuk setiap matriks persegi  $\mathbf{A}$  kalkulasi determinan dari  $\mathbf{A}$ , yaitu  $\det(\mathbf{A})$ , dilakukan dengan sintaks  $\mathbf{A}.determinant()$ .
4. Untuk setiap matriks persegi  $\mathbf{A}$ , ketika  $\det(\mathbf{A})$  adalah suatu unit pada ring yang ditinjau, maka kita dapat menentukan invers dari  $\mathbf{A}$ , yaitu  $\mathbf{A}^{-1}$ , menggunakan sintaks  $\mathbf{A}^{-1}$  maupun  $\mathbf{A}.inverse()$ .

Misalkan  $\mathbf{A}$  dan  $\mathbf{B}$  adalah dua matriks  $3 \times 3$  atas  $\mathbb{F}_5$  yang telah dijelaskan sebelumnya dan kita ingin menghitung nilai-nilai berikut:

1.  $\mathbf{C} = 2\mathbf{A} - 3\mathbf{B}$
2.  $\mathbf{D} = \mathbf{AB} - \mathbf{BA}$
3.  $\mathbf{E} = \mathbf{A}^T$
4.  $d = \det(\mathbf{B})$
5.  $\mathbf{F} = \mathbf{B}^{-1}$
6.  $t = \text{trace}(\mathbf{F})$

Kita dapat menggunakan skrip berikut pada SageMath:

```
A = matrix(GF(5), [[1,2,3], [0,-2,-3], [0,0,-4]]);  
B = matrix(GF(5), [[1,-1,1], [1,-2,0], [-2,0,0]]);  
C = 2*A - 3*B; print(C)  
D = A*B - B*A; print(D)  
E = A.transpose(); print(E)  
d = B.determinant(); print(d)  
F = B^-1; print(F)  
t = F.trace(); print(t)
```

Keluaran dari skrip ini adalah:

```
[4 2 3]  
[2 2 4]  
[1 0 2]  
[1 1 4]  
[3 3 1]  
[0 4 1]  
[1 0 0]  
[2 3 0]  
[3 2 1]  
1  
[0 0 2]  
[0 2 1]  
[1 2 4]  
1
```

Pada SageMath himpunan seluruh matriks berukuran  $m \times n$  atas ring  $R$ , atau ruang matriks (*matrix space*)  $R^{m \times n}$  atau  $M_{m \times n}(R)$ , didefinisikan dengan perintah `MatrixSpace(R, m, n)`.

Misalkan kita ingin mendefinisikan himpunan seluruh matriks berukuran  $2 \times 3$  atas  $\mathbb{Z}_4$ , maka kita dapat menggunakan skrip berikut:

```
MS = MatrixSpace(Integer(4), 2, 3); print(MS)
print(MS.dimension())
print(MS.dims())
```

Keluaran dari skrip ini adalah:

```
Full MatrixSpace of 2 by 3 dense matrices over Ring of integers modulo 4
6
(2, 3)
```

*Method .dimension()* pada skrip ini mengembalikan nilai dimensi dari ruang matriks yang ditinjau, sedangkan *method .dims()* memberikan ukuran dari matriks yang ada pada ruang matriks tersebut. Untuk mendefinisikan sebuah matriks  $A$  yang berada pada ruang matriks tertentu yang dinotasikan dengan  $MS$ , kita dapat menggunakan sintaks  $MS$  di depan definisi matriks tersebut. Misalkan kita ingin mendefinisikan matriks  $A = \begin{bmatrix} -1 & 2 & -3 \\ 3 & -2 & 1 \end{bmatrix}$  pada  $M_{2 \times 3}(\mathbb{Z}_4)$ , maka kita dapat menggunakan skrip berikut:

```
MS = MatrixSpace(Integer(4), 2, 3)
A = MS([[-1, 2, -3], [3, -2, 1]]); print(A)
```

SageMath akan memberikan keluaran berupa matriks  $A = \begin{bmatrix} 3 & 2 & 1 \\ 3 & 2 & 1 \end{bmatrix}$ , perhatikan bahwa SageMath secara otomatis akan mengubah nilai “negatif” menjadi nilai “positif”.

Pada SageMath sebuah vektor  $v \in R^n$  direpresentasikan dalam matriks kolom ketika dikalikan dengan matriks dari sebelah kanan dan direpresentasikan dalam matriks baris ketika dikalikan dengan matriks dari sebelah kiri. Misalkan  $A$  adalah sebuah matriks atas  $R$  yang berukuran  $m \times n$ ,  $u \in R^n$ , dan  $v \in R^m$ , maka kita dapat mendefinisikan nilai dari  $Au$  dan  $vA$ . Pada ekspresi  $Au$  SageMath memperlakukan  $u$  sebagai matriks kolom sehingga hasil dari  $Au$  adalah sebuah matriks berukuran  $m \times 1$  atau sebuah vektor kolom dengan  $m$  komponen. SageMath menuliskan vektor ini dalam bentuk  $m$ -tupel. Kemudian pada ekspresi  $vA$  SageMath memperlakukan  $v$  sebagai matriks baris sehingga hasil  $vA$  adalah sebuah matriks berukuran  $1 \times n$  atau sebuah vektor baris dengan  $n$  komponen. SageMath menuliskan vektor ini dalam bentuk  $n$ -tupel.

Untuk memperjelas perhatikan contoh berikut. Misalkan  $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 1 \end{bmatrix}$  adalah sebuah matriks atas  $\mathbb{F}_5$ ,  $u = (1, 0, 3) \in \mathbb{F}_5^3$ , dan  $v = (3, 4) \in \mathbb{F}_5^2$ , maka SageMath dapat melakukan kalkulasi-kalkulasi berikut:

$$1. \quad Au = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}. \text{ SageMath menghasilkan } Au = (0, 2).$$

$$2. \quad vA = [3 \ 4] \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 1 \end{bmatrix} = [4 \ 1 \ 3]. \text{ SageMath menghasilkan } vA = (4, 1, 3).$$

**Latihan 2.6.2** Gunakan SageMath untuk menghitung nilai-nilai berikut apabila semua matriks dan vektor ditinjau atas ring bilangan bulat  $\mathbb{Z}$ .

$$1. \quad \begin{bmatrix} 1 & 3 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 4 & 8 \\ 9 & 15 \end{bmatrix}$$

2. Periksa apakah terdapat vektor  $u \in \mathbb{Z}^2$  yang memenuhi  $Au = v$  untuk matriks-matriks  $A$  dan vektor-vektor  $v$  berikut:

$$(a) \quad A = \begin{bmatrix} 1 & 4 \\ 2 & 7 \end{bmatrix} \text{ dan } v = \begin{bmatrix} 10 \\ 9 \end{bmatrix}.$$

---


$$(b) A = \begin{bmatrix} 4 & 6 \\ 8 & -2 \end{bmatrix} \text{ dan } v = \begin{bmatrix} 9 \\ -12 \end{bmatrix}.$$

## 2.7 Kurva Eliptik (*Elliptic Curve*) pada SageMath

Kurva eliptik (*elliptic curve*) adalah salah satu struktur matematika yang dipakai pada kriptografi asimetrik. Misalkan  $R$  adalah sebuah ring, sebuah kurva eliptik atas  $R$  dalam bentuk persamaan Weierstrass yang panjang adalah persamaan dalam bentuk

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad (2.10)$$

dengan  $a_1, a_2, a_3, a_4, a_6 \in R$ . Kurva eliptik yang dibangkitkan dengan persamaan (2.10) merupakan himpunan pasangan terurut  $(x, y) \in R \times R$  yang memenuhi persamaan tersebut. Pada SageMath kurva eliptik yang dibangkitkan dengan persamaan (2.10) atas ring  $R$  dapat dikonstruksi dengan perintah `EllipticCurve(R, [a1, a2, a3, a4, a6])`. Pada kriptografi kita sering kali bekerja dengan kurva eliptik yang dinyatakan dalam persamaan berbentuk

$$y^2 = x^3 + a_4x + a_6, \quad (2.11)$$

dengan  $a_4, a_6 \in R$ . Perhatikan bahwa persamaan (2.11) pada dasarnya adalah persamaan (2.10) dengan nilai  $a_1 = a_2 = a_3 = 0$ . Kurva eliptik dengan persamaan (2.11) atas ring  $R$  pada SageMath dapat dikonstruksi dengan perintah `EllipticCurve(R, [a4, a6])`. Pada perintah `EllipticCurve(R, [a1, a2, a3, a4, a6])` maupun `EllipticCurve(R, [a4, a6])` kita boleh tidak menuliskan argumen  $R$ . Dalam hal ini secara *default* ring yang ditinjau adalah  $\mathbb{Q}$ .

Pada kriptografi yang memanfaatkan kurva eliptik kita biasanya bekerja pada kurva eliptik atas  $\mathbb{F}_p$  untuk bilangan prima  $p \geq 3$ . Definisi kurva eliptik yang kita tinjau untuk kriptografi dijelaskan pada Definisi 2.7.1.

**Definisi 2.7.1 (Definisi kurva eliptik atas  $\mathbb{F}_p$  [2])** Misalkan  $p \geq 3$  adalah sebuah bilangan prima. Sebuah kurva eliptik atas  $\mathbb{F}_p$  adalah persamaan dalam bentuk

$$E : y^2 = x^3 + ax + b, \text{ dengan } a, b \in \mathbb{F}_p \text{ memenuhi } 4a^3 + 27b^2 \neq 0. \quad (2.12)$$

Selanjutnya titik-titik pada kurva eliptik  $E$  dengan koordinat-koordinat atas  $\mathbb{F}_p$  adalah himpunan

$$E(\mathbb{F}_p) = \{(x, y) : x, y \in \mathbb{F}_p \text{ dan memenuhi } y^2 = x^3 + a + b\} \cup \{\mathcal{O}\},$$

dengan  $\mathcal{O}$  adalah titik tak hingga (*point at infinity*) yang merupakan elemen identitas penjumlahan pada  $E(\mathbb{F}_p)$ .

Perhatikan bahwa persamaan (2.12) pada Definisi 2.7.1 identik dengan persamaan (2.11). Misalkan kita menuliskan operator penjumlahan pada kurva eliptik sebagai  $\oplus$ . Pada kurva eliptik operasi penjumlahan titik  $P_1(x_1, y_1)$  dan  $P_2(x_2, y_2)$  dilakukan dengan aturan berikut:<sup>10</sup>

1. Jika salah satu dari  $P_1$  atau  $P_2$  adalah  $\mathcal{O}$  (*point at infinity*), maka kita memiliki  $P_1 \oplus \mathcal{O} = P_1$  dan  $\mathcal{O} \oplus P_2 = P_2$ .
2. Jika  $P_1(x_1, y_1)$  dan  $P_2(x_2, y_2)$  dengan  $x_1 = x_2$  dan  $y_1 = -y_2$ , maka  $P_1 \oplus P_2 = \mathcal{O}$ . Dengan perkataan lain  $P(x, y) \oplus P(x, -y) = \mathcal{O}$  untuk setiap titik  $P(x, y)$  pada kurva

---

<sup>10</sup>Pembaca dapat melihat [2, Teorema 6.6] untuk penjelasan lebih detail mengenai operasi penjumlahan dan perkalian skalar pada kurva eliptik.

---

eliptik yang kita tinjau. Dari sini kita juga memiliki  $\ominus P(x, y) = P(x, -y)$ . Di sini  $\ominus P(x, y)$  menyatakan invers dari titik  $P(x, y)$  terhadap operasi  $\oplus$ .

3. Jika  $P_1(x_1, y_1)$  dan  $P_2(x_2, y_2)$  adalah dua titik yang berbeda dengan  $x_1 \neq x_2$  atau  $y_1 \neq -y_2$ , maka  $P_1(x_1, y_1) \oplus P_2(x_2, y_2)$  dilakukan dengan langkah-langkah berikut:
  - (a) Definisikan  $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ . Semua operasi dilakukan pada  $\mathbb{F}_q$  dan ekspresi  $\frac{\alpha}{\beta}$  berarti  $\alpha\beta^{-1}$ .
  - (b) Definisikan  $x_3 = \lambda^2 - x_1 - x_2$  dan  $y_3 = \lambda(x_1 - x_3) - y_1$ .
  - (c) Definisikan  $P_1(x_1, y_1) \oplus P_2(x_2, y_2) = P_3(x_3, y_3)$ .
4. Jika  $P_1(x_1, y_1)$  dan  $P_2(x_2, y_2)$  adalah dua titik yang sama, maka  $P_1(x_1, y_1) \oplus P_2(x_2, y_2) = 2P_1(x_1, y_1)$  dilakukan dengan langkah-langkah berikut:
  - (a) Definisikan  $\lambda = \frac{3x_1^2 + a}{2y_1}$  dengan  $a$  berasal dari persamaan kurva eliptik pada persamaan (2.12) yang dijelaskan pada Definisi 2.7.1 serta ekspresi  $\frac{\alpha}{\beta}$  berarti  $\alpha\beta^{-1}$ .
  - (b) Definisikan  $x_3 = \lambda^2 - x_1 - x_2$  dan  $y_3 = \lambda(x_1 - x_3) - y_1$ .
  - (c) Definisikan  $P_1(x_1, y_1) \oplus P_1(x_1, y_1) = 2P_1(x_1, y_1) = P_3(x_3, y_3)$ .

Pada kurva eliptik perkalian skalar antara sebuah bilangan bulat dengan sebuah titik didefinisikan sebagai penjumlahan titik identik yang dilakukan secara berulang. Hal ini berarti jika  $P$  adalah sebuah titik pada suatu kurva eliptik dan  $k$  adalah sebuah bilangan bulat positif, maka

$$kP = \underbrace{P \oplus P \oplus \cdots \oplus P}_{n \text{ suku}},$$

dengan  $\oplus$  adalah penjumlahan titik yang telah dijelaskan sebelumnya.

Perhatikan bahwa untuk meringkas penulisan, kita menuliskan himpunan semua titik pada kurva eliptik atas  $\mathbb{F}_p$  (termasuk *point at infinity* yang merupakan elemen identitas) sebagai  $E(\mathbb{F}_p)$ . Misalkan kita meninjau kurva eliptik

$$y^2 = x^3 + 3x + 8 \text{ atas } \mathbb{F}_{13}. \quad (2.13)$$

Untuk mencari titik-titik pada  $E(\mathbb{F}_{13})$  kita perlu mencari semua  $(x, y)$  dengan  $x, y \in \mathbb{F}_{13}$  yang memenuhi persamaan (2.13). Salah satu caranya adalah dengan mensubstitusikan nilai  $x = 0, 1, 2, \dots, 12$  dan menghitung nilai  $x^3 + 3x + 8$  dan mencari nilai yang merupakan kuadrat sempurna pada  $\mathbb{F}_{13}$ . Pada SageMath hal ini dapat dilakukan dengan lebih ringkas menggunakan skrip berikut:

```
E = EllipticCurve(GF(13), [3, 8]); print(E)
for x in E:
    print(x)
```

Keluaran dari skrip ini adalah semua elemen pada  $E(\mathbb{F}_{13})$  yang dijelaskan pada persamaan (2.13):

```
Elliptic Curve defined by y^2 = x^3 + 3*x + 8 over Finite Field of size 13
(0 : 1 : 0)
(1 : 5 : 1)
(1 : 8 : 1)
(2 : 3 : 1)
(2 : 10 : 1)
(9 : 6 : 1)
(9 : 7 : 1)
(12 : 2 : 1)
(12 : 11 : 1)
```

---

Pada SageMath titik  $x, y$  direpresentasikan dalam bentuk  $(x : y : e)$  dengan  $e$  ber nilai 0 jika titik  $(x, y)$  dianggap sebagai *point at infinity*. Dari hasil SageMath kita memperoleh

$$E(\mathbb{F}_{13}) = \{\mathcal{O}, (1, 5), (1, 8), (2, 3), (2, 10), (9, 6), (9, 7), (12, 2), (12, 1)\}.$$

Misalkan kita ingin menghitung nilai-nilai berikut pada  $E(\mathbb{F}_{13})$  untuk kurva yang dijelaskan pada persamaan (2.13):

1.  $(1, 5) \oplus (1, 8)$
2.  $\ominus(1, 5)$
3.  $(1, 8) \oplus (2, 3)$
4.  $2(1, 5)$
5.  $3(1, 8)$
6.  $2(1, 8) \ominus 3(12, 11)$

maka kita dapat menggunakan skrip berikut:

```
P1 = E(1,5) + E(1,8); print(P1)
P2 = -E(1,5); print(P2)
P3 = E(1,8) + E(2,3); print(P3)
P4 = 2*E(1,5); print(P4)
P5 = 3*E(1,8); print(P5)
P6 = 2*E(1,8) - 3*E(12,11); print(P6)
```

SageMath memberikan keluaran berikut:

```
(0 : 1 : 0)
(1 : 8 : 1)
(9 : 6 : 1)
(2 : 10 : 1)
(9 : 6 : 1)
(1 : 5 : 1)
```

Keluaran ini berarti:

1.  $(1, 5) \oplus (1, 8) = \mathcal{O}$
2.  $\ominus(1, 5) = (1, 8)$
3.  $(1, 8) \oplus (2, 3) = (9, 6)$
4.  $2(1, 5) = (2, 10)$
5.  $3(1, 8) = (9, 6)$
6.  $2(1, 8) \ominus 3(12, 11) = (1, 5)$

**Latihan 2.7.2** Misalkan  $E(\mathbb{F}_p)$  adalah sebuah kurva eliptik atas *field*  $\mathbb{F}_p$ . Orde dari suatu titik  $P(x, y) \in E(\mathbb{F}_p)$  adalah bilangan bulat positif terkecil  $k$  yang memenuhi  $kP = \mathcal{O}$ , dengan  $\mathcal{O}$  adalah *point at infinity*. Apabila tidak ada bilangan bulat positif  $k$  yang memenuhi maka kita mengatakan bahwa orde dari  $P$  adalah tak hingga. Selanjutnya sebuah titik  $Q(x, y) \in E(\mathbb{F}_p)$  dikatakan sebagai pembangkit (*generator*) dari  $E(\mathbb{F}_p)$  apabila setiap titik  $R(x, y) \in E(\mathbb{F}_p)$  dapat dinyatakan sebagai kelipatan dari  $Q$ , dengan perkataan lain  $R = \alpha Q$  untuk suatu bilangan bulat  $\alpha$ .

- 
1. Tentukan orde dari semua elemen  $E(\mathbb{F}_{13})$  untuk kurva eliptik  $E$  yang dijelaskan pada persamaan (2.13).
  2. Periksa apakah  $E(\mathbb{F}_{13})$  yang dibangkitkan dengan persamaan (2.13) memiliki titik pembangkit atau tidak. Jika kurva tersebut memiliki titik pembangkit, tentukan salah satu titik pembangkitnya.

# Topik 3

## Contoh Kriptografi Klasik pada SageMath

Ketika komputer belum ditemukan, kriptografi hanya diterapkan menggunakan kode yang mengacu pada pengacakan karakter. Meskipun saat ini kriptografi klasik dapat dipecahkan dengan mudah menggunakan bantuan komputer, ilmu di dalamnya merupakan dasar pengembangan dari kriptografi modern. Dari kriptografi klasik kita dapat mempelajari berbagai konsep mengenai cara membangun keamanan sistem maupun menemukan kelemahannya. Hal ini dapat memperluas kemampuan analisis kriptografi siapa pun yang mempelajarinya. Untuk memahami hal ini dengan baik terlebih dulu kita akan mengkaji mengenai domain karakter dan konversi huruf ke angka yang dapat digunakan dalam kriptografi.

### 3.1 Domain Karakter dan Konversi dari Huruf ke Angka

Sebelum membahas mengenai kriptografi klasik lebih lanjut, kita akan membahas domain karakter atau alfabet yang didukung SageMath untuk kriptografi klasik. Pada tutorial ini kita akan meninjau domain karakter berupa himpunan huruf kapital, string heksadesimal, string biner, string oktal, dan string Radix-64. Selain itu kita juga akan membahas metode konversi dari huruf ke angka (dan sebaliknya) berdasarkan ASCII yang akan digunakan dalam kriptografi modern.

#### 3.1.1 Domain Karakter Huruf Kapital

Huruf kapital pada SageMath bisa diakses dengan perintah `AlphabeticStrings()`. Perhatikan skrip berikut:

```
a = AlphabeticStrings()
print(a.gen(0))
print(a.gen(25))
print(a([i for i in range(25)]))
```

Keluaran dari skrip ini adalah:

```
A
Z
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Perintah `.gen(i)` digunakan untuk membangkitkan elemen dari `class AlphabeticStrings()` pada posisi ke-*i*. Nilai *i* pada method `.gen(i)` haruslah memenuhi  $0 \leq i \leq 25$ . Jika *i* = 0 maka kita memperoleh karakter A dan jika *i* = 25 kita memperoleh karakter Z. Pada SageMath terdapat fungsi *built-in* yang sering digunakan yaitu menghitung banyaknya kemunculan karakter dan distribusi frekuensinya. Perhatikan skrip berikut:

---

```
a = AlphabeticStrings().encoding('saya bisa')
print(sorted(a.character_count().items()))
print(sorted(a.frequency_distribution().function().items()))
```

Keluaran dari skrip ini adalah:

```
[(A, 3), (B, 1), (I, 1), (S, 2), (Y, 1)]
[(A, 0.3750000000000000), (B, 0.1250000000000000),
(I, 0.1250000000000000), (S, 0.2500000000000000),
(Y, 0.1250000000000000)]
```

Skrip ini menghitung banyaknya kemunculan setiap karakter pada suatu string masukan. Perhatikan bahwa string masukan saya bisa memuat 3 huruf A, 2 huruf S, dan masing-masing satu huruf B, I, dan Y. Pada SageMath kita dapat melakukan hal ini menggunakan *method .character\_count()*. Distribusi dari suatu karakter dihitung dengan membagi banyaknya kemunculan suatu karakter dengan banyaknya kemunculan seluruh karakter yang ada pada suatu string. Dari skrip sebelumnya kita melihat bahwa pada SageMath hal ini dapat kita lakukan menggunakan *method .frequency\_distribution().function()*. Sebagai contoh pada string saya bisa banyaknya kemunculan S adalah 2, sehingga distribusi dari karakter S adalah  $\frac{2}{8} = 0.25$ . SageMath juga mendukung perintah *.encoding* yang digunakan untuk mengubah format dari karakter masukan menjadi *class AlphabeticStrings()*. Keluaran terkait perhitungan kemunculan karakter maupun distribusi frekuensinya juga dapat diurutkan dengan perintah *sorted*.

### 3.1.2 Domain Karakter Heksadesimal

Domain yang merepresentasikan karakter heksadesimal pada SageMath bisa diakses dengan perintah *HexadecimalStrings()*. Perhatikan skrip berikut:

```
h = HexadecimalStrings()
print(h([ i for i in range(16)]))
h = HexadecimalStrings().encoding('saya bisa')
print(h)
print(sorted(h.character_count().items()))
print(sorted(h.frequency_distribution().function().items()))
```

Kita dapat menekan tombol *Ctrl+Enter* pada *keyboard* untuk memperoleh keluaran dari skrip ini:

```
0123456789abcdef
736179612062697361
[(0, 1), (1, 3), (2, 2), (3, 2), (6, 5), (7, 3), (9, 2)]
[(0, 0.0555555555555556)
(1, 0.1666666666666667)
(2, 0.111111111111111)
(3, 0.111111111111111)
(6, 0.2777777777777778)
(7, 0.1666666666666667)
(9, 0.111111111111111)]
```

Ingat kembali bahwa struktur heksadesimal terdiri atas 16 karakter berbeda, yaitu bilangan bulat mulai dari 0 hingga 9 serta alfabet mulai dari a hingga f. Pada contoh skrip yang dijelaskan string saya bisa direpresentasikan sebagai 736179612062697361 dalam heksadesimal. Struktur heksadesimal juga dilengkapi dengan *method encoding*, *.character\_count()*, dan *.frequency\_distribution().function()* sebagaimana telah dijelaskan untuk struktur domain karakter huruf kapital pada Subtopik 3.1.1.

### 3.1.3 Domain Karakter String Biner

Domain karakter string biner pada SageMath dapat diakses dengan perintah `BinaryStrings()`. Perhatikan skrip berikut:

```
b = BinaryStrings()
print(b([ i for i in range(2)]))
b = BinaryStrings().encoding('saya bisa')
print(b)
print(sorted(b.character_count().items()))
print(sorted(b.frequency_distribution().function().items()))
```

Keluaran dari skrip ini adalah:

```
01
0111001101100001011110010110000100100000011000100110100101
11001101100001
[(0, 40), (1, 32)]
[(0, 0.5555555555555555), (1, 0.4444444444444445)]
```

Karakter pada representasi string biner adalah 0 dan 1 saja. Pada contoh yang dijelaskan, string `saya bisa` direpresentasikan sebagai string biner

```
0111001101100001011110010110000100100000011000100110100101.
```

Struktur string biner juga dilengkapi dengan *method* `encoding`, `.character_count()`, dan `.frequency_distribution().function()` sebagaimana telah dijelaskan untuk struktur domain karakter huruf kapital pada Subtopik 3.1.1 maupun struktur domain karakter heksadesimal pada Subtopik 3.1.2.

### 3.1.4 Domain Karakter Oktal

Domain karakter oktal pada SageMath bisa diakses dengan perintah `OctalStrings()`. Perhatikan skrip berikut:

```
o = OctalStrings()
print(o([ i for i in range(8)]))
ochar = o([0,1,7])
print(ochar.character_count().items())
print(ochar.frequency_distribution().function().items())
```

Untuk memperoleh hasil kalkulasi, kita dapat menekan tombol `Ctrl+Enter` pada *keyboard* sehingga diperoleh:

```
01234567
dict_items([(0, 1), (1, 1), (7, 1)])
dict_items([(0, 0.3333333333333333), (1, 0.3333333333333333),
(7, 0.3333333333333333)])
```

Karakter yang direpresentasikan oleh struktur oktal adalah bilangan bulat mulai dari 0 sampai dengan 7 (inklusif). Sebagaimana struktur-struktur sebelumnya struktur ini juga dilengkapi dengan *method* `.character_count()` maupun `.frequency_distribution()`. Namun berbeda dengan struktur-struktur sebelumnya, kedua *method* tersebut tidak memberikan keluaran berupa *list*, melainkan berupa *dictionary*. Selain itu struktur oktal juga tidak dilengkapi dengan *method* `.encoding()` sebagaimana struktur-struktur sebelumnya.

### 3.1.5 Domain Karakter Radix-64

Domain karakter string Radix-64 dapat diakses dengan perintah `Radix64Strings()`. Perhatikan skrip berikut:

---

```

Rad64 = Radix64Strings()
Rad64([ i for i in range(64) ])
rchar = Rad64([0,2,33,40])
print(rchar)
print(rchar.character_count().items())
print(rchar.frequency_distribution().function().items())

```

Untuk memperoleh hasil kalkulasi, kita dapat menekan tombol Ctrl+Enter pada keyboard sehingga diperoleh:

```

ACho
dict_items([(A, 1), (C, 1), (h, 1), (o, 1)])
dict_items([(A, 0.2500000000000000), (C, 0.2500000000000000),
(h, 0.2500000000000000), (o, 0.2500000000000000)])

```

Struktur Radix-64 memiliki keserupaan dengan struktur oktal yang dijelaskan pada Subtopik 3.1.4. Struktur ini juga dilengkapi dengan *method* .character\_count() maupun .frequency\_distribution(). Kedua *method* tersebut memberikan keluaran berupa *dictionary*. Struktur Radix-64 juga tidak dilengkapi dengan *method* .encoding().

### 3.1.6 Konversi Huruf ke Angka Menggunakan ASCII

Seiring dengan perkembangan teknologi perangkat keras, ketika kriptografi melibatkan mesin yang lebih kompleks, pesan yang perlu dikirimkan sering kali tidak hanya melibatkan huruf dan angka saja, namun juga tanda baca dan karakter-karakter non-alfanumerik lainnya. Pada komputer kita mengenal kode ASCII<sup>1</sup> yang memetakan huruf, angka, tanda baca, serta karakter non-alfanumerik ke dalam string biner dengan panjang 7. Dengan perkataan lain pada ASCII sebuah huruf/angka/tanda baca/karakter non-alfanumerik diwakili oleh sebuah bit string dengan panjang 7. Kita juga dapat memandang bahwa karakter-karakter tersebut dapat dinyatakan secara unik dalam bilangan bulat antara 0 sampai dengan 127 (inklusif). Pada Python kita dapat mengonversi sebuah string ke dalam *list* yang memuat nilai ASCII (dalam rentang 0 sampai dengan 127, inklusif) dari masing-masing karakter pada string tersebut menggunakan fungsi *ord*, perhatikan contoh berikut:

```

mystring = 'Ayo Belajar!'
print(mystring); print(len(mystring))
mylist = [ord(x) for x in mystring]
print(mylist); print(len(mylist))

```

Pada skrip ini kita mengonversi sebuah string Ayo Belajar! yang memuat 12 karakter (termasuk spasi dan tanda seru). Keluaran dari skrip ini adalah:

```

Ayo Belajar!
12
[65, 121, 111, 32, 66, 101, 108, 97, 106, 97, 114, 33]
12

```

Pada hasil keluaran SageMath kita melihat bahwa *list* yang dihasilkan memuat 12 komponen. Perhatikan bahwa nilai ASCII dari huruf A, y, dan o secara berturut-turut adalah 65, 121, dan 111. Kemudian kita juga dapat memeriksa bahwa nilai ASCII dari spasi adalah 32 dan nilai ASCII dari tanda seru (!) adalah 33. Kita dapat dengan mudah mengonversi *list* yang memuat nilai-nilai ASCII ke dalam struktur matematika lain, seperti bilangan bulat maupun vektor. Salah satu tekniknya adalah dengan meninjau *list* nilai-nilai ASCII sebagai representasi bilangan bulat dalam basis 128 (lihat Subtopik 2.3.2 untuk metode koversi ini). Lebih jauh, teknik dari konversi ini akan ditinjau kembali pada Topik 5.

---

<sup>1</sup>ASCII merupakan akronim dari *American Standard Code for Information Interchange*. Kode ASCII dikembangkan dari kode telegraf di Amerika Serikat pada tahun 1960-an.

---

Selanjutnya untuk mengonversi sebuah *list* yang memuat nilai-nilai ASCII ke dalam suatu string kita dapat menggunakan fungsi `chr` pada Python. Untuk setiap nilai  $x \in [0, 127]$ , `chr(x)` mengembalikan karakter yang nilai ASCII-nya adalah  $x$ . Untuk menyatukan *list* yang berisi karakter-karakter tersebut ke dalam satu string kita dapat menggunakan *method* `.join`. Sebagai ilustrasi perhatikan contoh berikut:

```
mylist = [65, 121, 111, 32, 66, 101, 108, 97, 106, 97, 114, 33]
print(mylist)
mystring0 = [chr(x) for x in mylist]
print(mystring0)
mystring1 = ''.join(mystring0)
print(mystring1)
```

Pada skrip ini `mystring0` memuat *list* dengan panjang 12 yang isinya adalah karakter-karakter yang memiliki nilai ASCII pada `mylist`. Untuk menggabungkan entri-entri pada `mystring0` menjadi sebuah *list* kita menggunakan perintah `''.join(mystring0)`. Tanda kutip buka dan tutup menunjukkan bahwa tidak ada pemisah antar karakter pada string yang dihasilkan. Keluaran dari skrip ini adalah:

```
[65, 121, 111, 32, 66, 101, 108, 97, 106, 97, 114, 33]
['A', 'y', 'o', ' ', 'B', 'e', 'l', 'a', 'j', 'a', 'r', '!']
Ayo Belajar
```

### Latihan 3.1.1

1. Lakukan konversi pada teks '*saya pasti bisa*' menjadi bentuk string heksadesimal kemudian hitung kemunculan tiap karakternya.
2. Konversikan teks '*soal ini mudah*' menjadi bentuk string biner kemudian hitung distribusi frekuensi dari setiap karakternya.
3. Buatlah sebuah fungsi dengan nama `string_to_ASCII` yang mengonversi sebuah string menjadi sebuah *list* yang memuat nilai-nilai ASCII dari karakter-karakter yang bersesuaian pada string tersebut. Sebagai contoh perintah `string_to_ASCII('Ayo makan!')` memberikan keluaran berupa *list* `[65, 121, 111, 32, 109, 97, 107, 97, 110, 33]`.
4. Buatlah sebuah fungsi dengan nama `ASCII_to_string` yang mengonversi sebuah *list* yang memuat nilai-nilai ASCII (dalam rentang 0 sampai dengan 127, inklusif) ke dalam string yang bersesuaian. Sebagai contoh perintah `ASCII_to_string(L)` untuk `L = [65, 121, 111, 32, 109, 97, 107, 97, 110, 33]` memberikan keluaran berupa string *Ayo makan!*.

## 3.2 *Substitution Cipher* (Sandi Substitusi)

*Substitution cipher* dapat diartikan sebagai metode enkripsi klasik yang mengganti setiap unit teks pesan (*plaintext*) dengan sebuah karakter teks sandi (*ciphertext*). Pada sandi substitusi kunci ditentukan terlebih dulu dengan tetap memperhatikan urutan posisi antar karakter. Unit pada teks pesan atau teks sandi dapat berupa karakter tunggal (*monoalphabetic*) atau lebih dari sebuah karakter (*polyalphabetic*). Kita akan meninjau beberapa jenis sandi substitusi, yaitu sandi *affine*, sandi Caesar (atau sandi geser), *decimation cipher*, sandi Hill, dan sandi Vigenère.

### 3.2.1 *Affine Cipher* (Sandi *Affine*)

*Affine cipher* merupakan salah satu contoh dari *monoalphabetic substitution cipher*. Pada SageMath, *affine cipher* ada pada class sage.crypto.classical.AffineCryptosystem. Dalam matematika sebuah pemetaan (transformasi)  $f : A \rightarrow B$  dikatakan sebagai transformasi *affine* jika  $f(x) = ax + b$  untuk suatu  $a, b \in A$ . Di sini kita memiliki  $ax \in B$  dan  $+$  adalah operasi penjumlahan pada  $B$ . Misalkan  $A = \{a_0, a_1, a_2, \dots, a_{n-1}\}$  adalah himpunan alfabet yang tidak kosong dan terdiri dari sejumlah  $n$  elemen yang berbeda-beda. Kita dapat mendefinisikan pemetaan

$$f : A \rightarrow \mathbb{Z}_n$$

dari alfabet  $A$  ke  $\mathbb{Z}_n$  untuk ring bilangan bulat modulo  $n$  dengan nilai  $f(a_i) = i$ . Dalam hal ini setiap elemen pada alfabet  $A$  dapat diidentifikasi secara unik dengan sebuah nilai bilangan bulat pada rentang  $0 \leq i \leq n - 1$ .

Sandi *affine* memiliki pasangan kunci rahasia  $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_n$  dengan  $\gcd(a, n) = 1$ . Jika diketahui  $p$  adalah teks pesan (*plaintext*) dengan  $p \in \mathbb{Z}_n$ , maka teks sandi (*ciphertext*)  $c$  dari nilai  $p$  tersebut didefinisikan sebagai

$$c \equiv (ap + b) \pmod{n}.$$

Jika diketahui teks sandi (*ciphertext*)  $c \in \mathbb{Z}_n$  dan pasangan kunci rahasia  $(a, b)$ , maka nilai teks pesan (*plaintext*) dapat diketahui melalui kongruensi

$$p \equiv (a^{-1}(c - b)) \pmod{n},$$

dengan  $a^{-1}$  adalah nilai invers dari  $a$  mod  $n$ . Pada SageMath domain karakter yang didukung oleh sandi *affine* secara *default* adalah AlphabeticsStrings(). Contoh penerapan sandi *affine* pada SageMath dengan  $a = 3$  dan  $b = 7$  dapat dilihat pada skrip berikut:

```
A = AffineCryptosystem(AlphabeticStrings())
P = A.encoding("ini pesan rahasia")
print(P)
(a,b) = (3,7)
C = A.enciphering(a,b,P)
print(C)
```

Untuk memperoleh hasil enkripsi, kita dapat menekan tombol Ctrl+Enter pada *keyboard* sehingga diperoleh:

```
INIPEANRAHASIA
FUFATJHUGHCHJFH
```

Pada enkripsi yang dilakukan kita menggunakan fungsi  $f(x) = (3x + 7) \pmod{26}$ . Misalkan huruf A dipetakan ke bilangan 0, B dipetakan ke bilangan 1, dan seterusnya hingga Z dipetakan ke bilangan 25. Fungsi  $f(x) = (3x + 7) \pmod{26}$  memetakan bilangan  $x$  ke bilangan  $(3x + 7) \pmod{26}$ . Sebagai contoh, hasil enkripsi dari huruf A adalah huruf H karena nilai dari  $f(0) = 7$ . Perhatikan bahwa library AffineCryptosystem tidak memandang spasi sebagai entitas tersendiri. Dengan perkataan lain spasi pada string masukan akan diabaikan dalam proses enkripsi (maupun dekripsi).

Penerapan proses dekripsi dan pengecekan apakah hasilnya sama dengan teks pesan (*plaintext*) dapat dijalankan dengan dua cara sebagaimana dijelaskan pada skrip berikut:

```

(a,b) = (3,7)
#cara 1
P1 = A.deciphering(a,b,C)
print(P1)
print(P1==P)

#cara 2
(ainv, binv) = A.inverse_key(a,b)
P2 = A. enciphering(ainv,binv,C)
print(P2)
print(P2==P)

```

Pada cara pertama kita menggunakan *method* `.deciphering` sedangkan pada cara kedua kita memakai *method* `enciphering` namun dengan kunci enkripsi `(ainv,binv)`. Untuk memperoleh hasil dekripsi, kita dapat menekan tombol `Ctrl+Enter` pada *keyboard* sehingga diperoleh:

```

INIPE SANRAHASIA
True
INIPE SANRAHASIA
True

```

Nilai kunci rahasia juga bisa didapatkan secara acak dari fungsi *built-in* yang ada di Sage-Math

```

A = AffineCryptosystem(AlphabeticStrings())
a,b = A.random_key()
print(a,b)
P = A.encoding("Pesan Baru")
print(P)
C = A. enciphering(a,b,P)
print(C)
print(A.deciphering(a,b,C) == P)

```

Untuk memperoleh hasil enkripsi dan dekripsi dari kunci yang dibangkitkan secara acak ini, kita dapat menekan tombol `Ctrl+Enter` pada *keyboard* sehingga diperoleh:

```

17 9
PESANBARU
EZDJWAJML
True

```

Kunci rahasia yang didapatkan pada setiap kali *run* dapat berbeda-beda karena nilainya diacak oleh sistem.

SageMath juga mendukung kriptanalisis untuk memecahkan nilai kunci rahasia dari *ciphertext* sehingga nilai *plaintext* dapat diketahui. Metode yang didukung oleh SageMath adalah *brute-force* dengan mencoba semua kemungkinan nilai kunci rahasia. Terdapat tiga pembagian fungsi pemeringkatan (*ranking*) yang didukung oleh SageMath yaitu:

1. "none" merupakan pemeringkatan *default*, artinya tidak menggunakan fungsi pemeringkatan apapun.
2. "chi\_square", pada pemeringkatan ini hasil yang didapat akan diurutkan berdasarkan nilai *chi-square* yang didapat dari metode `rank_by_chi_square`. Misalkan kita mengetahui
  - (a)  $A$  : himpunan alfabet yang tidak kosong yang memuat  $n$  anggota;
  - (b)  $n$  : banyaknya elemen dari himpunan alfabet  $A$ ;

- 
- (c)  $M$  : kandidat nilai dekripsi dari teks sandi  $C$  menggunakan percobaan kunci  $(a, b)$ , dalam hal ini  $(a, b)$  bisa saja tidak digunakan pada proses enkripsi;
- (d)  $L$  : panjang karakter (banyaknya karakter) dari  $M$ ;
- (e)  $F_A(e)$  : probabilitas frekuensi karakteristik untuk  $e \in A$  (*expected probability*);
- (f)  $F_M(e)$  : probabilitas frekuensi karakteristik  $e \in M$  atau distribusi dari rasio kemunculan karakter pada pesan (*observed probability*);
- (g) Frekuensi yang diamati (*observed frequency*),  $O_M(e)$ , yang memenuhi persamaan  $O_M(e) = F_M(e) \cdot L$ ;
- (h) Frekuensi yang diharapkan (*expected frequency*),  $E_A(e)$ , yang memenuhi persamaan  $E_A(e) = F_A(e) \cdot L$ ;

maka kita memperoleh peringkat *chi square* dari  $M$  untuk kunci  $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_n$  adalah

$$R_{\chi^2}(M) = \sum_{e \in A} \frac{(O_M(e) - E_A(e))^2}{E_A(e)}.$$

Jika terdapat sejumlah  $k$  pasangan kunci  $(a, b)$  maka terdapat  $D = M_{a_1, b_1}, M_{a_2, b_2}, \dots, M_{a_k, b_k}$  kandidat teks pesan (*plaintext*) dari sebuah teks sandi (*ciphertext*) yang diurutkan berdasarkan  $R_{\chi^2}(M_{a_i, b_i})$  secara menurun.

3. "squared\_differences", pada pemeringkatan ini hasil yang didapat akan diurutkan berdasarkan nilai *residual sum of squares* dari metode `rank_by_squared_differences`. Pemeringkatan *squared differences* dari  $M$  untuk kunci  $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_n$  memenuhi

$$R_{RSS}(M) = \sum_{e \in A} (O_M(e) - E_A(e))^2,$$

dengan  $O_M(e)$  dan  $E_A(e)$  berturut-turut merupakan frekuensi yang diamati dan frekuensi yang diharapkan dari suatu karakter  $e \in A$  dan  $e \in M$  sebagaimana dijelaskan pada pemeringkatan dengan metode *chi-square*.

Contoh penerapan kriptanalisis untuk sandi *affine* dapat dilihat pada skrip berikut:

```
A = AffineCryptosystem(AlphabeticStrings())
a, b = (3, 19)
P = A.encoding("Pesanan Asli")
print(P)
C = A.enciphering(a, b, P)
print(C)
BF = A.brute_force(C)
print('Hasil kriptanalisis brute-force: ')
print(sorted(BF.items())[:10])
print('Hasil kriptanalisis chi-square: ')
CS = A.brute_force(C, ranking="chisquare")
print(CS[:10])
print('Hasil kriptanalisis squared-difference: ')
SD = A.brute_force(C, ranking="squared_differences")
print(SD[:10])
```

Pada skrip ini kita melakukan enkripsi terhadap sebuah string PESANASLI menggunakan sandi *affine* dengan kunci  $(3, 19)$ . Kita juga mengurutkan hasil kriptanalisis dan mengambil

---

10 nilai terbesar dari masing-masing metode kriptanalisis yang dipakai (yaitu metode *brute-force*, pemeringkatan menggunakan *chi-square*, dan pemeringkatan menggunakan *squared-differences*). Hasil kriptanalisis pada SageMath adalah sebagai berikut:

```
PESANASLI
MFVTGTVAR
Hasil kriptanalisis brute-force:
[((1, 0), MFVTGTVAR), ((1, 1), LEUSFSUZQ),
((1, 2), KDTRERTYP), ((1, 3), JCSQDQSXO),
((1, 4), IBRPCPRWN), ((1, 5), HAQOBOQVM),
((1, 6), GZPNANPUL), ((1, 7), FYOMZMOTK),
((1, 8), EXNLYLNSJ), ((1, 9), DWMKXXKMRI)]
Hasil kriptanalisis chi-square:
[((17, 6), IDHNNANHST), ((7, 17), DCIEREIFA),
((3, 19), PESANASLI), ((19, 21), FGAEREADI),
((23, 19), LWIANAIPS), ((11, 21), LIAOBOARC),
((19, 15), TUOSFSORW), ((11, 14), OLDRERDUF),
((23, 0), WHTLYLTAD), ((21, 13), VMOEREONU)]
Hasil kriptanalisis squared-difference:
[((19, 21), FGAEREADI), ((7, 17), DCIEREIFA),
((3, 19), PESANASLI), ((17, 6), IDHNNANHST),
((9, 11), DIEYLYETS), ((9, 6), SXTNANTIH),
((21, 13), VMOEREONU), ((23, 5), PAMEREMTW),
((3, 7), TIWEREWPM), ((17, 15), JEIOBOITU)]
```

Dari hasil ini kita melihat bahwa teks pesan PESANASLI dienkripsi menggunakan sandi *affine* dengan kunci (3, 19) dan menghasilkan teks sandi MFVTGTVAR. Pada metode *brute-force* teks pesan yang mungkin diurutkan berdasarkan kemungkinan kunci rahasianya, yaitu mulai dari kunci (1, 0) (yang menghasilkan teks pesan MFVTGTVAR), (1, 1) (yang menghasilkan teks pesan LEUSFSUZQ), hingga kunci (1, 9) (yang menghasilkan teks pesan DWMKXXKMRI). Pada kriptanalisis dengan metode *brute-force* teks pesan awal PESANASLI tidak termasuk hasil yang dikeluarkan pada 10 teks pesan yang mungkin. Kemudian untuk metode kriptanalisis dengan pemeringkatan *chi-square* maupun *squared-differences* kita melihat bahwa teks pesan awal PESANASLI berada di peringkat ketiga.

**Latihan 3.2.1** Lakukan enkripsi untuk string 'Indonesia jaya' menggunakan sandi *affine* dengan pasangan kunci  $(a, b) = (4, 3)$  dan  $(a, b) = (3, 9)$ .

1. Adakah pasangan kunci yang tidak bisa digunakan? Jika ada, mengapa?
2. Adakah pasangan kunci yang bisa digunakan? Jika ada, tentukan teks sandi (*ciphertext*) yang dihasilkan?

### 3.2.2 Caesar Cipher (Sandi Caesar atau Sandi Geser)

Caesar *cipher* atau sandi geser merupakan salah satu contoh dari *monoalphabetic substitution cipher*. Pada SageMath, implementasi Caesar *cipher* bisa diturunkan dari sandi *affine* atau menggunakan *class ShiftCryptosystem*.

Sebagaimana namanya, sandi Caesar atau sandi geser bekerja dengan menggeser karakter pada teks sandi ke kanan (atau ke kiri) beberapa posisi. Banyaknya pergeseran posisi yang dilakukan merupakan kunci rahasia dari sandi Caesar.<sup>2</sup>

---

<sup>2</sup>Menurut cerita, sandi geser seperti ini digunakan oleh Julius Caesar dalam mengirimkan pesan-pesan militer yang sifatnya rahasia.

---

Sandi Caesar hanya memiliki sebuah kunci rahasia  $b \in \mathbb{Z}_n$ . Jika diketahui  $p$  adalah teks pesan (*plaintext*) dengan  $p \in \mathbb{Z}_n$ , maka teks sandi (*ciphertext*)  $c$  dari nilai  $p$  tersebut didefinisikan sebagai

$$c \equiv (p + b) \pmod{n}. \quad (3.1)$$

Jika diketahui teks sandi (*ciphertext*)  $c \in \mathbb{Z}_n$  dan kunci rahasia  $b$ , maka nilai teks pesan (*plaintext*) dapat diketahui melalui kongruensi

$$p \equiv (c - b) \pmod{n}. \quad (3.2)$$

Pada SageMath domain karakter yang didukung untuk penerapan Caesar *cipher* secara *default* adalah `AlphabeticsStrings()`. Perhatikan bahwa **Caesar cipher adalah affine cipher dengan kunci rahasia  $(1, b)$ .** Salah satu penerapan Caesar *cipher* dapat dilihat pada skrip berikut:

```
Caesar = AffineCryptosystem(AlphabeticStrings())
P = Caesar.encoding("ini pesan rahasia")
print(P)
(a,b) = (1,7)
C = Caesar.enciphering(a,b,P)
print(C)
print(Caesar.deciphering(a,b,C)==P)
```

Untuk memperoleh hasil enkripsi dan dekripsi pada Caesar *cipher*, kita dapat menekan tombol `Ctrl+Enter` pada *keyboard* sehingga diperoleh:

```
INIPE SANRAHASIA
PUPWLZHUYHOHZPH
True
```

Perhatikan bahwa kunci  $(1, 7)$  yang digunakan berarti masing-masing huruf digeser sebanyak 7 posisi ke kanan. Misalnya huruf A digeser menjadi huruf H. Pada SageMath, proses enkripsi dan dekripsi untuk Caesar *cipher* dapat dilakukan dengan sintaks `ShiftCryptosystem(...)` sebagai berikut:

```
Caesar2 = ShiftCryptosystem(AlphabeticStrings())
P2 = Caesar2.encoding("ini pesan rahasia")
print(P2)
b2 = 7
C2 = Caesar2.enciphering(b2,P2)
print(C2)
print(Caesar2.deciphering(b2,C2)==P2)
```

Hasil dari penggunaan *class* `ShiftCryptosystem` akan tepat sama dengan keluaran dari penggunaan *class* `AffineCryptosystem`. Mengingat sandi Caesar merupakan kasus khusus dari sandi *affine*, maka semua metode kriptanalisis pada SageMath untuk sandi *affine* juga dapat diterapkan pada sandi Caesar.

**Latihan 3.2.2** Lakukan enkripsi terhadap pesan 'detik detik menit' menggunakan Caesar *cipher* dengan kunci  $k_1 = 31$  dan  $k_2 = 8$ .

1. Adakah nilai kunci yang tidak bisa digunakan? Jika ada, mengapa dan bagaimana salah satu solusi yang dapat digunakan supaya Caesar *cipher* bisa dijalankan?
2. Adakah nilai kunci yang bisa digunakan? Tentukan nilai teks sandi (*ciphertext*) yang didapat dan hitung distribusi frekuensi teks sandi (*ciphertext*) tersebut.
3. Jalankan simulasi kriptanalisis menggunakan metode *brute-force* dan pemeringkatan *chi-square*.

### 3.2.3 Decimation Cipher

Seperti *affine cipher* dan *shift cipher*, *decimation cipher* merupakan salah satu contoh dari *monoalphabetic substitution cipher*. Sebagaimana *shift cipher*, pada SageMath implementasi *decimation cipher* dapat diturunkan dari *affine cipher*.

*Decimation cipher* hanya memiliki sebuah kunci rahasia  $a \in \mathbb{Z}_n$ . Jika diketahui  $p$  adalah teks pesan (*plaintext*) dengan  $p \in \mathbb{Z}_n$ , maka teks sandi (*ciphertext*)  $c$  dari nilai  $p$  tersebut didefinisikan sebagai

$$c \equiv ap \pmod{n}.$$

Jika diketahui teks sandi (*ciphertext*)  $c \in \mathbb{Z}_n$  dan kunci rahasia  $a$ , maka nilai teks pesan (*plaintext*) dapat diketahui melalui

$$p \equiv a^{-1}c \pmod{n},$$

dengan  $a^{-1}$  adalah nilai invers dari  $a \pmod{n}$ . Dalam hal ini jelas bahwa nilai  $a$  haruslah memenuhi  $\gcd(a, n) = 1$ . Secara *default* domain karakter yang didukung oleh *decimation cipher* adalah `AlphabeticsStrings()`. Perhatikan bahwa *decimation cipher* adalah *affine cipher* dengan kunci  $(a, 0)$ . Contoh penerapan *decimation cipher* pada SageMath dapat dilakukan dalam skrip berikut:

```
Decimation = AffineCryptosystem(AlphabeticStrings())
P = Decimation.encoding("ini pesan rahasia")
print(P)
(a,b) = (3,0)
C = Decimation.enciphering(a,b,P)
print(C)
print(Decimation.deciphering(a,b,C)==P)
```

Pada skrip ini kita menggunakan fungsi  $f(x) = 3x \pmod{26}$  dalam melakukan enkripsi. Sebagai contoh huruf E yang berkorespondensi dengan bilangan 4 dienkripsi menjadi huruf L yang berkorespondensi dengan bilangan  $3 \cdot 4 = 12$ . Kemudian secara matematis, proses dekripsi dilakukan menggunakan fungsi  $g(x) = 3^{-1} \pmod{26} = 9x \pmod{26}$ . Perhatikan bahwa  $\gcd(3, 26) = 1$ . Untuk memperoleh hasil enkripsi dan dekripsi pada *decimation cipher*, kita dapat menekan tombol `Ctrl+Enter` pada *keyboard* sehingga diperoleh:

```
INIPE SANRAHASIA
YNYTMCAN ZAVACYA
True
```

Mengingat *decimation cipher* merupakan kasus khusus dari sandi *affine*, maka semua metode kriptanalisis pada SageMath untuk sandi *affine* juga dapat diterapkan pada *decimation cipher*.

**Latihan 3.2.3** Lakukan enkripsi terhadap pesan 'pelatihan hari Kamis' menggunakan *decimation cipher* dengan pasangan kunci  $(a_1, 0) = (2, 0)$  dan  $(a_2, 0) = (7, 0)$ .

1. Adakah nilai kunci yang tidak bisa digunakan? Jika ada, mengapa?
2. Adakah nilai kunci yang bisa digunakan? Tentukan teks sandi (*ciphertext*) yang didapat dan hitung kemunculan pada tiap karakter teks sandi tersebut.
3. Jalankan simulasi kriptanalisis menggunakan metode *brute-force* dan pemeringkatan *squared-differences*.

### 3.2.4 Hill Cipher (Sandi Hill)

Hill cipher merupakan salah satu contoh dari *polyalphabetic substitution cipher*. Sandi ini pertama kali diperkenalkan oleh Lester S. Hill pada tahun 1929 [3]. Pada SageMath, Hill

---

*cipher* berada pada `class sage.crypto.classical.HillCryptosystem`. Hill *cipher* menggunakan matriks persegi berukuran  $m \times m$  atas  $\mathbb{Z}_n$  untuk suatu  $n > 0$ . Dimensi baris dan kolom menentukan permutasi sebuah blok dari pesan yang disandikan.

Secara matematis sandi Hill adalah perumuman dari sandi *affine*. Misalkan kita ingin menyandikan sebuah pesan yang terdiri atas  $m$  karakter dalam alfabet huruf kapital menggunakan sandi Hill. Pesan tersebut dapat dinyatakan dalam sebuah vektor di  $\mathbb{Z}_{26}^m$ . Kunci pada sandi Hill adalah sebuah matriks  $\mathbf{K}$  yang memiliki invers pada  $M_{m \times m}(\mathbb{Z}_{26})$ .<sup>3</sup> Untuk meringkas penulisan himpunan seluruh matriks yang berukuran  $m \times m$  atas  $\mathbb{Z}_n$  dan memiliki invers dinotasikan dengan  $GL_m(\mathbb{Z}_n)$ . Pada sandi Hill baik teks pesan maupun teks sandi dinyatakan dalam vektor baris. Misalkan  $\mathbf{p} = [p_1, p_2, \dots, p_m]$  adalah suatu teks pesan, maka teks sandi  $\mathbf{c} = [c_1, c_2, \dots, c_m]$  yang merupakan hasil enkripsi dari  $\mathbf{p}$  dengan suatu kunci  $\mathbf{K} \in GL_m(\mathbb{Z}_{26})$  diperoleh melalui persamaan

$$\begin{aligned}\mathbf{c} &= \mathbf{p}\mathbf{K} \\ [c_1, c_2, \dots, c_m] &= [p_1, p_2, \dots, p_m]\mathbf{K}.\end{aligned}\tag{3.3}$$

Persamaan (3.3) pada dasarnya adalah enkripsi dari suatu pesan  $\mathbf{p}$  menggunakan kunci berupa matriks  $\mathbf{K}$ . Proses dekripsi dilakukan menggunakan persamaan

$$\begin{aligned}\mathbf{p} &= \mathbf{c}\mathbf{K}^{-1} \\ [p_1, p_2, \dots, p_m] &= [c_1, c_2, \dots, c_m]\mathbf{K}^{-1},\end{aligned}\tag{3.4}$$

perhatikan bahwa secara matematis persamaan (3.4) pada dasarnya identik dengan persamaan (3.3) ketika matriks  $\mathbf{K}$  memiliki invers. Untuk memperjelas, misalkan kita ingin menyandikan sebuah string OK menggunakan sandi Hill dengan kunci matriks  $\mathbf{K} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}$ . Perhatikan bahwa  $\det(\mathbf{K}) = -1 = 25$  sehingga  $\gcd(\det(\mathbf{K}), 26) = 1$ , akibatnya  $\mathbf{K}$  memiliki invers. Pertama kita akan mengubah matriks teks pesan OK ke dalam vektor  $\mathbf{p} = [14, 10]$ . Untuk memperoleh sandi  $\mathbf{c}$  kita melakukan operasi berikut

$$\begin{aligned}\mathbf{c} &= \mathbf{p}\mathbf{K} \\ &= [14, 10] \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \\ &= [14 + 10, 2 \cdot 14 + 10] \\ &= [24, 12],\end{aligned}$$

hasil dari  $\mathbf{c} = [24, 12]$  selanjutnya diubah menjadi string YM.

Untuk mengubah sandi YM ke dalam teks pesan, kita mengalikan  $\mathbf{c} = [24, 12]$  dengan  $\mathbf{K}^{-1} = \begin{bmatrix} 25 & 2 \\ 1 & 25 \end{bmatrix}$ , perhatikan bahwa

$$\begin{aligned}\mathbf{c}\mathbf{K}^{-1} &= [24, 12] \begin{bmatrix} 25 & 2 \\ 1 & 25 \end{bmatrix} \\ &= [24 \cdot 25 + 12 \cdot 1, 24 \cdot 2 + 12 \cdot 25] = [14, 10].\end{aligned}$$

Dengan mengubah  $[14, 10]$  ke dalam alfabet huruf kapital kita memperoleh string OK. Simulasi proses enkripsi dan dekripsi sederhana ini pada SageMath dapat dilakukan menggunakan skrip berikut:

---

<sup>3</sup>Sebuah matriks  $\mathbf{A} \in M_{m \times m}(\mathbb{Z}_n)$  memiliki invers jika dan hanya jika  $\gcd(\det(\mathbf{A}), n) = 1$ .

```

dom = AlphabeticStrings()
n = 2
HC = HillCryptosystem(dom, n)
print(HC)
print(HC.block_length())
R = Integers(26)
MS = MatrixSpace(R, n, n)
K = MS([[1,2],[1,1]]); print(K)
P = dom.encoding('OK'); print(P)
Enk = HC(K)
Dek = HC(K^-1)
print('Enkripsi cara pertama:')
C1 = Enk(P); print(C1)
print('Enkripsi cara kedua:')
C2 = HC.enciphering(K, P); print(C2)
print('Dekripsi cara pertama:')
D1 = Dek(C1); print(D1)
print('Dekripsi cara kedua:')
D2 = HC.deciphering(K, C2); print(D2)

```

Pada skrip ini kita menggunakan domain `AlphabeticStrings` seperti pada sandi *affine*. Kita menggunakan sandi Hill dengan ukuran matriks  $2 \times 2$ . Ukuran matriks selanjutnya juga diistilahkan dengan panjang blok (*block length*) pada sandi Hill. Misalkan pesan yang akan disandikan adalah `P`. Ada dua cara untuk melakukan enkripsi dengan sandi Hill pada SageMath. Pertama kita dapat menggunakan fungsi `Enk (P)` dengan `Enk=HC (K)`. Kedua kita dapat menggunakan perintah `HC.enciphering (K, P)`. Baik pada cara pertama maupun kedua `K` adalah matriks yang dijadikan sebagai kunci. Selanjutnya sebagaimana proses enkripsi, proses dekripsi juga dapat dilakukan dalam dua cara. Misalkan sandi yang akan didekripsi adalah `C`. Pertama kita dapat menggunakan fungsi `Dek (C)` dengan `Dek = HC (K^-1)`. Kedua kita dapat menggunakan perintah `HC.deciphering (K, C)`. Hasil dari simulasi ini pada SageMath adalah sebagai berikut:

```

Hill cryptosystem on Free alphabetic string monoid on A-Z of block length 2
2
[1 2]
[1 1]
OK
Enkripsi cara pertama:
YM
Enkripsi cara kedua:
YM
Dekripsi cara pertama:
OK
Dekripsi cara kedua:
OK

```

Perhatikan bahwa kunci rahasia pada sandi Hill bersifat simetris, yang berarti kunci yang sama digunakan baik untuk proses enkripsi maupun dekripsi. Kunci ini merupakan matriks persegi dengan ukuran baris dan kolom yang sama. Ukuran dari baris atau kolom yang juga disebut *block length* ini memengaruhi panjang dari pesan (banyaknya karakter pada pesan) yang dapat dienkripsi. Pada SageMath, sebuah pesan dapat dienkripsi menggunakan sandi Hill dengan kunci rahasia berupa matriks berukuran  $m \times m$  apabila banyaknya karakter pada pesan tersebut adalah kelipatan positif dari  $m$ . Secara matematis ini berarti banyaknya karakter dari pesan harus sama dengan  $\alpha m$  untuk suatu bilangan bulat positif  $\alpha$ . Dalam hal ini proses enkripsi dan dekripsi akan dilakukan pada  $\alpha$  buah blok pesan. Skrip berikut memberikan ilustrasi simulasi enkripsi Hill cipher pada pesan yang terdiri atas enam karakter menggunakan matriks yang berukuran  $3 \times 3$ .

```

dom = AlphabeticStrings()
n = 3
HC = HillCryptosystem(dom, n)
print(HC)
print(HC.block_length())

R = Integers(26)
M = MatrixSpace(R, n, n)
K = M([[1, 0, 2], [0, 1, 1], [2, 2, 3]])
print(K)

P = dom.encoding('OKELAH'); print(P)

print('Enkripsi cara pertama:')
Enk = HC(K);
C1 = Enk(P)
print(C1)
print('Enkripsi cara kedua:')
C2 = HC.enciphering(K, P)
print(C2)

```

Pada skrip tersebut matriks yang menjadi kunci adalah  $K = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 2 & 2 & 3 \end{bmatrix}$ . Perhatikan bahwa

teks pesan adalah OKELAH yang terdiri tepat atas enam karakter. Proses enkripsi dan dekripsi dilakukan dengan meninjau dua blok. Blok pertama adalah substring OKE dan blok kedua adalah substring LAH. Hasil simulasi dari skrip ini adalah:

```

Hill cryptosystem on Free alphabetic string monoid on A-Z of block length 3
3
[1 0 2]
[0 1 1]
[2 2 3]
OKELAH
Enkripsi cara pertama:
WSYZOR
Enkripsi cara kedua:
WSYZOR

```

Kita dapat memeriksa bahwa hasil dari enkripsi string OKE menggunakan kunci K adalah string WSY dan hasil dari enkripsi dari string LAH menggunakan kunci yang sama adalah ZOR.

Proses mengembalikan teks sandi (*ciphertext*) menjadi teks pesan (*plaintext*) dapat dilakukan menggunakan metode enkripsi dengan memanfaatkan `inverse_key` atau menggunakan metode dekripsi dengan kunci rahasia (yang bersifat simetris) sebagaimana dijelaskan secara teori pada persamaan (3.3) dan (3.4). Hal ini diilustrasikan pada skrip berikut:

```

Kin = HC.inverse_key(K)
Dek = HC(Kin)
print('Dekripsi cara pertama:')
P1 = Dek(C1)
print(P1)
print('Dekripsi cara kedua')
P2 = HC.deciphering(K, C1)
print(P2)

```

Hasil dari kedua proses dekripsi ini identik, yaitu:

Dekripsi cara pertama:

OKELAH

Dekripsi cara kedua:

OKELAH

Ukuran *block length* bisa dibuat secara acak dengan memanfaatkan fungsi `randint` dan ukuran maksimum dari alfabet `AlphabeticStrings().ngens()`. Selain itu kunci juga bisa didapatkan secara acak dari SageMath dengan memanfaatkan fungsi `random_key()` se-*suai* dengan ukuran blok yang didefinisikan pada Hill *cipher*. Hal ini diilustrasikan dalam skrip berikut:

```
dom = AlphabeticStrings()
n = 6
HC = HillCryptosystem(dom, n)
key = HC.random_key()
print(HC.block_length())
print(key)
Enk = HC(key)
P = dom.encoding('PESANRAHASIA');
print('Enkripsi cara pertama:')
C = Enk(P); print(C)
print('Enkripsi cara kedua:')
print(HC.enciphering(key, P))
print('Dekripsi cara pertama:')
print(HC.enciphering(HC.inverse_key(key), C))
print('Dekripsi cara kedua:')
print(HC.deciphering(key, C))
```

Salah satu keluaran yang mungkin dari simulasi ini adalah:

```
6
[18  8  25 20  5  0]
[16  3  10  0  5 19]
[ 2 15 25 22  6 20]
[14 18 15  6 10  0]
[15  6  1   7 23 18]
[15  6  1 22  7 12]
```

Enkripsi cara pertama:

OKLRXQQDKIJR

Enkripsi cara kedua:

OKLRXQQDKIJR

Dekripsi cara pertama:

PESANRAHASIA

Dekripsi cara kedua:

PESANRAHASIA

Perhatikan bahwa teks pesan, yaitu string PESANRAHASIA, terdiri dari 12 karakter dan *block length* yang digunakan berukuran 6.

**Latihan 3.2.4** Lakukan enkripsi pesan 'dont worry be happy' pada SageMath menggunakan sandi Hill dengan ukuran blok  $n = 4$  jika diketahui kunci berbentuk matriks berukuran  $4 \times 4$  berikut:

$$\mathbf{K}_1 = \begin{bmatrix} 1 & 0 & 2 & 3 \\ 0 & 1 & 1 & 5 \\ 2 & 2 & 3 & 8 \\ 1 & 9 & 3 & 8 \end{bmatrix}, \mathbf{K}_2 = \begin{bmatrix} 2 & 8 & 7 & 2 \\ 17 & 23 & 2 & 9 \\ 24 & 3 & 24 & 9 \\ 17 & 5 & 11 & 18 \end{bmatrix}.$$

- 
1. Adakah kunci di antara  $K_1$  atau  $K_2$  yang tidak bisa digunakan pada saat enkripsi? Jika ada, mengapa? Jika tidak, tentukan teks sandi (*ciphertext*) dari enkripsi dengan kedua kunci tersebut.
  2. Adakah nilai kunci yang tidak bisa digunakan pada saat dekripsi? Jika ada, mengapa?

### 3.2.5 Vigenère Cipher (Sandi Vigenère)

Vigenère *cipher* atau sandi Vigenère merupakan salah satu contoh lain dari *polyalphabetic substitution cipher* selain sandi Hill. Sandi ini dinamai berdasarkan nama pengusulnya yang hidup pada abad ke-16, yaitu Blaise de Vigenère [3]. Pada SageMath, Vigenère *cipher* berada pada class sage.crypto.classical.VigenereCryptosystem.

Teks pesan pada sandi Vigenère direpresentasikan dalam sebuah vektor di  $\mathbb{Z}_{26}^n$  untuk suatu bilangan bulat positif  $n$ . Teks sandi yang dihasilkan maupun kunci yang dipakai juga direpresentasikan sebagai vektor atas  $\mathbb{Z}_{26}$  dengan panjang yang sama. Cara kerja sandi Vigenère mirip dengan sandi geser yang telah dijelaskan pada Subtopik 3.2.2, namun penjumlahan yang dilakukan ditinjau pada modul  $\mathbb{Z}_{26}^n$ .

Secara matematis proses enkripsi dan dekripsi pada sandi Vigenère dijelaskan sebagai berikut. Misalkan  $\mathbf{p} = (p_1, p_2, \dots, p_n) \in \mathbb{Z}_{26}^n$  adalah teks pesan dan  $\mathbf{k} = (k_1, k_2, \dots, k_n) \in \mathbb{Z}_{26}^n$  adalah kunci yang digunakan. Enkripsi dari pesan  $\mathbf{p}$  menggunakan kunci  $\mathbf{k}$  menghasilkan sandi  $\mathbf{c}$  yang memenuhi hubungan

$$\begin{aligned}\mathbf{c} &= \mathbf{p} + \mathbf{k} \\ (c_1, c_2, \dots, c_n) &= (p_1, p_2, \dots, p_n) + (k_1, k_2, \dots, k_n) \\ &= (p_1 + k_1, p_2 + k_2, \dots, p_n + k_n).\end{aligned}\tag{3.5}$$

Semua operasi aritmetika dilakukan atas ring  $\mathbb{Z}_{26}$ . Perhatikan bahwa persamaan (3.5) analog dengan persamaan (3.1) yang digunakan pada proses enkripsi sandi geser. Dekripsi dari suatu sandi  $\mathbf{c}$  menggunakan kunci  $\mathbf{k}$  memberikan pesan  $\mathbf{p}$  yang memenuhi hubungan

$$\begin{aligned}\mathbf{p} &= \mathbf{c} - \mathbf{k} \\ (p_1, p_2, \dots, p_n) &= (c_1, c_2, \dots, c_n) - (k_1, k_2, \dots, k_n) \\ &= (c_1 - k_1, c_2 - k_2, \dots, c_n - k_n).\end{aligned}\tag{3.6}$$

Sebagaimana proses enkripsi, semua operasi aritmetika dilakukan atas ring  $\mathbb{Z}_{26}$ . Tinjau bahwa persamaan (3.6) analog dengan persamaan (3.2) yang digunakan pada proses dekripsi sandi geser.

Meksipun secara matematis panjang dari kunci pada sandi Vigenère harus sama dengan panjang dari pesan yang dienkripsi, sebenarnya kita dapat menggunakan kunci yang panjangnya jauh lebih pendek dari panjang pesan. Caranya adalah dengan “memperpanjang” kunci yang lebih pendek tersebut dengan menyalin dan mengulang entri-entrinya. Misalkan pesan yang dienkripsi adalah  $\mathbf{p} = (p_1, p_2, \dots, p_n)$  dan kunci awal yang panjangnya lebih pendek adalah  $\mathbf{k}_{awal} = (k_1, k_2, \dots, k_m)$  untuk suatu  $m < n$ . Kita dapat mendefinisikan kunci enkripsi  $\mathbf{k} = (k_1, k_2, \dots, k_m, k_1, k_2, \dots, k_\ell)$  untuk suatu  $\ell \leq n$ .

Untuk memperjelas, perhatikan contoh berikut. Misalkan kita ingin mengirim pesan dalam alfabet kapital AYOMAIN menggunakan sandi Vigenère dengan kunci OKE. Pertama kita perlu mengubah pesan ke dalam vektor atas  $\mathbb{Z}_{26}$ . Karena AYOMAIN memuat 7 karakter maka kita menggunakan vektor dengan panjang 7. Dengan korespondensi  $A \leftrightarrow 0, B \leftrightarrow 1, \dots$ , dan  $Z \leftrightarrow 25$  kita memperoleh pesan  $\mathbf{p} = (0, 24, 14, 12, 0, 8, 13)$  dan kunci awal  $\mathbf{k}_{awal} = (14, 10, 4)$ . Kunci awal ini dapat diperpanjang sehingga diperoleh kunci enkripsi  $\mathbf{k} = (14, 10, 4, 14, 10, 4, 14)$ . Pada proses enkripsi kita menghitung  $\mathbf{c} = \mathbf{p} + \mathbf{k}$  sehingga diperoleh  $\mathbf{c} = (14, 8, 18, 0, 10, 12, 1)$

---

(ingat bahwa operasi aritmetika dilakukan modulo 26). Dari hasil ini kita memperoleh teks sandi OISAKMB. Untuk mendekripsi teks sandi kita melakukan kalkulasi  $\mathbf{p} = \mathbf{c} - \mathbf{k}$  sehingga diperoleh  $\mathbf{p} = (0, 24, 14, 12, 0, 8, 13)$  yang berkorespondensi dengan string AYOMAIN.

Secara *default* domain karakter yang didukung dalam penerapan Vigenère *cipher* pada SageMath adalah `AlphabeticStrings()`. Skrip berikut memberikan ilustrasi dari penerapan Vigenère *cipher* dari contoh yang telah dijelaskan pada SageMath:

```
dom = AlphabeticStrings()
Vig = VigenereCryptosystem(dom, 3)
print(Vig)
print('Kunci yang digunakan:')
Key = dom.encoding('OKE')
print(Key)
print('Teks pesan:')
P = dom.encoding('AYOMAIN')
print(P)
print('Teks sandi:')
C = Vig.enciphering(Key, P)
print(C)
print('Pengujian kesamaan:')
print(Vig.deciphering(Key, C) == P)
print(Vig.enciphering(Vig.inverse_key(Key), C) == P)
```

Keluaran dari simulasi yang dijelaskan pada skrip ini adalah:

```
Vigenere cryptosystem on Free alphabetic string monoid on A-Z of period 3
Kunci yang digunakan:
OKE
Teks pesan:
AYOMAIN
Teks sandi:
OISAKMB
Pengujian kesamaan:
True
True
```

Nilai kunci juga bisa didapatkan secara acak menggunakan fungsi `random_key()` sebagaimana diilustrasikan pada skrip berikut:

```
dom = AlphabeticStrings()
Vig = VigenereCryptosystem(dom, 4)
print(Vig)
print('Kunci yang digunakan:')
Key = Vig.random_key()
print(Key)
print('Teks pesan:')
P = dom.encoding('AYOMAIN')
print(P)
print('Teks sandi:')
C = Vig.enciphering(Key, P)
print(C)
print('Pengujian kesamaan:')
print(Vig.deciphering(Key, C) == P)
print(Vig.enciphering(Vig.inverse_key(Key), C) == P)
```

Pada SageMath perintah `VigenereCryptosystem(dom, n)` berfungsi untuk memanggil *library* sandi Vigenère yang meninjau domain alfabet `dom` (biasanya secara *default* `dom = AlphabeticStrings()`) yang menggunakan kunci awal dengan  $n$  karakter. Pada skrip sebelumnya kita memanggil *library* sandi Vigenère yang meninjau domain string alfabet (huruf

---

kapital) dan kunci awal dengan 4 karakter. Salah satu keluaran yang mungkin dari skrip ini adalah:

```
Vigenere cryptosystem on Free alphabetic string monoid on A-Z of period 4
Kunci yang digunakan:
VVIO
Teks pesan:
AYOMAIN
Teks sandi:
VTWAVDV
Pengujian kesamaan:
True
True
```

Pada contoh ini kita menggunakan kunci awal VVIO. Karena teks pesan terdiri atas 7 karakter, maka kita dapat menganalogikan bahwa kunci yang digunakan untuk enkripsi adalah VVIOVVVI.

**Latihan 3.2.5** Misalkan terdapat dua jenis penerapan sandi Vigenère dengan panjang kunci awal  $n = 7$  dan dengan nilai kunci dan pesan yang berbeda sebagai berikut:

- sistem 1: kunci: ITSOKE, pesan: CHINESE REMAINDER THEOREM,
  - sistem 2: kunci: ITSOEURS, pesan: RAHASIA
1. Di antara sistem 1 dan sistem 2, sistem manakah yang tidak dapat menerapkan Vigenère cipher pada saat enkripsi? Jelaskan jawaban Anda. Kemudian untuk setiap sistem yang dapat melakukan enkripsi tentukan pesan sandi yang dihasilkan.
  2. Untuk setiap sistem yang dapat melakukan enkripsi, terapkan proses dekripsi dari pesan sandi yang dihasilkan.

### 3.3 *Transposition Cipher* (Sandi Transposisi)

*Transposition cipher* atau sandi transposisi merupakan suatu metode enkripsi klasik yang menukar posisi setiap unit teks pesan (*plaintext*) berdasarkan aturan atau algoritma tertentu. Sandi transposisi juga disebut sebagai sandi permutasi karena aturan yang digunakan adalah permutasi yang didefinisikan atas grup permutasi (atau grup simetri) tertentu.

Pada sandi transposisi teks sandi (*ciphertext*) yang dihasilkan merupakan permutasi dari teks pesan. Dengan perkataan lain teks sandi diperoleh dari teks pesan dengan cara menyusun ulang urutan unit karakter yang ada pada teks pesan. Secara matematis cara kerja sandi transposisi dijelaskan sebagai berikut. Teks pesan yang memuat  $n$  karakter (memiliki panjang  $n$ ) dipetakan ke dalam sebuah vektor pada  $\mathbb{Z}_{26}^n$ . Kunci dari sandi transposisi adalah sebuah elemen pada grup permutasi  $S_n$  yang dijelaskan pada Subtopik 2.2.2. Misalkan kita akan menyandikan pesan  $\mathbf{p} = (p_1, p_2, \dots, p_n)$  menggunakan kunci  $\alpha \in S_n$ . Hasil dari enkripsi  $\mathbf{p}$  dengan kunci  $\alpha$  adalah sebuah teks sandi  $\mathbf{c} = (c_1, c_2, \dots, c_n)$  yang memenuhi

$$(c_1, c_2, \dots, c_n) = (p_{\alpha(1)}, p_{\alpha(2)}, \dots, p_{\alpha(n)}).$$

Untuk memperoleh teks pesan  $\mathbf{p} = (p_1, p_2, \dots, p_n)$  dari suatu teks sandi  $\mathbf{c} = (c_1, c_2, \dots, c_n)$  kita perlu menentukan invers dari permutasi  $\alpha \in S_n$  terlebih dulu. Proses dekripsi teks pesan  $\mathbf{p}$  dari teks sandi  $\mathbf{c}$  dilakukan menggunakan kunci  $\alpha^{-1} \in S_n$  dan sesuai dengan formulasi

$$(p_1, p_2, \dots, p_n) = (c_{\alpha^{-1}(1)}, c_{\alpha^{-1}(2)}, \dots, c_{\alpha^{-1}(n)}).$$

Sebagai ilustrasi misalkan kita memiliki pesan HALO dan kunci  $\alpha = (2\ 3\ 4\ 1)$ . Perhatikan bahwa  $\alpha$  merupakan permutasi pada  $S_4$  dengan definisi  $\alpha(1) = 2$ ,  $\alpha(2) = 3$ ,  $\alpha(3) = 4$ , dan

---

$\alpha(4) = 1$ . Untuk melakukan enkripsi kita mengubah pesan HALO ke dalam vektor pada  $\mathbb{Z}_{26}^4$  sehingga diperoleh teks pesan  $(7, 0, 11, 14)$ . Hasil dari enkripsi adalah

$$\begin{aligned}\mathbf{c} &= (p_{\alpha(1)}, p_{\alpha(2)}, p_{\alpha(3)}, p_{\alpha(4)}) \\ &= (p_2, p_3, p_4, p_1) \\ &= (0, 11, 14, 7).\end{aligned}$$

Dengan mengubah vektor  $\mathbf{c}$  ke dalam string kita memperoleh string ALOH. Perhatikan bahwa proses pengubahan dari teks pesan ke dalam vektor atas  $\mathbb{Z}_{26}$  sebenarnya tidak penting karena pada dasarnya teks sandi adalah permutasi dari karakter-karakter teks pesan.

Pada SageMath library yang digunakan dalam sandi transposisi terdapat dalam kelas

```
sage.crypto.classical.TranspositionCryptosystem.
```

Untuk mengenkripsi pesan yang memuat  $n$  karakter, kunci yang digunakan dapat berupa *list* yang isinya merupakan permutasi dari elemen-elemen pada himpunan  $\{1, 2, \dots, n\}$ . Kunci dapat dinyatakan sebagai elemen pada  $S_n$  yang ditulis dalam bentuk *bottom row format* (lihat Subtopik 2.2.2). Ilustrasi dari enkripsi pesan HALO menggunakan kunci  $\alpha = (2\ 3\ 4\ 1)$  yang telah dijelaskan sebelumnya dapat dilihat pada skrip berikut:

```
dom = AlphabeticStrings()
n = 4
trans = TranspositionCryptosystem(dom, n)
print(trans)
print('Kunci yang dipakai:')
Sn = SymmetricGroup(n)
K = [2,3,4,1];
Key = Sn(K);
print(Key)
print('Teks pesan:')
P = dom.encoding('HALO')
print(P)
print('Hasil enkripsi:')
C = trans.enciphering(Key, P)
print(C)
```

Hasil dari simulasi ini adalah:

```
Transposition cryptosystem on Free alphabetic string monoid on A-Z of block length 4
Kunci yang dipakai:
(1, 2, 3, 4)
Teks pesan:
HALO
Hasil enkripsi:
ALOH
```

Permutasi  $\alpha = (2\ 3\ 4\ 1)$  juga dapat dituliskan sebagai  $(1\ 2\ 3\ 4)$ . Perhatikan bahwa pada dasarnya proses dekripsi adalah enkripsi dari teks sandi dengan kunci  $\alpha^{-1}$  apabila  $\alpha$  adalah kunci yang digunakan pada saat enkripsi. Oleh karena itu dekripsi dapat dilakukan menggunakan dua cara sebagaimana dijelaskan pada contoh berikut:

---

```

print('Dekripsi cara pertama:')
D1 = trans.deciphering(Key,C)
print(D1)
print('Dekripsi cara kedua: ')
print('Kunci dekripsi:')
Keydec = trans.inverse_key(Key)
print(Keydec)
print('Hasil dekripsi:')
D2 = trans.enciphering(Keydec,C)
print(D2)

```

Hasil dari simulasi dekripsi ini adalah:

```

Dekripsi cara pertama:
HALO
Dekripsi cara kedua:
Kunci dekripsi:
(1, 4, 3, 2)
Hasil dekripsi:
HALO

```

Perhatikan bahwa jika  $\alpha = (1\ 2\ 3\ 4)$  maka  $\alpha^{-1} = (4\ 3\ 2\ 1)$ . Hal ini juga dapat dituliskan sebagai  $(1\ 4\ 3\ 2)$ . Kunci yang digunakan pada sandi transposisi juga dapat diperoleh secara acak dari sistem sebagaimana diilustrasikan pada skrip berikut:

```

dom = AlphabeticStrings()
n = 5
trans = TranspositionCryptosystem(dom,n)
print(trans)
print('Kunci yang digunakan')
K = trans.random_key()
print(K)
print('Teks pesan:')
P = dom.encoding('PESAN')
print(P)
print('Hasil enkripsi:')
C = trans.enciphering(K,P)
print(C)
print('Dekripsi cara pertama:')
D1 = trans.deciphering(K,C)
print(D1)
print('Dekripsi cara kedua:')
print('Kunci dekripsi:')
K2 = trans.inverse_key(K)
print(K2)
print('Hasil dekripsi:')
D2 = trans.enciphering(K2,C)
print(D2)

```

Pada skrip ini kita menggunakan sandi transposisi untuk menyandikan sebuah string PESAN yang terdiri atas lima karakter. Kunci yang digunakan diambil secara acak dari elemen pada  $S_5$ . Perhatikan bahwa dekripsi dapat dilakukan dengan dua cara, yaitu menggunakan perintah `trans.deciphering(K, C)` dengan  $K$  adalah kunci yang digunakan pada enkripsi dan  $C$  adalah teks sandi, atau menggunakan perintah `trans.enciphering(K2, C)` dengan  $K2$  adalah invers dari  $K$  (dalam hal ini baik  $K$  maupun  $K2$  ditinjau sebagai elemen pada  $S_5$ ).

---

Salah satu hasil dari simulasi ini adalah:

```
Transposition cryptosystem on Free alphabetic string monoid on A-Z of block length 5
Kunci yang digunakan
(1,4)(2,5,3)
Teks pesan:
PESAN
Hasil enkripsi:
ANEPS
Dekripsi cara pertama:
PESAN
Dekripsi cara kedua:
Kunci dekripsi:
(1,4)(2,3,5)
Hasil dekripsi:
PESAN
```

Pada simulasi ini SageMath menggunakan kunci  $\alpha = (1\ 4)(2\ 5\ 3)$  yang dapat ditinjau sebagai fungsi dari  $\{1, 2, 3, 4, 5\}$  ke  $\{1, 2, 3, 4, 5\}$  dengan definisi  $\alpha(1) = 4$ ,  $\alpha(2) = 5$ ,  $\alpha(3) = 2$ ,  $\alpha(4) = 1$ , dan  $\alpha(5) = 3$ . Akibatnya teks pesan PESAN dienkripsi menjadi ANEPS. Perhatikan bahwa  $\alpha^{-1} = (4\ 1)(3\ 5\ 2) = (1\ 4)(2\ 3\ 5)$ . Satu hal penting yang harus diingat pada *transposition cipher* ketika kita menggunakan domain `AlphabeticStrings` adalah bahwa **panjang *plaintext* setelah di-*encode* ke dalam `AlphabeticStrings` harus sama dengan panjang kunci**.

**Latihan 3.3.1** Misalkan pesan berupa string SAGEMATHISEASY akan dienkripsi menggunakan *transposition cipher* menggunakan kunci [14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1].

1. Apakah proses enkripsi terhadap string yang dimaksud dapat dijalankan? Jika ya, jelaskan *ciphertext* yang dihasilkan.
2. Apakah proses dekripsi terhadap teks sandi dapat dijalankan? Jika tidak, apakah alasananya? Jelaskan kunci yang digunakan untuk proses dekripsi.

# Topik 4

## Contoh Kriptografi Simetrik pada SageMath

Sistemkripto simetrik merupakan sistemkripto yang menggunakan kunci yang sama pada algoritma enkripsi maupun dekripsinya. Ada banyak jenis sistemkripto simetrik yang dipakai di dunia. Dua yang cukup terkenal adalah Standar Enkripsi Data (*Data Encryption Standard*, DES) dan Standar Enkripsi Lanjut (*Advanced Encryption Standard*, AES). Pada tutorial ini kita tidak akan mengkaji DES maupun AES dalam bentuk aslinya, namun kita akan meninjau penyederhanaan dari kedua sistemkripto tersebut untuk keperluan edukasi dan pengajaran. Kita akan meninjau sistemkripto DES yang disederhanakan atau *simplified DES* dan mini AES.

### 4.1 DES yang Disederhanakan (*Simplified DES*)

Pada tahun 1975, Biro Standar Nasional Amerika Serikat (yang kemudian menjadi Institut Nasional Standar dan Teknologi, *National Institute of Standard and Technology* (NIST)) menge luarkan *Data Encryption Standard* (DES) yang menjadi standar nasional enkripsi data (secara simetrik) di Amerika Serikat. Sejak saat itu DES banyak dipakai di industri perbankan. Misalkan ada dua bank yang ingin saling bertukar data secara rahasia, pertama kedua bank tersebut memakai sistemkripto asimetrik atau protokol pertukaran kunci untuk mengirimkan kunci yang dipakai untuk DES (alih-alih menggunakan sistemkripto asimetrik secara langsung) [4, Subbab 7. 1].

DES merupakan salah satu contoh sandi blok (*block cipher*) yang membagi pesan (*plaintext*) ke dalam blok 64 bit dan melakukan enkripsi secara terpisah pada masing-masing blok. Pada tutorial ini kita tidak akan membahas DES secara lengkap, namun kita akan membahas DES yang disederhanakan (*simplified DES*) yang memiliki karakteristik yang sama dengan DES, tetapi cukup kecil untuk dapat dijadikan contoh. Lebih jauh, teori DES yang disederhanakan pada tutorial ini diambil dari deskripsi yang diberikan oleh Trappe dan Washington [4, Bab 7.2]. Kita akan meringkas DES yang dimaksud dengan SDES.

**Catatan 4.1.1** *DES yang disederhanakan dibuat untuk keperluan edukasi/pengajaran saja dan tidak digunakan untuk keperluan pengembangan sistem. Pembaca yang tertarik mengkaji DES (bukan DES yang disederhanakan) dapat membaca buku-buku kriptografi seperti [4, Subbab 7.4]. Penerapan/simulasi DES pada SageMath juga dapat dilihat pada [1, Cryptography].*

Pada SDES, pesan yang kita tinjau (secara *default*) hanya terdiri dari satu blok saja. Pesan terdiri dari 12 bit dan ditulis dalam bentuk  $L_0R_0$ , dengan  $L_0$  merupakan 6 bit pertama dari pesan dan  $R_0$  adalah 6 bit terakhir dari pesan tersebut. SDES menggunakan kunci  $K$  yang berukuran 9 bit. Sebagaimana DES, SDES dapat terdiri dari beberapa putaran (*round*). Pada putaran ke- $i$

---

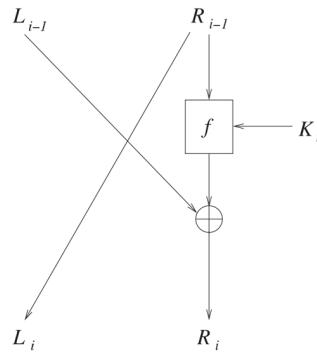
algoritma SDES menggunakan masukan berupa blok  $L_{i-1}R_{i-1}$  dan kunci  $K_i$  yang berukuran 8 bit yang diperoleh dari kunci  $K$  untuk memperoleh hasil  $L_iR_i$ .

Proses utama dari enkripsi adalah kalkulasi fungsi  $f(R_{i-1}, K_i)$  yang mengambil masukan  $R_{i-1}$  yang berukuran 6 bit dan kunci  $K_i$  yang berukuran 8 bit, serta menghasilkan keluaran yang berukuran 6 bit. Secara matematis keluaran dari putaran ke- $i$  adalah:

$$L_i = R_{i-1} \text{ dan } R_i = L_{i-1} \oplus f(R_{i-1}, K_i), \quad (4.1)$$

dengan  $\oplus$  merupakan operator *xor* atau penjumlahan dalam modulo 2.<sup>1</sup> Sebuah putaran dari SDES yang juga diistilahkan dengan sistem Feistel<sup>2</sup> dijelaskan pada Gambar 4.1.

Figure 7.1 One Round of a Feistel System



Gambar 4.1: Ilustrasi satu putaran sistem Feistel yang digunakan pada SDES. Gambar diambil dari [4, Gambar 7.1].

Operasi yang dijelaskan pada (4.1) dan Gambar 4.1 dilakukan sebanyak beberapa kali (berapa putaran) untuk menghasilkan teks sandi (*ciphertext*)  $L_nR_n$ . Kita akan membahas fungsi  $f(R_{i-1}, K_i)$  setelah kita mengkaji proses dekripsi dari SDES.

Misalkan diperoleh teks sandi  $L_nR_n$ . Untuk memperoleh *plaintext*  $L_0R_0$  hal pertama yang harus kita lakukan adalah menukar posisi  $L_n$  dan  $R_n$  (disebut dengan *switch*) sehingga diperoleh string  $R_nL_n$ . Selanjutnya kita menggunakan prosedur yang dijelaskan pada enkripsi namun dengan urutan kunci  $K_n, K_{n-1}, \dots, K_1$ , dengan  $n$  adalah banyaknya putaran yang dilakukan ketika enkripsi dilakukan.

Perhatikan bahwa jika masukan pada langkah pertama adalah  $R_nL_n$  dengan kunci  $K_n$  maka keluaran dari langkah ini adalah sebuah blok dengan bentuk

$$[L_n][R_n \oplus f(L_n, K_n)]. \quad (4.2)$$

Pada prosedur enkripsi kita memiliki kondisi (4.1) yang berarti  $L_n = R_{n-1}$  dan  $R_n = L_{n-1} \oplus$

<sup>1</sup>Kita memiliki  $x \oplus y = 1$  jika dan hanya jika  $x \neq y$  untuk  $x, y \in \{0, 1\}$ .

<sup>2</sup>Diambil dari istilah yang sama pada DES. Pada DES sistem ini diambil dari nama Horst Feistel yang mengembangkan LUCIFER yang merupakan cikal bakal DES.

---

$f(R_{n-1}, K_n)$ . Akibatnya blok pada (4.2) dapat ditulis sebagai

$$\begin{aligned}
 [L_n][R_n \oplus f(L_n, K_n)] &= [R_{n-1}][(L_{n-1} \oplus f(R_{n-1}, K_n)) \oplus f(L_n, K_n)] \\
 &= [R_{n-1}][(L_{n-1} \oplus f(R_{n-1}, K_n)) \oplus f(R_{n-1}, K_n)] \\
 &\quad \text{karena } L_n = R_{n-1} \text{ (lihat (4.1) dan Gambar 4.1)} \\
 &= [R_{n-1}][L_{n-1} \oplus (f(R_{n-1}, K_n) \oplus f(R_{n-1}, K_n))] \\
 &\quad \text{sifat asosiatif dari } \oplus \\
 &= [R_{n-1}][L_{n-1} \oplus 0] \\
 &\quad \text{karena } f(R_{n-1}, K_n) \oplus f(R_{n-1}, K_n) = 0 \\
 &= [R_{n-1}][L_{n-1}].
 \end{aligned} \tag{4.3}$$

Kondisi (4.3) menjelaskan bahwa hasil dekripsi putaran pertama dengan masukan  $R_n L_n$  dan kunci  $K_n$  adalah  $R_{n-1} L_{n-1}$ . Dengan cara yang serupa kita dapat memperoleh bahwa hasil putaran kedua dari dekripsi dengan masukan  $R_{n-1} L_{n-1}$  dan kunci  $K_{n-1}$  adalah  $R_{n-2} L_{n-2}$ . Perhatikan bahwa pada setiap akhir dekripsi putaran ke- $i$  kita mendapatkan  $R_{n-i} L_{n-i}$ . Melalui induksi matematika kita dapat memperoleh bahwa hasil proses dekripsi pada putaran ke- $n$  dengan masukan  $R_1 L_1$  dan kunci  $K_1$  adalah  $R_0 L_0$ . Untuk memperoleh *plaintext* kita cukup melakukan *switch* dari  $R_0 L_0$  sehingga diperoleh  $L_0 R_0$ .

Perhatikan bahwa proses dekripsi identik dengan proses enkripsi, kecuali pada proses penukaran blok (*switch*) dan urutan kunci yang digunakan. Ini berarti pada SDES pengirim maupun penerima memakai kunci yang sama dan mereka dapat memakai mesin (atau program) yang identik (meskipun penerima tetap harus melakukan *switch* untuk memperoleh *plaintext* yang diinginkan). Dengan demikian SDES adalah salah satu jenis sistemkripto simetrik.

Pada penjelasan yang telah diberikan kita melihat bahwa fungsi  $f$  pada (4.1) maupun Gambar 4.1 dapat berupa fungsi apapun (selama domain dan kodomain dari fungsi tersebut sesuai). Namun beberapa pilihan untuk  $f$  memberikan keamanan yang lebih dibandingkan dengan pilihan  $f$  yang lain. Pada tutorial ini kita akan membahas fungsi  $f$  yang skema komputasinya dijelaskan pada Gambar 4.2. Fungsi  $f$  menerima dua masukan pada putaran ke- $i$ , yaitu  $R_{i-1}$  dan  $K_i$ . Fungsi ini juga memuat dua fungsi lain, yaitu fungsi pengembang (*expander function*)  $E$  dan dua buah *S-box* (yang dinotasikan dengan  $S_1$  dan  $S_2$ ) yang akan dijelaskan selanjutnya.

Fungsi pengembang mengonversi string biner 6 bit menjadi 8 bit. Salah satu definisi matematis dari fungsi ini adalah

$$\begin{aligned}
 E : \mathbb{F}_2^6 &\rightarrow \mathbb{F}_2^8 \\
 (a_1, a_2, a_3, a_4, a_5, a_6) &\mapsto (a_1, a_2, a_4, a_3, a_4, a_3, a_5, a_6)
 \end{aligned} \tag{4.4}$$

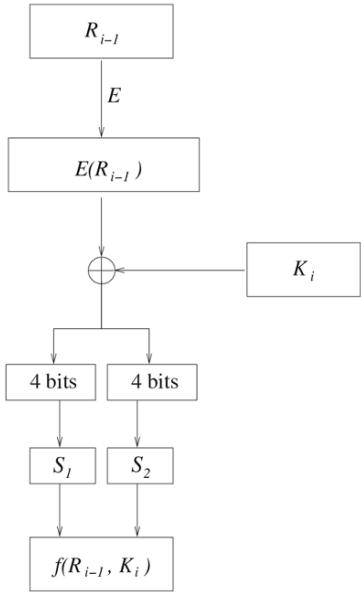
Fungsi pada (4.4) dapat diilustrasikan pada Gambar 4.3. Sebagai contoh kita memiliki  $E(0, 1, 1, 0, 0, 1) = (0, 1, 0, 1, 0, 1, 0, 1)$ . Untuk meringkas penulisan selanjutnya kita akan menuliskan elemen-elemen di  $\mathbb{F}_2^k$  tanpa menggunakan koma, misalnya  $(0, 1, 1, 0, 0, 1) \in \mathbb{F}_2^6$  ditulis  $(011001) \in \mathbb{F}_2^6$ .

Ada dua jenis *S-box* yang dipakai pada SDES pada [4, Subbab 7.2], yaitu

$$S_1 = \begin{bmatrix} 101 & 010 & 001 & 110 & 011 & 100 & 111 & 000 \\ 001 & 100 & 110 & 010 & 000 & 111 & 101 & 011 \end{bmatrix}, \tag{4.5}$$

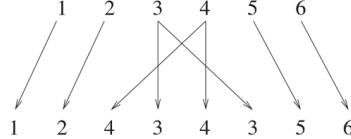
$$S_2 = \begin{bmatrix} 100 & 000 & 110 & 101 & 111 & 001 & 011 & 010 \\ 101 & 011 & 000 & 111 & 110 & 010 & 001 & 100 \end{bmatrix}. \tag{4.6}$$

Pada dasarnya sebuah *S-box* menerima masukan berupa string biner 4 bit  $b_0 b_1 b_2 b_3$ . Nilai  $b_0$  menentukan baris yang akan digunakan, kita memakai baris pertama jika  $b_0 = 0$  dan baris kedua



Gambar 4.2: Ilustrasi komputasi fungsi  $f(R_{i-1}, K_i)$  pada (4.1). Gambar diambil dari [4, Gambar 7.3].

Figure 7.2 The Expander Function



Gambar 4.3: Ilustrasi fungsi pengembang (*expander function*) yang digunakan pada definisi fungsi  $f$ . Gambar diambil dari [4, Gambar 7.2].

jika  $b_1 = 1$ . Tiga bit berikutnya, yaitu  $b_1 b_2 b_3$ , menentukan kolom pada  $S$ -box yang dipakai. Kolom ke-1 dipilih jika  $b_1 b_2 b_3 = 000$ , kolom ke-2 dipilih jika  $b_1 b_2 b_3 = 001$ , dan seterusnya hingga kolom ke-8 dipilih jika  $b_1 b_2 b_3 = 111$ . Dengan perkataan lain kolom ke- $i$  dipilih apabila nilai desimal dari  $b_1 b_2 b_3$  sama dengan  $i - 1$ . Sebagai contoh, jika string biner 4 bit yang menjadi masukan adalah  $b_0 b_1 b_2 b_3 = 1010$  dan kita menggunakan  $S_1$ , maka keluaran dari  $S_1$  kita peroleh dengan melihat baris ke-2 (karena  $b_0 = 1$ ) dan kolom ke-3 (karena representasi desimal dari 010 adalah 2 dan  $2 = 3 - 1$ ). Jadi keluaran dari 1010 menggunakan  $S_1$  adalah string biner 3 bit pada  $S_1$  yang ada pada baris ke-2 dan kolom ke-3 dari nilai  $S_1$  pada (4.5), yaitu 110.

Pada SDES yang dijelaskan di [4, Subbab 7.2] kunci  $K$  yang digunakan berukuran 9 bit. Kunci pada putaran ke- $i$ , yaitu  $K_i$ , diperoleh dari 8 bit dari  $K$  yang dibaca mulai bit ke- $i$  hingga 7 bit berikutnya di sebelah kanan. Konstruksi dilakukan secara siklis jika perlu. Misalnya jika  $K = 010011001$ , maka  $K_4 = 01100101$ .

Berdasarkan deskripsi yang telah diberikan, komputasi  $f(R_{i-1}, K_i)$  pada SDES yang diilustrasikan pada Gambar (4.2) dapat dirangkum sebagai berikut:

1. Masukan dari fungsi  $f$  adalah  $R_{i-1}$  yang terdiri dari 6 bit dan  $K_i$  yang memuat 8 bit karakter pada  $K$  dengan aturan yang telah dijelaskan sebelumnya.
2. String  $R_{i-1}$  yang ukurannya 6 bit dikembangkan dengan fungsi pengembang pada (4.4) menjadi  $E(R_{i-1})$  yang ukurannya 8 bit.
3. Proses selanjutnya adalah kalkulasi  $E(R_{i-1}) \oplus K_i$ , hasil dari kalkulasi ini dipisah menjadi dua bagian yang masing-masing berukuran 4 bit.

- 
4. Masing-masing bagian dengan ukuran 4 bit menjadi masukan untuk *S-box* yang bersesuaian. Contoh *S-box* yang digunakan dijelaskan pada (4.5) dan (4.6).
  5. Hasil dari komputasi dengan  $S_1$  dan  $S_2$  masing-masing adalah string 3 bit, misalkan hasil ini berturut-turut adalah  $B_1$  dan  $B_2$ . Hasil dari  $f(R_{i-1}, K_i)$  adalah  $B_1B_2$  (hasil perangkaian atau *concatenation* dari  $B_1$  dan  $B_2$ ) yang berukuran 6 bit.

Pada penjelasan berikut kita akan mengkaji sebuah contoh sederhana dari enkripsi dan dekripsi menggunakan SDES. Deskripsi pada contoh berikut diambil dari [16]. Misalkan kita memiliki pesan (*plaintext*)  $P = 100010110101$  dan kunci yang digunakan adalah  $K = 111000111$ . Misalkan enkripsi dilakukan dalam dua putaran (*two rounds*). Proses enkripsi dilakukan sebagai berikut:

1. Pada putaran pertama kita memiliki  $i = 0$ . Untuk  $P = 100010110101$  kita tuliskan  $L_0 = 100010$  dan  $R_0 = 110101$ . Kemudian  $K_1 = 11100011$ .
2. Kita memiliki  $L_1 = R_0 = 110101$  dan  $R_1 = L_0 \oplus f(R_0, K_1)$ . Kalkulasi  $f(R_0, K_1)$  dilakukan dengan langkah-langkah berikut:
  - (a) Kita hitung  $E(R_0) = E(110101) = 11101001$  dan  $E(R_0) \oplus K_1 = 11101001 \oplus 11100011 = 00001010$ .
  - (b) Hasil dari  $E(R_0) \oplus K_1$  dibagi menjadi dua bagian, yaitu 0000 dan 1010.
  - (c) Kita memiliki  $S_1(0000) = 101$  berdasarkan (4.5) dan  $S_2(1010) = 000$  berdasarkan (4.6). Selanjutnya hasil dari kedua *S-box* ini digabungkan sehingga diperoleh 101000.
3. Setelah diperoleh  $f(R_0, K_1) = 101000$  kita hitung  $R_1 = L_0 \oplus f(R_0, K_1) = 100010 \oplus 101000 = 001010$ .
4. Hasil dari putaran pertama adalah  $L_1 = 110101$  dan  $R_1 = 001010$ .
5. Selanjutnya pada putaran kedua kita memiliki  $i = 1$ ,  $L_1 = 110101$ , dan  $R_1 = 001010$ . Kita menggunakan kunci  $K_2 = 11000111$ .
6. Kita memiliki  $L_2 = R_1 = 001010$  dan  $R_2 = L_1 \oplus f(R_1, K_2)$ . Kalkulasi  $f(R_1, K_2)$  dilakukan dengan langkah-langkah berikut:
  - (a) Kita hitung  $E(R_1) = E(001010) = 00010110$  dan  $E(R_1) \oplus K_2 = 00010110 \oplus 11000111 = 11010001$ .
  - (b) Hasil dari  $E(R_1) \oplus K_2$  dibagi menjadi dua bagian, yaitu 1101 dan 0001.
  - (c) Kita memiliki  $S_1(1101) = 111$  berdasarkan (4.5) dan  $S_2(0001) = 000$  berdasarkan (4.6). Selanjutnya hasil dari kedua *S-box* ini digabungkan sehingga diperoleh 111000.
7. Setelah diperoleh  $f(R_1, K_2) = 111000$  kita hitung  $R_2 = L_1 \oplus f(R_1, K_2) = 110101 \oplus 111000 = 001101$ .
8. Hasil dari putaran kedua adalah  $L_2 = 001010$  dan  $R_2 = 001101$ .
9. Kita memperoleh teks sandi  $C = L_2R_2 = 001010001101$ .

Proses dekripsi dilakukan dengan langkah-langkah berikut:

- 
1. Kita mengetahui bahwa banyaknya putaran adalah 2 dan  $C = 001010001101$ .
  2. Pada putaran pertama kita tuliskan  $L_0 = 001010$  dan  $R_0 = 001101$ . Lakukan penukaran (*switch*) sehingga diperoleh  $L_0 = 001101$  dan  $R_0 = 001010$ . Kunci yang digunakan adalah  $K_2 = 11000111$ .
  3. Kita memiliki  $L_1 = R_0 = 001010$  dan  $R_1 = L_0 \oplus f(R_0, K_2)$ . Kalkulasi  $f(R_0, K_2)$  dilakukan dengan langkah-langkah berikut:
    - (a) Kita hitung  $E(R_0) = E(001010) = 00010110$  dan  $E(R_0) \oplus K_2 = 00010110 \oplus 11000111 = 11010001$ .
    - (b) Hasil dari  $E(R_2) \oplus K_2$  dibagi menjadi dua bagian, yaitu 1101 dan 0001.
    - (c) Kita memiliki  $S_1(1101) = 111$  berdasarkan (4.5) dan  $S_2(0001) = 000$  berdasarkan (4.6). Selanjutnya hasil dari kedua *S-box* ini digabungkan sehingga diperoleh 111000.
  4. Setelah diperoleh  $f(R_0, K_2) = 111000$  kita hitung  $R_1 = L_0 \oplus f(R_0, K_2) = 001101 \oplus 111000 = 110101$ .
  5. Hasil dari putaran pertama adalah  $L_1 = 001010$  dan  $R_1 = 110101$ .
  6. Pada putaran kedua kita menggunakan  $K_1 = 11100011$ . Kita memiliki  $L_2 = R_1 = 110101$  dan  $R_2 = L_1 \oplus f(R_1, K_1)$ . Kalkulasi  $f(R_1, K_1)$  dilakukan dengan langkah-langkah berikut:
    - (a) Kita hitung  $E(R_1) = E(110101) = 11101001$  dan  $E(R_1) \oplus K_1 = 11101001 \oplus 11100011 = 00001010$ .
    - (b) Hasil dari  $E(R_2) \oplus K_1$  dibagi menjadi dua bagian, yaitu 0000 dan 1010.
    - (c) Kita memiliki  $S_1(0000) = 101$  berdasarkan (4.5) dan  $S_2(1010) = 000$  berdasarkan (4.6). Selanjutnya hasil kedua *S-box* ini digabungkan sehingga diperoleh 101000.
  7. Setelah diperoleh  $f(R_1, K_1) = 101000$  kita hitung  $R_2 = L_1 \oplus f(R_1, K_1) = 001010 \oplus 101000 = 100010$ .
  8. Kita memperoleh hasil  $L_2 = 110101$  dan  $R_2 = 100010$ . Untuk memperoleh teks asal (*plaintext*) kita lakukan penukaran (*switch*) pada nilai  $L_2$  dan  $R_2$  sehingga didapatkan  $L_2 = 100010$  dan  $R_2 = 110101$ . Teks asal (*ciphertext*) yang dimaksud adalah  $L_2R_2 = 100010110101$ . Perhatikan bahwa  $L_2R_2$  sama dengan pesan  $P$  yang telah dijelaskan sebelumnya.

Pada tutorial berikut akan dijelaskan penerapan SDES pada SageMath. SageMath menggunakan SDES yang sedikit berbeda dengan penjelasan yang telah diberikan sebelumnya, meskipun cara kerjanya secara esensial sama. SageMath menggunakan deskripsi SDES sebagaimana dijelaskan di [17]. Beberapa perbedaan antara SDES yang telah dijelaskan dengan SDES pada SageMath berdasarkan [17] adalah:

1. Ukuran pesan yang digunakan pada SDES di SageMath adalah 8 bit, bukan 12 bit. Tujuannya adalah agar setiap karakter (yang berukuran 1 byte atau 8 bit) dapat direpresentasikan langsung dalam satu pesan.
2. Ukuran kunci yang digunakan pada SDES di SageMath adalah 10 bit, bukan 9 bit.

- 
3. *S-box* yang digunakan pada SDES di SageMath memiliki definisi yang berbeda bila dibandingkan dengan definisi *S-box* pada (4.5) dan (4.6).
  4. SDES pada SageMath tidak menggunakan fungsi pengembang (*expander function*) yang dijelaskan di (4.4).
  5. Pengguna tidak perlu mendefinisikan banyaknya putaran (*round*) pada SDES di SageMath.

Kita dapat menerapkan SDES pada SageMath dengan perintah berikut:

```
from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES(); print(sdes)
bin = BinaryStrings(); print(bin)
# Misalkan P adalah pesan
P = [bin(str(randint(0, 1))) for i in range(8)]; print(P)
# Misalkan K adalah kunci
K = sdes.random_key(); print(K)
# Misalkan C adalah ciphertext
C = sdes.encrypt(P,K); print(C)
plaintext = sdes.decrypt(C,K); print(plaintext);
print(P == plaintext)
```

Pada skrip ini kita memiliki pesan (*plaintext*)  $P$  yang dibangkitkan secara acak. Kita dapat mengganti  $P$  dengan *list* apapun yang panjangnya 8 dan entri-entrinya adalah 0 atau 1. Kemudian pada skrip ini kita juga menggunakan kunci  $K$  yang dibangkitkan secara acak menggunakan sintaks `sdes.random_key()`. Kita juga dapat mengganti  $K$  dengan *list* biner atas  $\{0, 1\}$  yang panjangnya 10. Proses enkripsi terhadap pesan  $P$  menggunakan kunci  $K$  dilakukan dengan sintaks `sdes.encrypt(P, K)`, sedangkan proses dekripsi terhadap teks sandi (*ciphertext*)  $C$  menggunakan kunci  $K$  dilakukan dengan sintaks `sdes.decrypt(C, K)`. Baris terakhir pada skrip memeriksa apakah hasil dekripsi sama dengan pesan awal yang dienkripsi. Salah satu keluaran yang mungkin dari skrip di atas adalah:

```
Simplified DES block cipher with 10-bit keys
Free binary string monoid
[0, 0, 0, 1, 0, 0, 0, 1]
[0, 0, 0, 1, 1, 1, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 1]
True
```

Salah satu hal yang penting diperhatikan adalah tipe data dari pesan (*plaintext*)  $P$  yang akan dienkripsi, kunci  $K$  yang digunakan, dan teks sandi  $C$ . Perhatikan bahwa  $P$ ,  $K$ , dan  $C$  harus dinyatakan dalam *list* biner. Misalkan pesan kita adalah sebuah string biner  $P = 10100101$  dan kunci yang digunakan adalah  $K = 1111100000$ , maka kita dapat memakai skrip berikut:

```

from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES(); print(sdes)
bin = BinaryStrings(); print(bin)
# Misalkan P adalah pesan dalam bentuk 10100101
P = [bin(str(1)), bin(str(0)), bin(str(1)), bin(str(0)),
      bin(str(0)), bin(str(1)), bin(str(0)), bin(str(1))]
print(P)
# Misalkan K adalah kunci dalam bentuk 1111100000
K = [bin(str(1)), bin(str(1)), bin(str(1)), bin(str(1)), bin(str(1)),
      bin(str(0)), bin(str(0)), bin(str(0)), bin(str(0)), bin(str(0))]
print(K)
# Misalkan C adalah ciphertext
C = sdes.encrypt(P,K); print(C)
print(C)
plaintext = sdes.decrypt(C,K); print(plaintext);
print(P == plaintext)

```

Hasil keluaran dari skrip tersebut adalah:

```

Simplified DES block cipher with 10-bit keys
Free binary string monoid
[1, 0, 1, 0, 0, 1, 0, 1]
[1, 1, 1, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 1, 0]
[0, 0, 0, 1, 1, 0, 1, 0]
[1, 0, 1, 0, 0, 1, 0, 1]
True

```

Proses enkripsi SDES pada SageMath yang dilakukan dengan fungsi `encrypt(..., ...)` dapat dilakukan menggunakan masukan *list* maupun string. Misalkan kita memiliki teks pesan (*plaintext*)  $P = 01010101$  dan kunci  $K = 1010000010$ , maka kita dapat melakukan enkripsi  $P$  menggunakan kunci  $K$  pada SDES sebagai berikut:

```

from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES()
P = [0, 1, 0, 1, 0, 1, 0, 1];
K = [1, 0, 1, 0, 0, 0, 0, 1, 0]
C = sdes.encrypt(P, K); print(C)

```

Hasil dari enkripsi ini adalah *list*  $[1, 1, 0, 0, 0, 0, 0, 1]$ . Pesan  $P$  maupun kunci  $K$  dapat dinyatakan dalam bentuk string sebagaimana yang dilakukan pada skrip berikut:

```

from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES()
P = '01010101';
K = '1010000010';
C = sdes.encrypt(sdes.string_to_list(P), sdes.string_to_list(K))
print(C)

```

Perhatikan bahwa terdapat *method* tambahan yang kita berikan, yaitu `.string_to_list`. *Method* ini dipakai untuk mengonversi sebuah string ke suatu *list*. Hasil dari proses enkripsi ini juga berupa *list*  $[1, 1, 0, 0, 0, 0, 0, 1]$ .

Proses dekripsi SDES pada SageMath yang dilakukan dengan fungsi `decrypt(..., ...)` dapat dilakukan menggunakan masukan *list* maupun string. Misalkan kita memiliki teks sandi (*ciphertext*)  $C = 01010101$  dan kunci  $K = 1010000010$ , maka kita dapat melakukan dekripsi  $C$  menggunakan kunci  $K$  pada SDES sebagai berikut:

---

```

from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES()
C = [0, 1, 0, 1, 0, 1, 0, 1]
K = [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
P = sdes.decrypt(C, K); print(P)

```

Hasil dari skrip ini adalah *list* [0, 0, 0, 1, 0, 1, 0, 1] yang menyatakan teks pesan  $P$ . Teks sandi  $C$  maupun kunci  $K$  dapat dinyatakan dalam bentuk string sebagai berikut:

```

from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES()
C = '01010101';
K = '1010000010';
P = sdes.decrypt(sdes.string_to_list(C), sdes.string_to_list(K));
print(P)

```

Hasil dari skrip ini juga berupa *list* [0, 0, 0, 1, 0, 1, 0, 1]. Sama seperti pada proses enkripsi sebelumnya, kita menggunakan *method* `string_to_list` untuk mengubah string biner ke dalam *list* biner. Dalam kriptografi simetrik, jika  $\mathcal{E}(P, K)$  adalah fungsi enkripsi untuk masukan teks pesan  $P$  dan kunci  $K$  serta  $\mathcal{D}(C, K)$  adalah fungsi dekripsi untuk masukan teks sandi  $C$  dan kunci  $K$  maka berlaku

$$\mathcal{D}(\mathcal{E}(P, K), K) = P, \quad (4.7)$$

hal ini dapat disimulasikan pada SageMath sebagai berikut:

```

from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES()
P = [randint(0,1) for i in range(8)]; print(P)
K = [randint(0,1) for i in range(10)]; print(K)
C = sdes.encrypt(P, K); print(C)
plaintext = sdes.decrypt(C,K); print(P)
print(plaintext == sdes.decrypt(sdes.encrypt(P, K), K))

```

Pada skrip ini kita menggunakan teks pesan  $P$  berupa *list* biner dengan panjang 8 yang dibangkitkan secara acak dan kunci  $K$  berupa *list* biner dengan panjang 10. Pada baris terakhir kita memeriksa apakah hasil dekripsi dari enkripsi  $P$  dengan kunci  $K$  sama dengan  $P$ . Salah satu hasil yang mungkin dari skrip ini adalah:

```

[1, 1, 1, 1, 1, 1, 0, 0]
[0, 0, 1, 1, 0, 0, 1, 0, 0, 1]
[1, 1, 0, 0, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 0, 0]
True

```

Di sini kita memakai  $P = 11111100$  dan  $K = 0011001001$ . Teks sandi yang dihasilkan adalah  $C = 11001111$ . Kita perlu berhati-hati dalam menggunakan SDES pada SageMath. *List* pesan maupun kunci harus didefinisikan atas string biner dengan karakter pada  $\{0, 1\}$ , bukan bilangan bulat (*integer*).

**Latihan 4.1.2** Misalkan kita menggunakan skrip berikut:

---

```

from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES(); print(sdes)
# Misalkan P adalah pesan dalam bentuk 10100101
P = [1,0,1,0,0,1,0,1]
print(P)
# Misalkan K adalah kunci dalam bentuk 1111100000
K = [1,1,1,1,1,0,0,0,0,0]
# Misalkan C adalah ciphertext
print(K)
C = sdes.encrypt(P,K); print(C)
print(C)
plaintext = sdes.decrypt(C,K); print(plaintext);
print(P == plaintext)
print(P)
print(plaintext)

```

Apa keluaran dari skrip di atas? Masalah apa yang terjadi pada skrip tersebut? Menurut Anda mengapa hal ini dapat terjadi?

Sejauh ini kita telah melihat penerapan SDES ketika teks pesan berupa *list* biner atau string biner. Secara *default* kita juga dapat melakukan **enkripsi terhadap sebuah pesan dalam string apa pun selama representasi string binernya memiliki panjang yang habis dibagi 8**. Misalkan kita akan mengenkripsi string *Encrypt this using S-DES!*, maka kita dapat menggunakan skrip berikut:

```

from sage.crypto.block_cipher.sdes import SimplifiedDES
from sage.crypto.util import bin_to_ascii
sdes = SimplifiedDES(); print(sdes)
bin = BinaryStrings(); print(bin)
P = bin.encoding('Encrypt this using S-DES!');
print(len(P));
print(len(P)% 8 == 0);
K = sdes.list_to_string(sdes.random_key()); print(K)
C = sdes(P,K,algorithm ="encrypt");
print(len(C))
plaintext = sdes(C, K, algorithm="decrypt");
print(plaintext == P)
print(bin_to_ascii(plaintext))

```

Pada skrip ini kita memakai perintah `sdes(P, K, algorithm="encrypt")` untuk melakukan enkripsi dan perintah `sdes(C, K, algorithm="decrypt")` untuk melakukan dekripsi. Perintah `sdes(P, K, algorithm="encrypt")` maupun `sdes(C, K, algorithm="decrypt")` memungkinkan masukan yang formatnya adalah string biner dengan panjang berapapun yang habis dibagi 8. Salah satu hasil dari skrip ini adalah:

```

Simplified DES block cipher with 10-bit keys
Free binary string monoid
200
True
0110011101
200
True
Encrypt this using S-DES!

```

Pada hasil ini kita melihat bahwa:

- 
1. Kita mengubah string Encrypt this using S-DES! ke dalam string biner menggunakan fungsi `bin.encoding(...)`. Hal ini juga berlaku untuk pesan-pesan yang lain.
  2. Panjang representasi biner untuk  $P$  adalah 200 bit dan 200 habis dibagi 8.
  3. Kunci yang dipakai adalah  $K = 0110011101$ .
  4. Panjang dari teks sandi yang dihasilkan, yaitu  $C$ , juga 200 bit.
  5. Hasil dari dekripsi teks sandi sama dengan teks pesan.
  6. Untuk mengubah pesan dalam string biner ke dalam string ASCII kita menggunakan fungsi `bin_to_ascii(...)`.

Ketika sebuah string masukan memiliki representasi biner yang panjangnya tidak habis dibagi 8, maka kita bisa melakukan *padding* agar representasi biner string tersebut habis dibagi 8. Untuk string yang direpresentasikan dalam ASCII, karena setiap karakter direpresentasikan dalam 8 bit, maka pasti representasi biner dari string tersebut habis dibagi 8.

**Latihan 4.1.3** Dengan memanfaatkan algoritma enkripsi dan dekripsi SDES pada SageMath, buatlah:

1. Fungsi dengan nama `SDES_Enc (P, K)` dengan masukan  $P$  berupa string dan  $K$  berupa string biner 10 bit serta menghasilkan enkripsi SDES terhadap pesan  $P$  menggunakan kunci  $K$ . Lihat Gambar (4.4) untuk ilustrasi.
2. Fungsi dengan nama `SDES_Dec (C, K)` dengan masukan  $C$  berupa string dan  $K$  berupa string biner 10 bit serta menghasilkan dekripsi SDES terhadap sandi  $C$  menggunakan kunci  $K$ . Lihat Gambar (4.5) untuk ilustrasi.

Petunjuk: gunakan perintah `sdes(P, K, algorithm="encrypt")` untuk proses enkripsi dan perintah `sdes(C, K, algorithm="decrypt")` untuk proses dekripsi.

```
print(SDES_Enc('Hallo','1111111111'))
print(SDES_Enc('SageMath','1110001110'))
print(SDES_Enc('Cryptography','1010101010'))
```

gCã  
I`BÁS`ÿ7  
fQßz OQ©B

Gambar 4.4: Ilustrasi masukan dan keluaran fungsi `SDES_Enc (P, K)`.

```
print(SDES_Dec('gC••ã','1111111111'))
print(SDES_Dec('I`BÁS`ÿ7','1110001110'))
print(SDES_Dec('fQ•ßz—OQ©B••','1010101010'))
```

Hallo  
SageMath  
Cryptography

Gambar 4.5: Ilustrasi masukan dan keluaran fungsi `SDES_Dec (P, K)`.

## 4.2 Mini AES

Pada tahun 1997 NIST membuat sayembara dalam rangka mencari sistem kripto untuk menggantikan DES. Algoritma enkripsi tersebut harus dapat mengakomodasi kunci dengan ukuran 128, 192, dan 256 bit, serta blok masukan dari sistem tersebut harus berukuran 128 bit. Algoritma ini selanjutnya dinamakan sebagai Standar Enkripsi Lanjut (*Advanced Encryption Standard*, AES). Dari seluruh kandidat algoritma enkripsi yang diajukan, algoritma yang dibuat oleh Joan Daemen dan Vincent Rijmen (yang dinamakan dengan Rijndael) terpilih sebagai algoritma yang digunakan untuk standar enkripsi ini.

Pada tutorial ini kita tidak akan membahas AES secara detail, namun kita akan mengkaji sistem kripto mini AES. Algoritma kriptografi simetrik ini dikembangkan oleh Raphael Phan [18] sebagai sarana pembelajaran AES. Sebagaimana SDES, mini AES tidak digunakan untuk keperluan pengembangan sistem.

Pada mini AES kita menggunakan *field* hingga  $GF(16)$ . Sebuah pesan (*plaintext*) pada mini AES direpresentasikan dalam string 16 bit. Untuk melakukan enkripsi blok pesan 16 bit tersebut dibagi menjadi menjadi empat bagian yang masing-masing berukuran 4 bit sebagaimana diilustrasikan pada Gambar 4.6. Misalkan kita memiliki pesan  $P = p_0 p_1 p_2 p_3$  dengan  $p_i$  ( $1 \leq i \leq 4$ ) masing-masing berukuran 4 bit, pada langkah pertama kita membuat matriks  $2 \times 2$  atas  $GF(16)$ , yang dinamakan dengan  $\mathbf{P}$ , dengan definisi  $\mathbf{P} = \begin{bmatrix} p_0 & p_2 \\ p_1 & p_3 \end{bmatrix}$ .

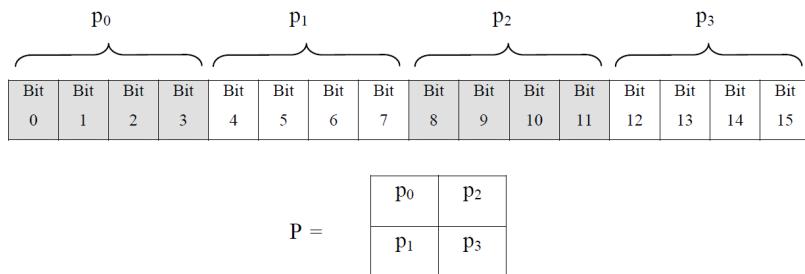


Figure 2:  $2 \times 2$  Matrix Representation of the 16-bit Block

Gambar 4.6: Ilustrasi cara mengubah pesan  $P$  ke dalam matriks  $\mathbf{P}$  yang berukuran  $2 \times 2$  atas  $GF(16)$ . Gambar diambil dari [18, Gambar 2].

Sebagaimana SDES, mini AES dapat dijalankan dalam beberapa putaran. Satu putaran pada mini AES terdiri dari empat komponen, yaitu *NibbleSub*, *ShiftRow*, *MixColumn*, dan *KeyAddition*. *NibbleSub* pada mini AES serupa dengan *S-box* pada SDES, sedangkan komponen-komponen lainnya menggunakan manipulasi aljabar pada matriks. Penjelasan detail dan contoh dari proses enkripsi dan dekripsi mini AES dapat dilihat pada [18]. Satu putaran enkripsi pada mini AES dapat dilihat pada Gambar 4.7. Mini AES menggunakan kunci berupa matriks  $2 \times 2$  atas  $GF(16)$ .

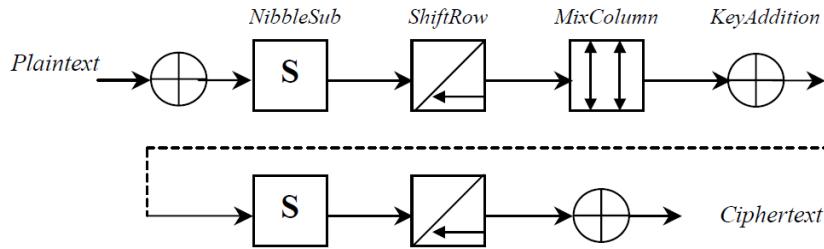
Misalkan kita memiliki pesan yang telah dikonversi ke dalam matriks atas  $GF(16)$  dengan pembangkit  $x$  berbentuk

$$\mathbf{P} = \begin{bmatrix} x^3 + x & x^2 + 1 \\ x^2 + x & x^3 + x^2 \end{bmatrix}$$

dan kunci yang digunakan dalam bentuk matriks adalah

$$\mathbf{K} = \begin{bmatrix} x^3 + x^2 & x^3 + x \\ x^3 + x^2 + x & x^2 + x + 1 \end{bmatrix}.$$

Proses enkripsi dari pesan ini menggunakan mini AES dapat dilakukan dengan skrip berikut:



**Figure 7:** The Mini-AES Encryption Process

Gambar 4.7: Satu putaran ekripsi pada mini AES. Gambar diambil dari [18, Gambar 7].

```

from sage.crypto.block_cipher.miniaes import MiniAES
maes = MiniAES()
# membangkitkan GF(16)
F = FiniteField(16, "x")
# membangkitkan ruang matriks GF(16) yang berukuran 2 kali 2
MS = MatrixSpace(F, 2, 2)
# Pesan ditulis dalam elemen di GF(16)
print('Pesan yang dienkripsi:')
P = MS([F("x^3 + x"), F("x^2 + 1"), F("x^2 + x"), F("x^3 + x^2")]); print(P)
# Kunci adalah matriks atas GF(16) yang berukuran 2 x 2
print('Kunci yang dipakai:')
K = MS([F("x^3 + x^2"), F("x^3 + x"), F("x^3 + x^2 + x"), F("x^2 + x + 1")]); print(K)
print('Hasil enkripsi: ')
C = maes.encrypt(P, K); print(C) # proses enkripsi
print('Hasil dekripsi: ')
plaintext = maes.decrypt(C, K); print(plaintext) # proses dekripsi
print('Pemeriksaan kesamaan teks pesan dan hasil dekripsi')
print(plaintext == P)

```

Hasil dari proses ini adalah

Pesan yang dienkripsi:

```
[ x^3 + x  x^2 + 1]
[ x^2 + x  x^3 + x^2]
```

Kunci yang dipakai:

```
[ x^3 + x^2      x^3 + x]
[ x^3 + x^2 + x  x^2 + x + 1]
```

Hasil enkripsi:

```
[      x      x^2 + x]
[ x^3 + x^2 + x  x^3 + x]
```

Hasil dekripsi:

```
[ x^3 + x  x^2 + 1]
[ x^2 + x  x^3 + x^2]
```

Pemeriksaan kesamaan teks pesan dan hasil dekripsi

True

Proses yang telah dijelaskan memberikan teks sandi (dalam bentuk matriks  $2 \times 2$  atas  $GF(16)$ ) berbentuk  $C = \begin{bmatrix} x & x^2 + x \\ x^3 + x^2 + x & x^3 + x \end{bmatrix}$ .

Kita dapat menguji persamaan (4.7) untuk mini AES dengan skrip berikut:

---

```

from sage.crypto.block_cipher.miniaes import MiniAES
maes = MiniAES()
# membangkitkan ruang matriks 2 kali 2 atas GF(16)
MS = MatrixSpace(GF(16, 'x'), 2, 2)
print('Pesan yang digunakan: ')
# membangkitkan pesan secara acak
P = MS.random_element(); print(P)
# membangkitkan kunci secara acak
print('Kunci yang digunakan: ')
K = maes.random_key(); print(K)
# melakukan enkripsi
print('Hasil enkripsi: ')
C = maes.encrypt(P, K); print(C)
# melakukan dekripsi
plaintext = maes.decrypt(C, K)
print('Kesamaan teks awal dan hasil dekripsi: ')
# memeriksa kesamaan teks awal dan hasil dekripsi
print(plaintext == P)

```

Salah satu keluaran dari proses ini adalah:

```

Pesan yang digunakan:
[      x      x^3]
[x^2 + x + 1  x^3 + x]
Kunci yang digunakan:
[ x^3 + x + 1      x]
[x^3 + x^2 + 1  x^3 + x^2]
Hasil enkripsi:
[      0      1]
[x^3 + x^2  x^3 + x]
Kesamaan teks awal dan hasil dekripsi:
True

```

Sebagaimana telah dijelaskan sebelumnya, mini AES meninjau  $GF(16)$  pada pesan maupun kuncinya. Sebuah pesan maupun kunci pada mini AES harus dapat direpresentasikan dalam string biner 16 bit (pesan sebenarnya juga dapat direpresentasikan dalam string biner yang panjangnya adalah kelipatan 16). Untuk pesan maupun kunci yang terdiri dari 16 bit, pesan atau kunci tersebut kemudian dibagi menjadi empat bagian yang masing-masing terdiri atas 4 bit. String biner 4 bit dipetakan menjadi elemen pada  $GF(16)$  dengan pemetaan yang dijelaskan berikut:

| String 4 bit | $GF(2^4)$     | String 4 bit | $GF(2^4)$           |
|--------------|---------------|--------------|---------------------|
| 0000         | 0             | 1000         | $x^3$               |
| 0001         | 1             | 1001         | $x^3 + 1$           |
| 0010         | $x$           | 1010         | $x^3 + x$           |
| 0011         | $x + 1$       | 1011         | $x^3 + x + 1$       |
| 0100         | $x^2$         | 1100         | $x^3 + x^2$         |
| 0101         | $x^2 + 1$     | 1101         | $x^3 + x^2 + 1$     |
| 0110         | $x^2 + x$     | 1110         | $x^3 + x^2 + x$     |
| 0111         | $x^2 + x + 1$ | 1111         | $x^3 + x^2 + x + 1$ |

Pemetaan ini berarti string  $a_0a_1a_2a_3$  diasosiasikan dengan  $a_0x^3 + a_1x^2 + a_2x + a_3$ . Akibatnya kita juga dapat menggunakan string sebagai masukan pada SageMath. Kunci pada mini

---

AES adalah string biner dengan panjang 16 (yang selanjutnya akan dikonversi ke dalam matriks  $2 \times 2$  atas  $GF(16)$ ). Proses enkripsi dari pesan Encrypt this secret message! menggunakan kunci  $K = 0100101101000101$  dapat dilakukan menggunakan skrip berikut:

```
from sage.crypto.block_cipher.miniaes import MiniAES
from sage.crypto.util import bin_to_ascii
maes = MiniAES()
bin = BinaryStrings()
key = bin(str('0100101101000101'))
# pesan awal:
message = 'Encrypt this secret message!'
print(message)
P = bin.encoding(message)
# hasil enkripsi:
C = maes(P, key, algorithm="encrypt")
print(bin_to_ascii(C))
plaintext = maes(C, key, algorithm="decrypt")
# memeriksa kesamaan teks awal dan hasil dekripsi
print(plaintext == P)
```

Hasil dari keluaran skrip ini dapat dilihat pada Gambar 4.8.

```
Encrypt this secret message!
!õxLíGmRi¬ç#²"§ÙGm E0pæ¼
True
```

Gambar 4.8: Contoh keluaran simulasi proses enkripsi dan dekripsi pada mini AES.

Secara *default* penggunaan mini AES pada SageMath mengharuskan string masukan memiliki panjang 16 bit atau kelipatannya. **Ini berarti banyaknya karakter pada string masukan haruslah genap (karena satu karakter direpresentasikan dalam 8 bit).** Jika tidak, maka kita harus menambahkan satu karakter supaya panjang dari string masukan adalah kelipatan 16 bit.

**Latihan 4.2.1** Dengan memanfaatkan algoritma enkripsi dan dekripsi mini AES pada SageMath, buatlah:

1. Fungsi dengan nama `maes_enc(P, K)` dengan masukan  $P$  berupa string dan  $K$  berupa string biner 16 bit serta menghasilkan enkripsi mini AES terhadap pesan  $P$  menggunakan kunci  $K$ . Apabila banyaknya karakter pada pesan tidak genap, maka string pesan ditambahkan dengan huruf  $a$  di bagian akhir. Lihat Gambar 4.9 untuk ilustrasi.
2. Fungsi dengan nama `maes_dec(C, K)` dengan masukan  $C$  berupa string dan  $K$  berupa string biner 16 bit serta menghasilkan dekripsi mini AES terhadap sandi  $C$  menggunakan kunci  $K$ . Lihat Gambar 4.10 untuk ilustrasi. Teks sandi  $C$  diasumsikan memiliki sebanyak genap karakter.

```
In [54]: print(maes_enc('Halloo','1111000011110000'))  
print(maes_enc('Hallo','1111000011110000'))  
print(maes_enc('SageMath','0000111100001111'))  
print(maes_enc('Mathematics','0000111100001111'))  
  
,ÝÁþ  
,Ýþ  
ÑJÓÙü  
ÑÜñëæœ—æT
```

Gambar 4.9: Ilustrasi masukan dan keluaran fungsi `maes_enc(P, K)`.

```
In [55]: print(maes_dec('•Ý·Ap', '1111000011110000'))  
print(maes_dec('•Ý·;ø', '1111000011110000'))  
print(maes_dec('ÑJÚ•Ûü•', '0000111100001111'))  
print(maes_dec('•Ûü•»¾æ•-•aI', '0000111100001111'))
```

Halloo  
Halloa  
SageMath  
Mathematicsa

Gambar 4.10: Ilustrasi masukan dan keluaran fungsi `maes_dec(P, K)`.

# Topik 5

## Contoh Kriptografi Asimetrik pada SageMath

Pada topik ini kita akan membahas beberapa contoh penerapan kriptografi asimetrik pada SageMath. Pembahasan akan dimulai dengan kajian tentang pertukaran (atau persetujuan) kunci Diffie-Hellman yang merupakan salah satu bahasan penting dalam kriptografi modern. Selanjutnya kita akan mengkaji salah satu varian dari protokol Diffie-Hellman yang meninjau kurva eliptik. Sebelum membahas mengenai sistemkripto asimetrik lebih jauh, terlebih dulu kita akan mengkaji cara merepresentasikan pesan sebagai bilangan sebuah ring atau *field*. Hal ini cukup penting mengingat pada sistemkripto asimetrik seperti RSA maupun Elgamal pesan direpresentasikan sebagai elemen di sebuah ring atau *field* tertentu.

### 5.1 Pertukaran Kunci Diffie-Hellman

Dari pembahasan yang telah dijelaskan pada Topik 4 kita melihat dua contoh penyederhanaan dari sistemkripto modern yang dapat digunakan untuk mengamankan pesan secara digital. Perhatikan bahwa pada sistemkripto simetrik pengirim pesan dan penerimanya harus menyetujui sebuah kunci rahasia yang digunakan pada saat enkripsi maupun dekripsi. Persetujuan kunci ini biasanya dilakukan sebelum transmisi pesan dilakukan. Idealnya kunci yang digunakan tidak boleh diketahui siapa pun selain pengirim dan penerima pesan tersebut. Agar kerahasiaan kunci terjamin, maka pengirim maupun penerima dapat bertemu secara rahasia atau menggunakan saluran komunikasi lain yang aman dan rahasia untuk menyetujui kunci yang digunakan. Sayangnya hal ini pada praktiknya tidak selalu dapat dilakukan.

Ada beberapa solusi dari permasalahan yang telah dijelaskan sebelumnya. Pertama adalah penggunaan skema pertukaran (atau persetujuan) kunci yang memungkinkan setiap partisipan (pengirim maupun penerima) menyetujui kunci yang digunakan pada proses enkripsi maupun dekripsi. Solusi lain adalah pemakaian sistemkripto asimetrik yang menggunakan kunci yang berbeda untuk proses enkripsi maupun dekripsi. Skema pertukaran kunci memungkinkan kita menggunakan sistemkripto simetrik dalam kondisi ketika pengirim maupun penerima pesan tidak dapat bertemu atau berkomunikasi secara aman untuk menyetujui kunci yang digunakan dalam komunikasi rahasia.

Pada subtopik ini kita akan membahas protokol pertukaran kunci (atau persetujuan kunci) Diffie-Hellman untuk dua partisipan. Protokol ini pertama kali diajukan pada 1976 oleh W. Diffie dan M. Hellman [19] dan dijelaskan pada Protokol 1. Untuk mempermudah misalkan kedua partisipan yang terlibat adalah Alice dan Bob.

**Protokol 1 (Protokol Diffie-Hellman yang dirangkum dari Tabel 2.2 pada [2])** Protokol pertukaran kunci atau persetujuan kunci Diffie-Hellman untuk dua partisipan dijelaskan sebagai

---

berikut:

1. **Pembentukan parameter publik.** Pihak yang terpercaya memilih *field* hingga  $\mathbb{F}_q$  dan sebuah elemen  $g \in \mathbb{F}_q^*$  yang merupakan pembangkit dari  $\mathbb{F}_q^*$  (lihat Teorema 2.5.2).
2. **Komputasi nilai rahasia.** Alice memilih sebuah bilangan bulat  $a \in [1, q - 1]$  dan menghitung  $A = g^a$ . Secara serupa Bob memilih sebuah bilangan bulat  $b \in [1, q - 1]$  dan menghitung  $B = g^b$ . Di sini baik  $A$  maupun  $B$  adalah elemen di  $\mathbb{F}_q^*$ . Nilai dari  $a$  maupun  $b$  bersifat acak dan rahasia (nilai  $a$  hanya diketahui oleh Alice sedangkan nilai  $b$  hanya diketahui oleh Bob).
3. **Pertukaran kunci secara publik.** Alice mengirimkan nilai  $A$  kepada Bob dan secara serupa Bob mengirimkan nilai  $B$  kepada Alice. Pertukaran kunci dapat dilakukan secara publik.
4. **Komputasi kunci bersama.** Untuk menghitung kunci bersama, Alice menghitung nilai dari  $B^a$  sedangkan Bob menghitung nilai dari  $A^b$ . Secara matematis  $B^a = A^b$  dan nilai ini adalah kunci bersama yang akan digunakan Alice dan Bob untuk komunikasi lebih lanjut.

Pada Protokol 1 Alice menghitung  $B^a$  sedangkan Bob menghitung  $A^b$ , perhatikan bahwa

$$B^a = (g^b)^a = g^{ba} = g^{ab} = (g^a)^b = A^b. \quad (5.1)$$

Pada ekspresi (5.1) semua perhitungan dilakukan pada  $\mathbb{F}_q^*$ . Sebagai ilustrasi perhatikan contoh pertukaran kunci Diffie-Hellman yang diambil dari [2, halaman 69] berikut. Misalkan pihak yang terpercaya memilih  $\mathbb{F}_{941}^*$  dan pembangkit  $g = 627$ . Pada komputasi nilai rahasia Alice memilih  $a = 347$  dan menghitung  $A = 627^{347} = 390$  sedangkan Bob memilih  $b = 781$  dan menghitung  $B = 627^{781} = 691$ . Pada praktiknya nilai  $a$  dan  $b$  dipilih secara acak pada rentang  $[1, q - 1]$ . Selanjutnya Alice dan Bob saling bertukar nilai  $A$  dan  $B$ . Untuk memperoleh kunci bersama Alice menghitung nilai  $B^a = 691^{347} = 470$  sedangkan Bob menghitung nilai  $A^b = 390^{781} = 470$ . Pada contoh ini semua aritmetika dilakukan pada  $\mathbb{F}_{941}^*$ . Nilai 470 adalah kunci rahasia bersama yang digunakan Alice dan Bob. Ilustrasi dari proses ini dapat dijelaskan dalam simulasi berikut pada SageMath:

```

F = GF(941)
# Memeriksa apakah 627 adalah pembangkit dari F*
g = F(627)
print(g.is_primitive_root())
# Komputasi nilai rahasia yang dilakukan Alice.
a = 347
# Nilai publik dari Alice
print('Nilai publik dari Alice:')
A = g^a; print(A)
# Komputasi nilai rahasia yang dilakukan Bob.
b = 781
# Nilai publik dari Bob
print('Nilai publik dari Bob:')
B = g^b; print(B)
# Pertukaran kunci
# Alice mengirimkan A ke Bob. Bob mengirimkan B ke Alice.
# Perhitungan kunci bersama.
# Alice menghitung B^a
keyAlice = A^b;
# Bob menghitung A^b
keyBob = B^a;
# Memeriksa kesamaan dari kunci bersama yang diperoleh
print('Memeriksa kesamaan kunci bersama yang diperoleh:')
print(keyAlice == keyBob)

```

Pada SageMath *method* `.is_primitive_root()` digunakan untuk memeriksa apakah suatu elemen pada suatu *field* hingga merupakan pembangkit atau bukan. Nilai dari pembangkit  $g$  harus didefinisikan pada  $\mathbb{F}_q^*$ . Pada skrip sebelumnya hal ini dilakukan dengan perintah `g = F(627)`. Nilai dari  $a$  dan  $b$  yang berturut-turut merupakan nilai rahasia dari Alice dan Bob adalah bilangan bulat pada rentang  $[1, q - 1]$ . Hasil keluaran dari skrip ini adalah:

```

True
Nilai publik dari Alice:
390
Nilai publik dari Bob:
691
Memeriksa kesamaan kunci bersama yang diperoleh:
True

```

Kita dapat membuat simulasi pertukaran kunci Diffie-Hellman secara lebih umum pada SageMath dengan memperhatikan kaidah-kaidah berikut:

1. Pertukaran kunci dilakukan dengan aturan aritmetika pada  $\mathbb{F}_p^*$  untuk nilai  $p$  tertentu yang merupakan bilangan prima.
2. Pembangkit  $g$  dapat dipilih secara acak dari pembangkit-pembangkit yang mungkin pada  $\mathbb{F}_p^*$ . Pada contoh sebelumnya kita dapat memeriksa bahwa 2 adalah pembangkit dari  $\mathbb{F}_{941}^*$ .
3. Nilai rahasia Alice merupakan bilangan bulat  $a$  yang dipilih secara acak pada rentang  $[1, p - 1]$ . Kemudian nilai rahasia Bob adalah bilangan bulat  $b$  yang dipilih secara acak pada rentang yang sama.
4. Alice menghitung  $A = g^a$  dan Bob menghitung  $B = g^b$ . Alice dan Bob kemudian saling bertukar nilai ini secara publik.

- 
5. Untuk memperoleh kunci bersama, Alice menghitung  $B^a$  sedangkan Bob menghitung  $A^b$ .

Simulasi pertukaran kunci dengan kaidah-kaidah yang telah dijelaskan dapat disimulasikan pada SageMath dengan skrip berikut:

```
import random
p = Integer(input('Masukan bilangan prima:'))
F = GF(p)
# Mencari pembangkit dari F*.
print('Mencari pembangkit dari F*.')
g = F.primitive_element(); print(g)
# Komputasi rahasia Alice dan Bob.
# Pangkat rahasia Alice.
a = random.randint(1,p-1)
# Pangkat rahasia Bob.
b = random.randint(1,p-1)
print('Nilai publik dan pertukarannya.')
# Kalkulasi nilai publik dari Alice dan Bob.
A = g^a; # A adalah nilai publik dari Alice.
print('Nilai publik dari Alice, A: ', A)
B = g^b; # B adalah nilai publik dari Bob.
print('Nilai publik dari Bob, B: ', B)
print('Alice dan Bob bertukar nilai A dan B')
# Kalkulasi kunci bersama.
print('Kalkulasi kunci bersama.')
# keyAlice adalah kunci yang dihitung Alice
keyAlice = B^a
# keyBob adalah kunci yang dihitung Bob
keyBob = A^b
print('Pemeriksaan kesamaan kunci.')
print(keyAlice == keyBob)
```

Pada skrip simulasi kita memakai *library* random pada Python untuk menggunakan fungsi `random.randint(a,b)` yang digunakan untuk membangkitkan bilangan bulat secara acak antara  $a$  dan  $b$  (inklusif). Misalkan kita memasukkan nilai  $p = 1009$  pada simulasi pertukaran kunci Diffie-Hellman ini, maka salah satu keluaran yang mungkin adalah:

```
Masukan bilangan prima: 1009
Mencari pembangkit dari F*.
11
Nilai publik dan pertukarannya.
Nilai publik dari Alice, A: 787
Nilai publik dari Bob, B: 717
Alice dan Bob bertukar nilai A dan B
Kalkulasi kunci bersama.
Pemeriksaan kesamaan kunci.
True
```

Hasil ini memberikan informasi berikut:

1. Kita memakai *field* hingga  $\mathbb{F}_{1009}^*$  dengan pembangkit  $g = 11$ .
2. Kita mengetahui bahwa nilai  $A = 11^a = 787$  dan nilai  $B = 11^b = 717$ .
3. Nilai kunci bersama yang diperoleh oleh Alice maupun Bob adalah identik.

---

Salah satu cara untuk menyerang protokol pertukaran kunci Diffie-Hellman, atau dalam hal ini adalah mengetahui kunci bersama yang digunakan, adalah dengan menyelesaikan persamaan  $A = g^a$  dan  $B = g^b$  pada  $\mathbb{F}_q^*$  untuk nilai  $a$  dan  $b$  dari nilai-nilai  $A, B, g, q$  dan yang diketahui. Permasalahan ini terkait dengan masalah logaritma diskrit dan dijelaskan pada Permasalahan 5.1.1.

**Permasalahan 5.1.1** Misalkan  $g$  adalah pembangkit dari  $\mathbb{F}_q^*$  dan  $h \in \mathbb{F}_q^*$ . Masalah logaritma diskrit (discrete logarithm problem) pada  $\mathbb{F}_q^*$  dengan instace  $g$  dan  $h$  adalah masalah pencarian nilai  $x$  yang memenuhi

$$g^x = h. \quad (5.2)$$

Jika nilai  $x$  yang memenuhi persamaan (5.2) ada, maka  $x$  dikatakan sebagai logaritma diskrit dari  $h$  dalam basis  $g$  dan dapat ditulis dengan  $x = \log_g(h)$ .

Pada simulasi sebelumnya, kita memiliki masalah logaritma diskrit  $11^a = 787$  dan  $11^b = 717$  pada  $\mathbb{F}_{1009}^*$ . Kita dapat menggunakan SageMath untuk menyelesaikan masalah logaritma diskrit ini. Untuk mencari logaritma diskrit dari  $h$  dalam basis  $g$  pada  $\mathbb{F}_q^*$  kita dapat menggunakan sintaks `h.log(g)` dengan terlebih dulu mendefinisikan  $g$  sebagai elemen pada  $\mathbb{F}_q^*$ . Perhatikan skrip berikut:

```
p = 1009
F = GF(p)
g = F(11); print(g.is_primitive_root())
A = F(787);
B = F(717);
# Mencari nilai a dari A = g^a pada GF(p).
print('Pangkat rahasia Alice: ')
a = A.log(g); print(a)
# Mencari nilai b dari B = g^b pada GF(p).
print('Pangkat rahasia Bob: ')
b = B.log(g); print(b)
# Mencari kunci rahasia bersama.
print('Kunci rahasia bersama: ')
key = g^(a*b);
print(key)
```

Hasil dari skrip ini adalah:

```
True
Pangkat rahasia Alice:
246
Pangkat rahasia Bob:
959
Kunci rahasia bersama:
648
```

Skrip ini menunjukkan bahwa solusi dari  $11^a = 787$  dan  $11^b = 717$  pada  $\mathbb{F}_{1009}^*$  berturut-turut adalah  $a = 246$  dan  $b = 959$ . Akibatnya kita dengan mudah memperoleh  $11^{ab} = 648$  sebagai kunci bersama yang digunakan oleh Alice dan Bob. Pada praktiknya agar protokol pertukaran kunci Diffie-Hellman tidak mudah untuk diserang maka nilai  $p$  pada  $\mathbb{F}_p^*$  yang digunakan setidaknya memiliki representasi biner 1000 bit, atau  $p \geq 2^{1000}$ . Ini berarti representasi desimal dari  $p$  setidaknya adalah 301 digit.

**Latihan 5.1.2** Pertukaran kunci Diffie-Hellman dapat diperumum untuk tiga atau lebih partisipan (referensi lebih lanjut dapat dilihat di [20–23]). Proses pertukaran kunci Diffie-Hellman

---

untuk tiga partisipan yang paling sederhana dijelaskan dengan langkah-langkah berikut. Misalkan partisipan yang terlibat adalah Alice, Bob, dan Charlie.

1. **Pembentukan parameter publik.** Parameter publik yang digunakan sama seperti yang dijelaskan pada Protokol 1.
2. **Komputasi nilai rahasia tahap 1.** Alice memilih bilangan bulat  $a \in [1, q - 1]$ , Bob memilih bilangan bulat  $b \in [1, q - 1]$ , dan Charlie memilih bilangan bulat  $c \in [1, q - 1]$ . Selanjutnya Alice menghitung nilai  $A = g^a$ , Bob menghitung nilai  $B = g^b$ , dan Charlie menghitung nilai  $C = g^c$ .
3. **Pertukaran kunci secara publik tahap 1.** Alice mengirim nilai  $A$  kepada Bob, Bob mengirim nilai  $B$  kepada Charlie, dan Charlie mengirim nilai  $C$  kepada Alice.
4. **Komputasi nilai rahasia tahap 2.** Alice menghitung  $A_1 = C^a$ , Bob menghitung  $B_1 = A^b$ , dan Charlie menghitung nilai  $C_1 = B^c$ .
5. **Pertukaran kunci secara publik tahap 2.** Alice mengirim  $A_1$  kepada Bob, Bob mengirim nilai  $B_1$  kepada Charlie, dan Charlie mengirim nilai  $C_1$  kepada Alice.
6. **Komputasi kunci bersama.** Alice menghitung  $C_1^a$ , Bob menghitung  $A_1^b$ , dan Charlie menghitung  $B_1^c$ . Perhatikan bahwa  $C_1^a = A_1^b = B_1^c = g^{abc}$ .

Dari protokol yang telah dijelaskan buatlah skrip SageMath untuk melakukan simulasi pertukaran kunci Diffie-Hellman untuk tiga partisipan.

## 5.2 Pertukaran Kunci Diffie-Hellman dengan Kurva Eliptik

Pada Protokol 1 yang dijelaskan pada Subtopik 5.1 kita melihat bahwa struktur  $\mathbb{F}_q^*$  yang digunakan dapat diganti dengan grup komutatif lain, misalnya kurva eliptik yang telah dibahas pada Subtopik 2.7. Pada Latihan 2.7.2 kita melihat bahwa suatu titik  $P$  pada kurva eliptik  $E(\mathbb{F}_q)$  dikatakan sebagai titik pembangkit apabila setiap titik  $Q \in E(\mathbb{F}_q)$  dapat dinyatakan sebagai  $Q = \alpha P$  untuk suatu bilangan bulat positif  $\alpha$ . Protokol pertukaran kunci yang diperoleh dengan meninjau grup komutatif kurva Eliptik (terhadap operasi penjumlahan) dinamakan sebagai pertukaran kunci Diffie-Hellman berbasis kurva eliptik (*elliptic Diffie-Hellman key-exchange*, ECDH) yang dijelaskan pada Protokol 2.

**Protokol 2 (ECDH yang dirangkum dari Tabel 6.5 pada [2])** Protokol pertukaran kunci Diffie-Hellman berbasis kurva eliptik untuk dua partisipan dijelaskan sebagai berikut:

1. **Pembentukan parameter publik.** Pihak yang terpercaya memilih kurva eliptik  $E(\mathbb{F}_p)$  atas *field* hingga  $\mathbb{F}_p$  dan sebuah titik pembangkit  $P \in E(\mathbb{F}_p)$ . Jika titik pembangkit dari kurva eliptik  $E(\mathbb{F}_p)$  sulit dicari, maka kita dapat memilih titik  $P$  yang ordenya cukup besar.
2. **Komputasi nilai rahasia.** Alice memilih bilangan bulat  $n_A$  dan menghitung  $Q_A = n_A P$ . Secara serupa Bob memilih bilangan bulat  $n_B$  dan menghitung  $Q_B = n_B P$ . Pada praktiknya nilai  $n_A$  dan  $n_B$  adalah bilangan bulat pada rentang  $[1, \#E(\mathbb{F}_p)]$ , dengan  $\#E(\mathbb{F}_p)$  menyatakan banyaknya titik pada kurva eliptik  $E(\mathbb{F}_p)$ . Secara umum, ketika titik  $P$  memiliki orde  $k$  kita dapat memilih  $n_A$  dan  $n_B$  sebagai bilangan bulat pada rentang  $[1, k]$ .
3. **Pertukaran kunci secara publik.** Alice mengirimkan nilai  $Q_A$  kepada Bob dan secara serupa Bob mengirimkan nilai  $Q_B$  kepada Alice. Sebagaimana pada Protokol 1, pertukaran kunci pada ECDH juga dapat dilakukan secara publik.

- 
4. **Komputasi kunci bersama.** Untuk menghitung kunci bersama, Alice menghitung nilai dari  $n_A Q_B$  sedangkan Bob menghitung nilai dari  $n_B Q_A$ . Secara matematis  $n_A Q_B = n_B Q_A$  dan nilai ini adalah kunci bersama yang akan digunakan Alice dan Bob untuk komunikasi lebih lanjut.

Pada Protokol 2 Alice menghitung  $n_A Q_B$  sedangkan Bob menghitung  $n_B Q_A$ , tinjau bahwa

$$n_A Q_B = n_A n_B P = n_B n_A P = n_B Q_A. \quad (5.3)$$

Semua perhitungan pada ekspresi (5.3) dilakukan pada  $E(\mathbb{F}_p)$ . Definisi perkalian skalar pada kurva eliptik dapat dilihat pada Subtopik 2.7. Kita akan meninjau ilustrasi cara kerja Protokol 2 menggunakan contoh yang diperoleh dari [2, Halaman 317]. Misalkan kita menggunakan kurva eliptik  $E : y^2 = x^3 + 324x + 1287$  atas  $\mathbb{F}_{3851}$ . Perhatikan bahwa nilai dari  $4(324)^3 + 27(1287)^2 \neq 0$ .<sup>1</sup> Misalkan titik pembangkit yang digunakan adalah  $P = (920, 303) \in E(\mathbb{F}_{3851})$ . Kemudian misalkan Alice menggunakan nilai rahasia  $n_A = 1194$  sedangkan Bob menggunakan nilai rahasia  $n_B = 1759$ . Kita dapat memeriksa bahwa  $\#E(\mathbb{F}_{3851}) = 3928$ . Nilai publik dari Alice adalah  $Q_A = n_A P = 1194P = (2067, 2178)$  sedangkan nilai publik dari Bob adalah  $Q_B = n_B P = 1759P = (3684, 3125)$ . Setelah bertukar nilai  $Q_A$  dan  $Q_B$ , Alice dan Bob dapat menghitung kunci bersama yang hanya diketahui oleh mereka berdua. Alice menghitung  $n_A Q_B = (3347, 1242)$  sedangkan Bob menghitung nilai  $n_B Q_A = (3347, 1242)$ . Titik  $(3347, 1242)$  adalah kunci rahasia bersama yang selanjutnya dapat dipakai Alice dan Bob untuk berkomunikasi menggunakan sistemkripto simetrik. Simulasi pertukaran kunci menggunakan ECDH yang telah dijelaskan ini dapat dituliskan dalam skrip SageMath berikut:

```

F = GF(3851)
E = EllipticCurve(F, [324, 1287])
print('Kurva eliptik yang digunakan.')
print(E)
print('Banyaknya titik pada kurva eliptik: ', E.cardinality())
# Titik pembangkit dari E(F).
P = E(1583, 2701);
print('Titik pembangkit yang digunakan: ', P)
print('Orde dari titik pembangkit: ', P.order())
print('Perhitungan nilai publik.')
nA = 1194 # nilai rahasia dari Alice
QA = nA*P # nilai publik dari Alice
print('Nilai publik dari Alice: ', QA)
nB = 1759 # nilai rahasia dari Bob
QB = nB*P # nilai publik dari Bob
print('Nilai publik dari Bob: ', QB)
print('Alice dan Bob bertukar nilai QA dan QB.')
# Kalkulasi kunci bersama.
keyAlice = nA*QB
keyBob = nB*QA
print('Memeriksa kesamaan kunci yang digunakan.')
print(keyAlice == keyBob)

```

Untuk mengetahui banyaknya titik yang ada di suatu kurva eliptik  $E(\mathbb{F}_p)$  kita dapat memakai perintah  $E.\text{cardinality}()$ . Kemudian untuk memeriksa orde dari suatu titik  $P \in E(\mathbb{F}_p)$  kita dapat menggunakan perintah  $P.\text{order}()$ . Keluaran dari skrip ini adalah:

---

<sup>1</sup>Kurva eliptik  $E : y^2 = x^3 + ax + b$  yang kita gunakan haruslah memenuhi  $4a^3 + 27b^2 \neq 0$ . Lihat Definisi 2.7.1.

---

```

Kurva eliptik yang digunakan.
Elliptic Curve defined by y^2 = x^3 + 324*x + 1287 over Finite Field of size 3851
Banyaknya titik pada kurva eliptik: 3928
Titik pembangkit yang digunakan: (1583 : 2701 : 1)
Orde dari titik pembangkit: 1964
Perhitungan nilai publik.
Nilai publik dari Alice: (2247 : 2460 : 1)
Nilai publik dari Bob: (1538 : 2504 : 1)
Alice dan Bob bertukar nilai QA dan QB.
Memeriksa kesamaan kunci yang digunakan.
True

```

Dari hasil keluaran kita mengetahui bahwa  $\#E(\mathbb{F}_{3851}) = 3928$  dan titik  $P = (920, 303) \in E(\mathbb{F}_{3851})$  memiliki orde  $1964 = \#E(\mathbb{F}_{3851})/2$ . Sebagaimana penjelasan simulasi pertukaran kunci Diffie-Hellman untuk dua partisipan pada Subtopik 5.1, kita dapat memperumum simulasi ECDH dengan langkah-langkah berikut:

1. Masukan dari simulasi adalah sebuah kurva eliptik  $E(\mathbb{F}_p)$  dengan nilai  $\#E(\mathbb{F}_p)$  yang cukup besar. Misalkan kurva eliptik yang dipakai adalah  $E : y^2 = x^3 + ax + b$ , maka haruslah  $4a^3 + 27b^2 \neq 0$ .
2. Titik pembangkit  $P \in E(\mathbb{F}_p)$  dapat dikonstruksi dengan sintaks `E.gens() [0]`. Pada SageMath perintah `E.gens()` untuk kurva eliptik  $E(\mathbb{F}_p)$  memberikan sebuah tupel yang memuat titik-titik pembangkit pada  $E(\mathbb{F}_p)$ . Perhatikan bahwa kita cukup mengambil satu titik saja sehingga kita memakai perintah `E.gens() [0]`.
3. Kita dapat menghitung orde dari titik pembangkit  $P \in E(\mathbb{F}_p)$  dengan perintah `P.order()`. Misalkan orde dari titik  $P$  adalah  $\text{ord}(P)$ , maka kita dapat membangkitkan nilai rahasia dari Alice, yaitu  $n_A$ , sebagai bilangan bulat yang diambil acak pada rentang  $[1, \text{ord}(P)]$ . Nilai rahasia dari Bob, yaitu  $n_B$ , juga dibangkitkan dengan cara yang serupa. Perlu diingat bahwa SageMath bisa saja memberikan sebuah titik  $P$  dengan  $\text{ord}(P) < \#E(\mathbb{F}_p)$ . Alasan inilah yang menyebabkan langkah ini penting untuk dilakukan.
4. Nilai publik yang dikirimkan Alice adalah  $Q_A = n_A P$  sedangkan nilai publik yang dikirimkan Bob adalah  $Q_B = n_B P$ . Nilai ini adalah nilai yang saling dipertukarkan secara publik.
5. Untuk memperoleh kunci bersama Alice menghitung  $n_A Q_B$  sedangkan Bob menghitung  $n_B Q_A$ .

Simulasi ECDH menggunakan prinsip-prinsip yang telah dijelaskan dapat diterapkan menggunakan skrip SageMath berikut:

```

import random
# Field hingga yang dipakai.
F = GF(1009)
# Definisi kurva eliptik yang dipakai.
print('Definisi kurva eliptik yang dipakai: ')
E = EllipticCurve(F, [5, 1])
print(E)
print('Banyaknya titik pada kurva eliptik: ', E.cardinality())
# Mencari titik pembangkit atau titik dengan orde besar.
P = E.gens()[0]
print('Titik pembangkit: ', P)
print('Orde dari titik pembangkit: ', P.order())
# Simulasi pembangkitan nilai rahasia Alice dan Bob.
ordP = P.order()
nA = random.randint(1, ordP) # nilai rahasia Alice
nB = random.randint(1, ordP) # nilai rahasia Bob
# Kalkulasi nilai publik.
print('Nilai publik yang dipertukarkan.')
QA = nA * P # nilai publik Alice
print('Nilai publik dari Alice, QA: ', QA)
QB = nB * P # nilai publik Bob
print('Nilai publik dari Bob, QB: ', QB)
print('Alice dan Bob bertukar nilai QA dan QB.')
# Kalkulasi kunci bersama.
print('Kalkulasi kunci bersama.')
keyAlice = nA * QB
keyBob = nB * QA
print('Pemeriksaan kesamaan kunci.')
print(keyAlice == keyBob)

```

Pada skrip ini kita memakai kurva eliptik  $E : y^2 = x^3 + 5x + 1$  atas  $\mathbb{F}_{1009}$ . Kita memakai library `random` pada Python untuk membangkitkan nilai-nilai rahasia dari Alice (yaitu  $n_A$ ) dan Bob (yaitu  $n_B$ ). Salah satu keluaran dari hasil simulasi ini adalah:

```

Definisi kurva eliptik yang dipakai:
Elliptic Curve defined by y^2 = x^3 + 5*x + 1 over Finite Field of size 1009
Banyaknya titik pada kurva eliptik: 1039
Titik pembangkit: (553 : 684 : 1)
Orde dari titik pembangkit: 1039
Nilai publik yang dipertukarkan.
Nilai publik dari Alice, QA: (243 : 211 : 1)
Nilai publik dari Bob, QB: (321 : 982 : 1)
Alice dan Bob bertukar nilai QA dan QB.
Kalkulasi kunci bersama.
Pemeriksaan kesamaan kunci.
True

```

Dari hasil simulasi ini kita memiliki informasi penting berikut:

- Untuk  $E : y^2 = x^3 + 5x + 1$  atas  $\mathbb{F}_{1009}$  kita memiliki  $\#E(\mathbb{F}_{1009}) = 1039$ .
- Titik pembangkit yang digunakan pada simulasi adalah  $P = (553, 684) \in E(\mathbb{F}_{1009})$ . Dari hasil simulasi kita juga mengetahui bahwa  $\text{ord}(P) = \#E(\mathbb{F}_{1009})$ .
- Alice mengirimkan titik  $Q_A = (243, 211)$  kepada Bob, sedangkan Bob mengirimkan titik  $Q_B = (321, 982)$  kepada Alice. Nilai ini dapat saling dipertukarkan secara publik.

Sebagaimana pertukaran kunci Diffie-Hellman atas  $\mathbb{F}_q^*$ , salah satu cara untuk menyerang protokol ECDH (atau mengetahui kunci bersama yang pada akhirnya digunakan) adalah dengan menyelesaikan persamaan  $Q_A = n_A P$  dan  $Q_B = n_B P$  pada  $E(\mathbb{F}_p)$  untuk nilai  $n_A$  dan  $n_B$  dari

---

nilai-nilai  $Q_A$ ,  $Q_B$ ,  $P$ , dan kurva  $E(\mathbb{F}_p)$  yang diketahui. Masalah ini terkait dengan masalah logaritma diskrit pada kurva eliptik dan dijelaskan pada Permasalahan 5.2.1.

**Permasalahan 5.2.1** Misalkan  $E(\mathbb{F}_p)$  adalah sebuah kurva eliptik atas  $\mathbb{F}_p$  dan  $P, Q \in E(\mathbb{F}_p)$  adalah dua titik pada  $E(\mathbb{F}_p)$ . Masalah logaritma diskrit kurva eliptik (*elliptic curve discrete logarithm problem, ECDLP*) pada  $E(\mathbb{F}_p)$  dengan instance  $P$  dan  $Q$  adalah pencarian nilai  $n$  yang memenuhi

$$nP = Q. \quad (5.4)$$

Jika nilai  $n$  yang memenuhi (5.4) ada, maka nilai tersebut dikatakan sebagai logaritma kurva eliptik dari  $Q$  terhadap  $P$ .

Perhatikan bahwa Permasalahan 5.2.1 analog dengan Permasalahan 5.1.1 yang telah dijelaskan sebelumnya. Pada simulasi terakhir kita menggunakan kurva eliptik  $E : y^2 = x^3 + 5x + 1$  dan  $P = (553, 684)$  serta memiliki nilai  $n_A P = (243, 211)$  dan  $n_B P = (321, 982)$ . Untuk mencari kunci bersama yang sifatnya rahasia kita dapat menyelesaikan masalah logaritma diskrit kurva eliptik dari dua persamaan terakhir. Pada SageMath penyelesaian dari masalah logaritma diskrit kurva eliptik pada persamaan (5.4) dapat dilakukan menggunakan perintah `discrete_log(Q, P, P.order(), operation='+')`. Perintah ini pada dasarnya mencari nilai  $n \in [1, \text{ord}(P)]$  yang memenuhi persamaan (5.4) dengan meninjau operasi penjumlahan pada grup komutatif kurva eliptik yang dimiliki. Simulasi dari proses penyerangan ini dapat dilihat pada skrip berikut:

```
# Field hingga yang dipakai.
F = GF(1009)
# Definisi kurva eliptik yang dipakai.
print('Definisi kurva eliptik yang dipakai: ')
E = EllipticCurve(F, [5,1])
# Titik pembangkit yang dipakai.
P = E(553,684)
print('Titik pembangkit yang dipakai: ', P)
# Nilai-nilai publik yang diketahui.
QA = E(243,211)
print('Nilai publik Alice: ', QA)
QB = E(321,982)
print('Nilai publik Bob: ', QB)
# Pencarian nilai rahasia Alice dan Bob.
print('Pencarian nilai rahasia Alice dan Bob')
nA = discrete_log(QA,P,P.order(),operation='+')
print('Nilai rahasia Alice: ', nA)
nB = discrete_log(QB,P,P.order(),operation='+')
print('Nilai rahasia Bob: ', nB)
# Mencari kunci bersama yang sifatnya rahasia.
key = nA * nB * P
print('Kunci rahasia bersama: ', key)
```

Hasil keluaran dari skrip ini adalah:

---

```

Definisi kurva eliptik yang dipakai:
Titik pembangkit yang dipakai: (553 : 684 : 1)
Nilai publik Alice: (243 : 211 : 1)
Nilai publik Bob: (321 : 982 : 1)
Pencarian nilai rahasia Alice dan Bob
Nilai rahasia Alice: 738
Nilai rahasia Bob: 250
Kunci rahasia bersama: (849 : 390 : 1)

```

Dari skrip ini kita memperoleh titik  $(849, 390) \in E(\mathbb{F}_{1009})$  sebagai kunci rahasia bersama yang digunakan oleh Alice dan Bob. Nilai ini diperoleh dari nilai  $n_A n_B P$ , dengan nilai  $n_A = 738$  dan  $n_B = 250$ . Kedua nilai ini diperoleh menggunakan fungsi logaritma diskrit `discrete_log` yang telah dijelaskan sebelumnya. Sebagaimana pada protokol Diffie-Hellman yang dijelaskan pada Subtopik 5.1, pada praktiknya kita menggunakan kurva eliptik  $E(\mathbb{F}_p)$  yang memuat banyak titik dan titik pembangkit  $P$  yang memiliki orde yang cukup besar. Kajian detail mengenai hal ini dapat dilihat pada [2, Bab 6].

Meskipun penerapan ECDH terlihat agak lebih merepotkan dibandingkan dengan pertukaran kunci Diffie-Hellman biasa, ECDH menjadi alternatif yang lebih banyak dipakai karena alasan keamanan. Masalah logaritma diskrit pada  $\mathbb{F}_p^*$  yang telah dijelaskan sebelumnya dapat diselesaikan menggunakan metode *index calculus* (lihat [2, Subbab 3.8]) dalam waktu *subeksponensial* terhadap ukuran bit  $p$  yang digunakan. Ini berarti pencarian logaritma diskrit pada  $\mathbb{F}_p^*$  dapat diselesaikan dalam waktu  $\mathcal{O}(p^\epsilon)$  untuk setiap  $\epsilon > 0$ . Sedangkan, hingga saat ini algoritma paling efisien yang dapat menyelesaikan ECDLP memiliki kompleksitas waktu  $\mathcal{O}(\sqrt{p})$  atau  $\mathcal{O}(p^{1/2})$ . Ini berarti pencarian logaritma diskrit pada kurva eliptik  $E(\mathbb{F}_p)$  paling cepat diselesaikan dalam waktu eksponensial terhadap ukuran bit  $p$ .

### Latihan 5.2.2

1. Buatlah simulasi pada SageMath yang mempermudah pertukaran kunci yang ECDH untuk tiga partisipan. Gunakan ide konstruksi yang serupa dengan yang dijelaskan pada Latihan 5.1.2.
2. Kita melihat bahwa pada Protokol 2 nilai yang dipertukarkan adalah dua titik  $Q_A$  dan  $Q_B$  pada kurva eliptik  $E(\mathbb{F}_p)$ . Misalkan kita memakai kurva  $E : y^2 = x^3 + ax + b$ . Sebuah titik  $P = (x_P, y_P) \in E(\mathbb{F}_p)$  sebenarnya dapat direpresentasikan secara lebih singkat karena nilai  $(x_P, y_P)$  memenuhi hubungan  $y_P^2 = x_P^3 + ax_P + b$ . Jika nilai  $x_P$  diketahui kita dapat menentukan nilai  $y_P$  dengan cukup mudah (terutama jika kita bekerja di  $\mathbb{F}_p^*$  untuk  $p \equiv 3 \pmod{4}$ )<sup>2</sup>. Berdasarkan penjelasan pada [2, Contoh 6.21] nilai koordinat  $y_P$  dari titik  $P = (x_P, y_P)$  dapat diabaikan dalam transmisi. Kunci bersama yang digunakan adalah koordinat  $x$  dari titik yang dihasilkan pada Protokol 2. Buatlah sebuah simulasi pada SageMath yang menerapkan ECDH dengan memakai prinsip ini (pengiriman yang dilakukan hanya meninjau nilai koordinat  $x$  dari titik pada kurva eliptik). Kunci bersama yang dipakai adalah nilai koordinat  $x$  pada titik kurva eliptik yang dihasilkan. Untuk mempermudah, kurva eliptik yang digunakan adalah kurva atas  $\mathbb{F}_p$  dengan  $p \equiv 3 \pmod{4}$ .

## 5.3 Konversi String ke Bilangan Bulat dan Sebaliknya

Sebelum membahas mengenai sistemkripto RSA pada Subtopik 5.4 dan sistemkripto Elgamal pada Subtopik 5.5, kita terlebih dulu akan mengkaji cara merepresentasikan string pesan sebagai sebuah bilangan bulat. Hal ini berguna ketika kita perlu merepresentasikan string pesan

---

<sup>2</sup>Untuk  $p \equiv 3 \pmod{4}$ , solusi dari kongruensi  $x^2 \equiv a \pmod{p}$  adalah  $x \equiv a^{(p+1)/4} \pmod{p}$ .

---

sebagai elemen pada ring  $\mathbb{Z}_m$ . Pada kriptografi modern, pesan yang dikirimkan dapat memuat tanda baca maupun karakter non-alfanumerik. Oleh karena itu sebuah string pesan biasanya terlebih dulu dikonversi ke dalam sebuah *list* yang komponen-komponennya adalah nilai ASCII dari setiap karakter yang terdapat pada string pesan tersebut. Hal ini sebelumnya sudah dibahas pada Subtopik 3.1.6.

Pada dasarnya skema pengodean ASCII memetakan sebuah karakter (huruf, angka, tanda baca, maupun karakter non-alfanumerik lain) ke dalam bilangan bulat antara 0 sampai dengan 127 (inklusif) [24]. Oleh karenanya nilai ASCII dari suatu karakter juga dapat direpresentasikan sebagai string biner 7 bit. Pada Subtopik 3.1.6 kita telah mempelajari cara yang dapat dipakai pada Python untuk mengubah string pesan *mystring* menjadi *list mylist* yang komponen-komponennya adalah nilai ASCII dari setiap karakter pada *mystring* menggunakan perintah *mylist = [ord(x) for x in mystring]*. *List* yang dihasilkan memiliki komponen-komponen berupa bilangan bulat pada rentang 0 sampai dengan 127. Misalkan *list* yang dihasilkan adalah  $L = [a_0, a_1, \dots, a_{k-2}, a_{k-1}]$ . Kita dapat mengonversi  $L$  menjadi sebuah bilangan bulat  $n$  dengan definisi

$$n = a_{k-1} \cdot 128^{k-1} + a_{k-2} \cdot 128^{k-2} + \dots + a_1 \cdot 128 + a_0. \quad (5.5)$$

Perhatikan bahwa persamaan (5.5) merupakan representasi desimal dari sebuah bilangan dalam basis 128 yang dinyatakan sebagai  $(a_{k-1}, a_{k-2}, \dots, a_0, a_1)$ . Ini berarti konversi dari *list*  $L = [a_0, a_1, \dots, a_{k-2}, a_{k-1}]$  ke bilangan  $n$  dapat dilakukan dengan perintah *Integer(L, base=128)* sebagaimana dijelaskan pada Subtopik 2.3.2. Perhatikan bahwa jika string pesan memuat  $k$  karakter (yang nilai ASCII masing-masing karakternya adalah  $a_0, a_1, \dots, a_{k-1}$ ), maka bilangan bulat  $n$  yang berkorespondensi memenuhi

$$\begin{aligned} n &< 127 \cdot 128^{k-1} + 127 \cdot 128^{k-2} + \dots + 127 \cdot 128 + 127 \\ &= 127 \cdot (128^{k-1} + 128^{k-2} + \dots + 128 + 1) \\ &= 127 \cdot \frac{128^k - 1}{128 - 1} \end{aligned} \quad (5.6)$$

$$\begin{aligned} &= 127 \cdot \frac{128^k - 1}{127} \\ &= 128^k - 1 \end{aligned} \quad (5.7)$$

Ekspresi (5.6) diperoleh dari deret geometri  $1 + a + a^2 + \dots + a^{k-1} = (a^k - 1)/(a - 1)$ . Dari ekspresi (5.7) kita memperoleh bahwa  $n \leq 128^k - 1$  dengan  $k$  adalah banyaknya karakter pada string yang dikonversi menjadi suatu bilangan. Kita juga dapat menyimpulkan bahwa representasi bilangan bulat dari string yang memuat  $k$  karakter pasti tidak lebih dari  $128^k - 1$ . Sebagai ilustrasi, pesan yang memuat 10 karakter pasti direpresentasikan dalam bilangan bulat antara 0 hingga  $128^{10} - 1$  (inklusif).

Dari penjelasan yang diberikan kita dapat membuat algoritma untuk mengonversi sebuah string  $S$  menjadi sebuah bilangan bulat nonnegatif  $n$  menggunakan Algoritma 2.

---

**Algoritma 2** Konversi dari string ke bilangan bulat menggunakan nilai ASCII.

---

**Masukan:** String  $S$  yang karakter-karakternya dapat dipetakan ke dalam ASCII.

**Keluaran:** Bilangan bulat  $n$  yang berkorespondensi satu-satu dengan  $S$ .

1. Konstruksi *list*  $L$  yang komponen-komponennya adalah nilai ASCII dari masing-masing karakter yang bersesuaian pada string  $S$ . Misalkan  $S$  memuat  $k$  karakter dan  $L = [a_0, a_1, \dots, a_{k-1}]$ .
  2. Ubah *list*  $L$  ke dalam bilangan bulat  $n$  dengan aturan yang dijelaskan pada (5.5).
-

---

Pada SageMath Algoritma 2 dapat diterapkan dengan fungsi `string_to_int` yang dijelaskan pada skrip berikut:

```
def string_to_int(mystring):
    listm = [ord(x) for x in mystring]
    myint = Integer(listm,base=128)
    return myint
```

Skrip ini mengubah string `mystring` ke dalam bilangan bulat `myint` menggunakan langkah-langkah pada Algoritma 2. Sebagai contoh, jika kita memiliki string `HALO`, maka *list L* yang terbentuk adalah  $[72, 65, 76, 79]$ . Nilai ini menghasilkan bilangan bulat  $n = 72 + 65 \cdot 128 + 76 \cdot 128^2 + 79 \cdot 128^3 = 166928584$ . Proses serupa juga dapat diterapkan pada string lain.

Perhatikan bahwa setiap bilangan bulat positif  $n$  dinyatakan secara tunggal dalam basis 128 (lihat Teorema 2.3.2 untuk lebih jelas). Akibatnya kita dapat “membalik” proses pada Algoritma 2 untuk memperoleh sebuah string dari suatu bilangan bulat tertentu. Proses ini dijelaskan dalam Algoritma 3.

---

### Algoritma 3 Konversi dari bilangan bulat ke string menggunakan nilai ASCII.

---

**Masukan:** Bilangan bulat  $n$ .

**Keluaran:** String  $S$  yang berkorespondensi satu-satu dengan  $n$ .

1. Konstruksi *list L* =  $[a_0, a_1, \dots, a_{k-1}]$ , di sini  $a_0, a_1, \dots, a_{k-1}$  adalah nilai-nilai yang diperoleh dari  $n$  berdasarkan aturan pada (5.5).
  2. Ubah masing-masing nilai  $a_0, a_1, \dots, a_{k-1}$  menjadi karakter sesuai nilai ASCII-nya masing-masing.
  3. Gabungkan karakter yang diperoleh pada langkah sebelumnya menjadi suatu string.
- 

Kita dapat menggunakan metode yang dijelaskan pada Subtopik 3.1.6 untuk mengubah *list* yang memuat nilai-nilai ASCII (bilangan bulat antara 0 sampai dengan 127, inklusif) menjadi *list* yang memuat karakter ASCII yang nilainya bersesuaian. Pada Python hal ini dilakukan dengan perintah `chr`. Algoritma 3 dapat diterapkan dengan skrip berikut pada SageMath:

```
def int_to_string(n):
    L = Integer(n).digits(base=128)
    S = [chr(x) for x in L]
    return ''.join(S)
```

Skrip ini menerima masukan sebuah bilangan bulat positif  $n$  lalu membangun *list L* yang merepresentasikan nilai  $n$  dalam basis 128. Selanjutnya skrip ini membuat *list S* yang komponen-komponennya adalah karakter-karakter dengan nilai ASCII yang muncul sebagai entri-entri *list L*. Hasil dari *list S* kemudian digabungkan dengan *method .join* tanpa mempertimbangkan pemisah antar string. Sebagai contoh, masukan berupa bilangan  $n = 166928584$  pertama-tama akan diubah menjadi *list L* =  $[72, 65, 76, 79]$  yang merupakan representasi dari  $n$  dalam basis 128. Selanjutnya kita membangun *list S* =  $['H', 'A', 'L', 'O']$  yang masing-masing komponennya adalah karakter dengan nilai-nilai ASCII yang ada pada *list L*. Setiap komponen dari *list S* kemudian digabungkan dengan *method .join* sehingga diperoleh string `HALO`.

**Latihan 5.3.1** Pada kriptografi asimetrik kita sering kali merepresentasikan pesan sebagai sebuah elemen pada ring  $\mathbb{Z}_m$  atau *field*  $\mathbb{F}_q$ . Dalam sebuah transmisi, setiap pesan yang dikirimkan dapat dibuat berkorespondensi satu-satu dengan sebuah elemen pada ring atau *field* yang ditinjau. Misalkan kita memiliki sebuah ring atau *field* dengan  $q$  elemen berbeda. Pada latihan ini tugas Anda adalah menentukan (secara teori) panjang maksimal (banyaknya karakter) dari pesan yang dapat dikirimkan jika sebuah pesan harus direpresentasikan sebagai elemen pada ring atau *field* dengan  $q$  elemen. Pengodean yang digunakan adalah pengodean ASCII yang memetakan sebuah karakter ke bilangan bulat antara 0 sampai dengan 127 (inklusif). Untuk

---

mempermudah dalam menjawab soal ini, Anda dapat mencoba dua soal yang lebih mudah sebagai berikut:

1. Tentukan bilangan bulat terkecil  $n$  yang dapat digunakan untuk merepresentasikan **semua pesan** dengan panjang 20 (atau kurang) berdasarkan aturan pengodean yang dijelaskan pada Algoritma 2 dan Algoritma 3.
2. Tentukan banyak maksimal dari karakter berbeda (atau panjang pesan) dari pesan yang representasi bilangan bulatnya tidak lebih dari  $10^{18}$  apabila aturan pengodean yang digunakan mengikuti aturan pada Algoritma 2 dan Algoritma 3.

## 5.4 Sistemkripto RSA

Pada subtopik ini kita akan membahas mengenai sistemkripto RSA sebagaimana dijelaskan pada [9, Bab 7], [2, Subbab 3.2], [25, Subbab 8.2], dan [26]. RSA merupakan huruf pertama dari ketiga nama pengusulnya, yaitu Ronald Rivest, Adi Shamir, dan Leonard Adleman. Sistemkripto ini pertama kali diperkenalkan secara publik pada tahun 1978 [27].

Sebenarnya ide maupun penerapan dari skema pertukaran kunci maupun sistemkripto asimetrik sudah ada sejak sebelum protokol Diffie-Hellman maupun sistemkripto RSA disebarluaskan secara publik. Konsep kriptografi kunci publik sudah diketahui oleh James Ellis yang bekerja di dinas komunikasi pemerintah Inggris (*British Government Communication Headquarters, GCHQ*) pada tahun 1969 [2, Subbab 2.1]. Penemuan ini dirahasiakan oleh pemerintah Inggris hingga tahun 1997 ketika Ellis sudah meninggal dunia. Pegawai GCHQ yang lain, yaitu Malcolm Williamson, menemukan apa yang saat ini kita namakan sebagai pertukaran kunci Diffie-Hellman sebelum tahun 1976. Kemudian Clifford Cocks, yang juga merupakan pegawai GCHQ, menemukan apa yang sekarang kita kenal sebagai sistemkripto RSA sebelum tahun 1978. Semua penemuan-penemuan tersebut dirahasiakan oleh GCHQ dan tidak disebarluaskan secara publik hingga tahun 1990-an [2, Subbab 2.1].

Subtopik ini dibagi menjadi tiga pokok bahasan. Pada bagian pertama kita akan mengkaji protokol RSA yang digunakan untuk proses enkripsi dan dekripsi saja. Pada bagian kedua kita akan membahas skema tanda tangan digital menggunakan RSA (hal ini juga dikenal sebagai RSA *digital signature scheme*). Di bagian terakhir kita akan membahas beberapa teknik sederhana yang dapat digunakan untuk menyerang RSA.

### 5.4.1 Protokol RSA untuk Enkripsi dan Dekripsi

Sistemkripto RSA merupakan sistemkripto asimetrik yang memungkinkan pengirim dan penerima pesan menggunakan kunci yang berbeda untuk merahasiakan pesannya. Misalkan terdapat dua pihak, yaitu Alice dan Bob. Alice ingin mengirim pesan kepada Bob menggunakan kunci yang bersifat publik, kunci ini disediakan oleh Bob. Pada dasarnya siapa pun yang ingin mengirim pesan kepada Bob mengetahui kunci ini. Dengan melakukan enkripsi Alice berharap agar tidak seorang pun kecuali Bob yang dapat mengetahui pesan yang dikirimkannya. Untuk membuka pesan tersandi (*encrypted message*) yang dikirim oleh Alice, Bob tidak menggunakan kunci yang sama seperti yang digunakan Alice (karena jika demikian, maka semua orang dapat mengetahui pesan Alice). Bob memiliki kunci rahasia (atau kunci privat, *private key*) yang hanya diketahui olehnya. Secara ringkas penyandian berbasis RSA dijelaskan pada Protokol 3.

**Protokol 3 (Protokol enkripsi dan dekripsi RSA yang dirangkum dari Tabel 3.1 pada [2])**  
Protokol sistemkripto RSA untuk penyandian pesan dijelaskan sebagai berikut:

- 
1. **Pembuatan kunci.** Pembuatan kunci dilakukan oleh Bob. Pertama Bob memilih dua bilangan prima  $p$  dan  $q$  yang berbeda dan cukup besar. Bob juga mendefinisikan sebuah bilangan bulat positif  $e$  dengan sifat  $\gcd(e, (p-1)(q-1)) = 1$ . Nilai dari  $N = pq$  dan  $e$  diberikan secara publik. Pasangan  $(N, e)$  selanjutnya dinamakan sebagai kunci publik. Untuk meringkas penulisan, kita menuliskan  $(p-1)(q-1) = \phi(N)$ . Penyederhanaan notasi ini sesuai dengan definisi fungsi phi Euler untuk  $N = pq$  yang ada pada persamaan (2.6) maupun persamaan (2.7) yang sudah dibahas pada Subtopik 2.3.2.
  2. **Enkripsi.** Misalkan Alice ingin mengirim pesan kepada Bob menggunakan kunci publik  $(N, e)$ . Pesan dinyatakan dalam sebuah bilangan bulat positif  $m$ . Pada praktiknya haruslah  $1 < m < N$ . Misalkan  $c$  adalah teks sandi dari  $m$ , maka Alice menghitung  $c = m^e \pmod{N}$ . Nilai  $c$  kemudian dikirim kepada Bob.
  3. **Dekripsi.** Untuk melakukan dekripsi Bob mencari bilangan bulat positif  $d$  dengan sifat  $ed \equiv 1 \pmod{\phi(N)}$ . Perhatikan bahwa karena  $\gcd(e, \phi(N)) = \gcd(e, (p-1)(q-1)) = 1$ , maka nilai  $d$  dijamin ada. Untuk memperoleh pesan dari Alice, Bob menghitung  $m' = c^d \pmod{N}$ . Secara matematis nilai  $m' \equiv m \pmod{N}$ . Jika nilai  $m'$  dan  $m$  berada pada rentang  $[1, N-1]$ , maka haruslah  $m' = m$ .

Dalam RSA teks pesan (*plaintext*) maupun teks sandi (*ciphertext*) merupakan elemen ring  $\mathbb{Z}_N$ . Perhatikan bahwa pada Protokol 3 seorang penyadap (*eavesdropper, wiretapper*) mungkin mengetahui nilai  $c$ . Penyadap tersebut dapat mencari nilai  $m$  (pesan rahasia yang dikirimkan) yang memenuhi  $c \equiv m^e \pmod{N}$  dengan memanfaatkan nilai  $N$  dan  $e$  yang diketahui. Permasalahan ini secara matematis dinamakan sebagai permasalahan akar modular (*modular root problem*). Secara formal hal ini dijelaskan pada Permasalahan 5.4.1.

**Permasalahan 5.4.1** *Misalkan  $c$ ,  $e$ , dan  $N$  adalah bilangan bulat, permasalahan akar modular ke- $e$  pada modulo  $N$  (*e-th modular root problem in modulo N*) adalah pencarian nilai  $x$  yang memenuhi  $x^e \equiv c \pmod{N}$ .*

Seorang penyadap juga dapat mengetahui pesan rahasia  $m$  jika dia dapat menghitung nilai  $p$  dan  $q$  dari nilai  $N$  yang diketahui. Dengan perkataan lain jika penyadap mengetahui faktorisasi prima dari  $N$ , maka dia dapat mengetahui nilai  $m$  dengan mudah. Perhatikan bahwa jika nilai  $p$  dan  $q$  diketahui, maka penyadap dapat mengetahui nilai  $\phi(N) = (p-1)(q-1)$ . Karena nilai  $e$  diketahui secara publik, maka nilai  $d$  dapat diperoleh menggunakan *extended Euclid algorithm* sebagaimana dijelaskan pada Subtopik 2.1.2 (karena  $\gcd(e, \phi(N)) = 1$ ). Ketika nilai  $d$  diketahui maka penyadap dapat menghitung nilai  $m \equiv c^d \pmod{N}$ .

Perhatikan bahwa jika faktorisasi nilai  $N$  sulit dilakukan, maka penyadap juga sulit untuk memperoleh nilai  $d$ . Kita melihat bahwa pencarian solusi kongruensi  $x^e \equiv c \pmod{N}$  dapat dilakukan dengan mudah jika kita memiliki informasi tambahan (dalam hal ini nilai  $d$ ).

Pada protokol RSA yang dijelaskan pada Protokol 3, nilai  $e$  dikatakan sebagai pangkat (atau eksponen) enkripsi (*encryption exponent*) dan nilai  $d$  dikatakan sebagai pangkat (atau eksponen) dekripsi (*decryption exponent*). Nilai  $e$  dan  $d$  memenuhi kongruensi

$$\begin{aligned} ed &\equiv 1 \pmod{\phi(N)} \\ ed &\equiv 1 \pmod{(p-1)(q-1)}. \end{aligned} \tag{5.8}$$

Perhatikan bahwa pada kongruensi (5.8) nilai  $d$  adalah invers dari  $e$  pada  $\mathbb{Z}_{\phi(N)}$ . Untuk setiap nilai  $e \in \mathbb{Z}_{\phi(N)}$  dengan  $\gcd(e, \phi(N)) = 1$  hanya ada satu nilai  $d$  yang memenuhi kongruensi (5.8) dalam  $\mathbb{Z}_{\phi(N)}$ . Dengan perkataan lain untuk setiap pangkat enkripsi  $e \in \mathbb{Z}_{\phi(N)}$  hanya ada satu pangkat dekripsi  $d \in \mathbb{Z}_{\phi(N)}$  yang bersesuaian dengannya.

Pada RSA, enkripsi dari pesan  $m$  dengan pangkat enkripsi  $e$  adalah kalkulasi  $m^e \pmod{N}$ . Kemudian, proses dekripsi dari sandi  $c$  dengan pangkat dekripsi  $d$  adalah kalkulasi  $c^d \pmod{N}$ .

Secara teori, komputasi keduanya dapat dilakukan dalam waktu  $\mathcal{O}(\log N)$  menggunakan algoritma *fast modular exponentiation*.<sup>3</sup> Ini berarti kalkulasi teks sandi maupun perolehan kembali teks pesan dapat dilakukan dengan cukup cepat. Untuk membuktikan kebenaran proses enkripsi dan dekripsi dari RSA kita perlu menunjukkan bahwa  $(m^e)^d \equiv m \pmod{N}$ . Karena  $N = pq$  dan  $p \neq q$  (yang berarti  $\gcd(p, q) = 1$ ) maka berdasarkan Teorema Sisa Tiongkok (Teorema 2.3.4) kita dapat membuktikan bahwa  $m^{ed} \equiv m \pmod{N}$  dengan membuktikan bahwa  $m^{ed} \equiv m \pmod{p}$  dan  $m^{ed} \equiv m \pmod{q}$  secara terpisah. Untuk membuktikan hal ini kita juga akan memakai Teorema Kecil Fermat (*Fermat's Little Theorem*) yang menyatakan bahwa  $a^{p-1} \equiv 1 \pmod{p}$  untuk setiap bilangan prima  $p$  dan bilangan bulat  $a$  yang tidak habis bagi  $p$ . Perhatikan bahwa:

1. Jika  $m \equiv 0 \pmod{p}$  atau  $m$  habis dibagi  $p$ , maka  $m$  adalah kelipatan dari  $p$ . Akibatnya  $m^{ed}$  juga kelipatan dari  $p$ . Ini berarti  $m^{ed} \equiv 0 \pmod{p}$ . Jadi diperoleh  $m^{ed} \equiv m \pmod{p}$ .
2. Jika  $m \not\equiv 0 \pmod{p}$  maka berdasarkan Teorema kecil Fermat  $m^{p-1} \equiv 1 \pmod{p}$ . Karena  $\gcd(e, (p-1)(q-1)) = 1$  maka  $ed + t(p-1)(q-1) = 1$  untuk suatu  $t \in \mathbb{Z}$ . Kita memiliki  $ed = 1 - t(p-1)(q-1)$ . Akibatnya

$$\begin{aligned}
 m^{ed} &\equiv m^{1-t(p-1)(q-1)} \pmod{p} \\
 &\equiv m^1 \cdot m^{(p-1)(q-1) \cdot (-t)} \pmod{p} \\
 &\equiv m^1 \cdot (m^{p-1})^{(q-1) \cdot (-t)} \pmod{p} \\
 &\equiv m^1 \cdot 1 \pmod{p} \\
 &\text{menggunakan Teorema Kecil Fermat } m^{p-1} \equiv 1 \pmod{p} \\
 &\equiv m \pmod{p}
 \end{aligned} \tag{5.9}$$

Dengan metode yang serupa seperti pada (5.9) kita dapat menunjukkan bahwa  $m^{ed} \equiv m \pmod{q}$ . Dengan menggabungkan kedua hasil ini menggunakan Teorema Sisa Tiongkok kita memperoleh  $m^{ed} \equiv m \pmod{pq}$  atau  $m^{ed} \equiv m \pmod{N}$ . Selain itu pada [2, Proposisi 3.5] dijelaskan bahwa solusi dari kongruensi  $x^e \equiv c \pmod{pq}$  untuk  $e$  yang memenuhi  $\gcd(e, (p-1)(q-1)) = 1$  pasti ada dan tunggal. Nilai  $x$  yang merupakan solusi adalah  $x \equiv c^d \pmod{pq}$ . Ini berarti proses enkripsi maupun dekripsi pada RSA adalah sebuah pemetaan yang injektif pada  $\mathbb{Z}_N$ . Artinya tidak ada dua pesan berbeda yang dienkripsi menjadi sebuah sandi yang sama dan tidak ada dua sandi berbeda yang didekripsi menjadi sebuah pesan yang sama.

Dalam RSA dengan kunci publik  $(N, e)$  pesan yang digunakan adalah  $m$  yang memenuhi  $1 < m < N$ . Nilai  $m = 0$  tidak digunakan sebagai pesan karena  $0^e \equiv 0 \pmod{N}$  sedangkan nilai  $m = 1$  tidak digunakan karena  $1^e \equiv 1 \pmod{N}$ . Kemudian pada praktiknya RSA yang aman digunakan menggunakan nilai  $N = pq$  dengan  $p$  dan  $q$  yang setidaknya terdiri dari seratus digit (nilai  $p$  dan  $q$  merupakan bilangan prima yang tidak kurang dari  $10^{99}$ ). Namun pada tutorial ini kita hanya akan meninjau contoh RSA untuk nilai  $p$  dan  $q$  yang tidak terlalu besar. Contoh berikut diadaptasi dari [2, Contoh 3.9]. Tahap pertama pada RSA adalah pembuatan parameter publik, misalkan Bob memilih  $p = 1223$  dan  $q = 1987$  sehingga diperoleh  $N = p \cdot q = 2430101$ . Di sini Bob juga dapat dengan mudah menghitung  $\phi(N) = (p-1)(q-1) = 2426892$ . Untuk membangun pangkat enkripsi Bob memilih  $e \in \mathbb{Z}_{\phi(N)}$  dengan  $\gcd(e, \phi(N)) = 1$ . Misalkan Bob memilih  $e = 948047$ . Kita dapat memeriksa bahwa  $\gcd(948047, 2426892) = 1$ . Bob mengumumkan kunci publiknya sebagai pasangan  $(N, e) = (2430101, 948047)$ . Pada tahap ini Bob juga dapat membangkitkan pangkat dekripsinya menggunakan *extended Euclid's*

---

<sup>3</sup>Kajian mengenai *fast modular exponentiation* atau *fast powering algorithm* dapat dilihat pada [2, Subbab 1.3.2].

---

*algorithm* (hal ini dapat dilakukan menggunakan fungsi `xgcd( . . . , . . . )` pada SageMath). Bob memperoleh  $d = 1051235$ .

Misalkan Alice akan mengirim sebuah pesan. Alice mengonversi pesannya ke dalam bilangan pada  $\mathbb{Z}_N \setminus \{0, 1\}$ . Pada praktiknya Alice dapat menerapkan Algoritma 2 yang dijelaskan pada Subtopik 5.3. Misalkan Alice memiliki pesan  $m = 1070777$ . Alice menggunakan kunci publik  $(N, e) = (2430101, 948047)$  untuk mengenkripsi  $m$ . Dalam hal ini Alice menghitung  $c = m^e \pmod{N}$ , sehingga diperoleh  $c = 1070777^{948047} \equiv 1473513 \pmod{2430101}$ . Nilai  $c = 1473513$  kemudian dikirim oleh Alice kepada Bob.

Untuk mendekripsi pesan Alice, Bob menghitung  $m' = c^d \pmod{N}$  dengan  $d$  adalah pangkat dekripsi yang telah dihitung sebelumnya. Bob memiliki

$$m' = 1473513^{1051235} \equiv 1070777 \pmod{2430101}.$$

Simulasi protokol RSA pada SageMath dapat dijalankan sebagai berikut. Pertama untuk menentukan bilangan prima  $p$  dan  $q$  yang berbeda dan cukup besar kita dapat menggunakan fungsi `prime_list` yang telah dibuat pada Subtopik 2.1.3. Misalkan kita ingin memastikan bahwa nilai  $p$  dan  $q$  memenuhi  $10^9 - 1000 \leq p, q \leq 10^9$  maka kita dapat memakai skrip berikut:

```
import random
def prime_list(a,b):
    L = []
    for i in range(a,b+1):
        if Integer(i).is_prime() == True: L.append(Integer(i))
    return L
p = random.choice(prime_list(10^9-1000,10^9)); print(p)
q = random.choice(prime_list(10^9-1000,10^9)); print(q)
```

Library `random` pada Python kita pakai untuk menggunakan perintah `random.choice` agar kita dapat mengambil sebuah elemen dari suatu *list* secara acak. Kita harus memastikan bahwa  $p \neq q$ . Jika  $p = q$  maka kita dapat mengulang eksekusi skrip yang sebelumnya telah dilakukan.

Misalkan kita memiliki  $p = 999999667$  dan  $q = 999999337$ . Kita dapat menggunakan SageMath untuk mebangkitkan parameter publik  $(N, e)$ . Nilai  $N = p \cdot q$  dan nilai  $e$  diambil secara acak dari nilai-nilai pada  $\mathbb{Z}_N$  yang relatif prima dengan  $\phi(N) = (p-1)(q-1)$ . Ketika nilai  $\phi(N)$  tidak terlalu besar kita dapat mencari nilai  $e$  yang tidak memiliki faktor prima yang merupakan faktor prima dari  $(p-1)(q-1)$ . Misalkan kita memiliki  $p = 999999667$  dan  $q = 999999337$ , kita dapat mencari faktor prima dari  $(p-1)(q-1)$  dengan skrip berikut:

```
p = 999999667
q = 999999337
phi = euler_phi(p*q);
print(phi);
print((p-1)*(q-1))
factors = factor(phi); print(factors)
L = []
# mencari faktor prima dari (p-1)(q-1)
for i in range(len(factors)):
    L.append(factors[i][0])
print(L)
```

Pada skrip ini kita memakai fungsi `euler_phi( . . . )` untuk mencari nilai fungsi phi Euler dari suatu bilangan bulat positif. Fungsi ini telah dibahas pada Subtopik 2.3.2 dan definisi matematisnya dapat dilihat pada persamaan (2.6) dan (2.7). Hasil keluaran dari skrip sebelumnya adalah:

```

999999002000221776
999999002000221776
2^4 * 3^3 * 7 * 19 * 23 * 37 * 53 * 257 * 1501501
[2, 3, 7, 19, 23, 37, 53, 257, 1501501]

```

Hasil ini berarti kita tidak boleh memilih pangkat enkripsi  $e$  yang faktor primanya adalah salah satu di antara 2, 3, 7, 19, 23, 37, 53, 257, dan 150151. Kita dapat memilih  $e = 101 \cdot 103 = 10403$ . Pada praktiknya ketika nilai  $\phi(N) = (p-1)(q-1)$  sangat besar (misalnya lebih dari 200 digit) maka untuk memperoleh  $e$  kita dapat membangkitkan bilangan secara acak pada  $\mathbb{Z}_{\phi(N)}$  (dapat berupa bilangan prima pada  $\mathbb{Z}$ ) dan memeriksa apakah bilangan tersebut relatif prima dengan  $(p-1)(q-1)$ .

Jika diberikan bilangan prima  $p$  dan  $q$  yang berbeda serta sebuah bilangan  $e \in \mathbb{Z}_N$  kita dapat membangkitkan dan memeriksa parameter kunci publik pada RSA dengan skrip berikut:

```

# fungsi untuk membangkitkan dan memeriksa parameter publik
# nilai p dan q diasumsikan prima dan berbeda
def cekparamRSA(p,q,e):
    N = p*q
    phi = euler_phi(N)
    if gcd(e,phi) == 1:
        return (N,e)
    else:
        return('eksponen tidak valid')

```

Kita dapat menggunakan fungsi ini untuk memeriksa apakah sebuah nilai  $e$  relatif prima dengan  $\phi(N) = (p-1)(q-1)$ . Jika  $e$  relatif prima dengan  $\phi(N)$  maka fungsi akan mengembalikan nilai  $(N, e)$ . Jika tidak maka fungsi mengembalikan string eksponen tidak valid. Misalkan kita ingin memeriksa apakah 1961 dan 10403 merupakan eksponen yang dapat digunakan pada RSA untuk nilai  $p = 999999667$  dan  $q = 999999337$ , kita dapat melakukan hal berikut:

```

print(cekparamRSA(999999667,999999337,1961))
print(cekparamRSA(999999667,999999337,10403))

```

Keluaran dari skrip ini adalah:

```

eksponen tidak valid
(999999004000220779, 10403)

```

Nilai  $e = 1961$  tidak dapat dipakai ketika  $p = 999999667$  dan  $q = 999999337$  karena  $1961 = 37 \cdot 53$  dan 37 maupun 53 adalah faktor prima dari  $(p-1)(q-1)$ . Proses enkripsi dari suatu pesan  $m$  yang dinyatakan sebagai elemen  $\mathbb{Z}_N$  merupakan perpangkatan modulo  $N$  dan dapat dilakukan dengan skrip berikut:

```

def encRSA(m,e,N):
    return pow(m,e,N)

```

Misalkan kita ingin menentukan hasil enkripsi  $m = 12332111999888777$  dengan  $e = 10403$  pada  $\mathbb{Z}_N$  yang telah dibahas sebelumnya. Kita dapat menggunakan skrip berikut:

```

c = encRSA(12332111999888777,10403,999999004000220779)
print(c)

```

Perhatikan bahwa kita memiliki  $1 < m < N$ . Hasil dari skrip ini adalah sebuah bilangan  $c = 507424141543035091$  yang dijadikan sebagai teks sandi. Teks sandi ini dikirimkan ke Bob. Untuk mengetahui teks pesan, Bob harus menghitung  $c^d \pmod{N}$ . Nilai  $d$  dapat dihitung menggunakan fungsi `xgcd` pada SageMath. Bob dapat mengetahui nilai teks pesan dengan skrip berikut:

```

def decRSA(c, e, p, q):
    phi = euler_phi(p*q)
    L = xgcd(e, phi)
    d = L[1] %phi
    return pow(c, d, p*q)

```

Pada proses dekripsi perhatikan bahwa Bob mengetahui nilai  $p$  dan  $q$  yang digunakan. Untuk mendekripsi sandi  $c = 507424141543035091$  Bob dapat menggunakan skrip berikut:

```

plaintext = decRSA(507424141543035091, 10403, 999999667, 999999337)
print(plaintext)

```

Hasil dari skrip ini adalah pesan awal 12332111199988777.

Pada Subtopik 5.3 kita telah melihat sebuah metode untuk memetakan sebuah string menjadi sebuah bilangan bulat secara injektif. Pembahasan mengenai hal ini dapat dilihat pada [9, Bab 7]. Kita dapat menggabungkan metode tersebut dengan proses enkripsi dan dekripsi pada RSA agar pesan yang dikirimkan dapat berupa string. Pemilihan bilangan prima  $p$  dan  $q$  harus dilakukan dengan mempertimbangkan banyaknya karakter pesan yang dapat dikodekan ke dalam bilangan bulat. Lihat Latihan 5.3.1 untuk lebih jelas.

### Latihan 5.4.2

- Buatlah sebuah fungsi `encRSAString(P, e, N)` yang melakukan enkripsi pada string masukan  $P$  yang merepresentasikan teks pesan (*plaintext*) dalam format string dengan parameter publik ( $N, e$ ) menggunakan RSA dengan memanfaatkan Algoritma 2. Hasil enkripsi adalah sebuah bilangan bulat  $c \in [1, N - 1]$  yang merepresentasikan teks sandi (*ciphertext*) dari string  $S$ . Sebagai ilustrasi, masukan berikut:

```

ciphera = encRSAString('SageMath', 10403, 999999004000220779); print(ciphera)
cipherb = encRSAString('Aljabar', 999999323, 999999004000220779); print(cipherb)
cipherc = encRSAString('Oke123', 100000007, 999999004000220779); print(cipherc)

```

memberikan keluaran:

```

2041895125359412
16827134532710049
731548682203395148

```

- Buatlah sebuah fungsi `decRSAString(c, e, p, q)` yang melakukan dekripsi pada sebuah bilangan bulat  $c \in [1, N - 1]$  yang merepresentasikan suatu teks sandi (*ciphertext*) dengan memanfaatkan parameter  $(e, p, q)$ , dengan  $e$  adalah eksponen enkripsi serta  $p$  dan  $q$  adalah parameter rahasia pada RSA. Hasil dekripsi adalah sebuah string  $P$  yang merupakan teks pesan (*plaintext*) yang bersesuaian dengan  $c$  dengan memanfaatkan Algoritma 3. Sebagai ilustrasi, masukan berikut:

```

plaina = decRSAString(2041895125359412, 10403, 999999337, 999999667); print(plaina)
plainb = decRSAString(16827134532710049, 999999323, 999999337, 999999667); print(plainb)
plainc = decRSAString(731548682203395148, 100000007, 999999337, 999999667); print(plainc)

```

memberikan keluaran:

```

SageMath
Aljabar
Oke123

```

### 5.4.2 Skema Tanda Tangan Digital RSA

Pada Subtopik 5.4.1 kita telah melihat cara untuk mengirimkan pesan secara rahasia yang menggunakan kunci enkripsi dan dekripsi yang berbeda. Metode ini menyelesaikan permasalahan ketika pengirim dan penerima pesan tidak dapat bertemu untuk menyetujui suatu kunci yang sama. Namun, pada Protokol 3 Bob sebagai penerima pesan diasumsikan sudah mengenal

---

Alice sebagai pengirim pesan. Masalah timbul jika Bob belum mengenal Alice, atau bila seseorang (selain Alice) mencoba untuk berpura-pura menjadi Alice dan mengirimkan pesan kepada Bob. Skema tanda tangan digital dapat digunakan untuk mengatasi masalah ini.

Sebelum Alice mengirimkan pesan terenkripsi, ia terlebih dulu membuat tanda tangan digital yang dikirimkan bersamaan dengan hasil enkripsi pesan tersebut. Tanda tangan digital ini dibuat menggunakan sebuah kunci rahasia yang hanya diketahui oleh Alice. Alice juga membuat kunci publik yang dapat digunakan siapa pun untuk memverifikasi pesan yang dikirimkannya. Kunci publik ini digunakan untuk memeriksa apakah sebuah pesan terenkripsi benar-benar dikirimkan oleh Alice atau tidak. Tentunya kunci rahasia yang digunakan Alice berbeda dengan kunci publik yang akan digunakan untuk verifikasi.

Sebuah penandatanganan digital dapat dianalogikan sebuah fungsi atau algoritma  $\text{sign}$  dengan masukan  $D$  yang berupa dokumen/pesan dan  $k_{priv}$  yang merupakan kunci privat (kunci rahasia) yang digunakan untuk menandatangani dokumen  $D$ . Secara matematis proses penandatanganan sebuah dokumen  $D$  menggunakan kunci privat  $k_{priv}$  yang menghasilkan tanda tangan  $Sig$  dinyatakan dengan persamaan

$$\text{sign}(D, k_{priv}) = Sig. \quad (5.10)$$

Pengirim dokumen akan mengirimkan  $D$  berikut dengan tanda tangan  $Sig$  kepada penerima yang dituju. Pengirim dokumen juga membuat sebuah kunci publik  $k_{pub}$  yang tidak sama dengan  $k_{priv}$  untuk digunakan oleh penerima dalam memverifikasi pesan yang diterimanya.

Perhatikan bahwa penerima pesan memiliki dokumen  $D$  dan tanda tangan  $Sig$  yang diperoleh dari pengirim pesan (pembuat dokumen). Ia juga memiliki nilai  $k_{pub}$  yang dibuat oleh pembuat dokumen. Untuk memeriksa apakah dokumen  $D$  dan tanda tangan  $Sig$  yang diperolehnya autentik (memang benar-benar dikirimkan atau dibuat oleh pembuat dokumen) ia akan menggunakan fungsi verifikasi  $\text{ver}$  dengan masukan pesan  $D$ , tanda tangan  $Sig$ , dan kunci publik  $k_{pub}$  yang secara matematis memenuhi

$$\text{ver}(D, Sig, k_{pub}) = \text{true} \quad (5.11)$$

jika dan hanya jika tanda tangan digital dari  $D$  menggunakan kunci privat  $k_{priv}$  adalah  $Sig$ . Secara matematis ini berarti  $\text{ver}(D, Sig, k_{pub}) = \text{true} \Leftrightarrow \text{sign}(D, k_{priv}) = Sig$ . Pada praktiknya karena ukuran pesan atau dokumen yang dikirimkan bisa sangat besar, penandatanganan digital tidak dilakukan pada keseluruhan dokumen  $D$ , melainkan pada nilai hash dari  $D$ . Penjelasan lebih jauh mengenai skema tanda tangan digital secara umum dapat dilihat pada [2, Bab 4] dan [3, Bab 8].

Kita melihat bahwa proses penandatanganan dokumen pada suatu skema tanda tangan digital analog dengan proses enkripsi dari suatu pesan pada kriptografi asimetrik. Kemudian proses verifikasi kebenaran dari suatu dokumen berikut tanda tangannya yang dilakukan menggunakan kunci publik analog dengan proses dekripsi dari suatu pesan pada kriptografi asimetrik.<sup>4</sup> Untuk memperjelas ilustrasi misalkan pemilik atau pengirim dokumen adalah Samantha dan penerima dokumen adalah Victor. Skema tanda tangan digital menggunakan RSA dijelaskan pada Protokol 4.

#### Protokol 4 (Skema tanda tangan digital berbasis RSA yang dirangkum dari Tabel 4.1 pada [2])

Skema tanda tangan digital RSA (*RSA digital signature scheme*) dijelaskan sebagai berikut:

1. **Pembuatan kunci.** Pembuatan kunci dilakukan oleh Samantha sebagai pemilik dokumen (pengirim pesan). Samantha memilih dua bilangan prima  $p$  dan  $q$  yang berbeda dan cukup

<sup>4</sup>Salah satu perbedaan kecil adalah kunci publik yang digunakan untuk enkripsi dipakai sebagai kunci publik yang digunakan untuk memverifikasi keautentikan pesan. Kemudian kunci privat yang digunakan untuk dekripsi dipakai sebagai kunci privat untuk membuat tanda tangan digital.

---

besar. Samantha mendefinisikan  $N = pq$  dan  $\phi(N) = (p-1)(q-1)$ . Samantha memilih sebuah bilangan  $v \in \mathbb{Z}_{\phi(N)}$  dengan sifat  $\gcd(v, \phi(N)) = 1$ . Nilai  $(N, v)$  adalah kunci publik yang digunakan untuk memverifikasi pesan yang ditandatangani Samantha.

2. **Penandatanganan pesan/dokumen.** Proses penandatanganan dokumen oleh Samantha dilakukan dengan merepresentasikan dokumen (atau nilai hashnya) sebagai bilangan bulat pada rentang  $(1, N)$ . Samantha juga membangkitkan sebuah bilangan rahasia  $s \in \mathbb{Z}_{\phi(N)}$  yang memenuhi  $sv \equiv 1 \pmod{\phi(N)}$ . Karena  $\gcd(v, \phi(N)) = 1$  maka nilai  $s$  dijamin ada. Tanda tangan dari dokumen  $D$  menggunakan kunci rahasia  $s$ , dinotasikan dengan  $Sig$ , diperoleh menggunakan kalkulasi  $Sig = D^s \pmod{N}$ . Dokumen  $D$  dikirimkan bersamaan dengan nilai tanda tangannya ( $Sig$ ) kepada Victor.
3. **Verifikasi tanda tangan digital.** Victor menerima dokumen  $D$  berikut dengan tanda tangan  $Sig$ . Untuk memeriksa keautentikan dokumen  $D$  dan tanda tangan  $Sig$ , Victor menggunakan nilai kunci publik  $(N, v)$  dan memeriksa apakah  $Sig^v \equiv D \pmod{N}$  dipenuhi. Jika hal ini dipenuhi, maka Victor menyimpulkan bahwa dokumen  $D$  dan tanda tangan  $Sig$  adalah autentik dibuat oleh Samantha.

Pada Protokol 4 nilai  $s$  yang digunakan Samantha untuk menandatangani dokumen  $D$  dinamakan sebagai pangkat/eksponen penandatanganan (*signing exponent*). Nilai  $s$  bersifat rahasia dan hanya diketahui oleh Samantha saja. Tujuannya agar tanda tangan digital dari Samantha tidak dapat dipalsukan pihak lain. Kemudian nilai  $v$  yang digunakan Victor untuk memverifikasi keautentikan dokumen  $D$  dan tanda tangan  $Sig$  dinamakan sebagai pangkat/eksponen verifikasi (*verification exponent*). Nilai  $v$  bersifat publik dan diketahui oleh semua pihak yang ingin memverifikasi bahwa dokumen  $D$  beserta tanda tangan  $Sig$  adalah benar dibuat oleh Samantha.

Pada Protokol 4 kita melihat bahwa fungsi penandatanganan sign yang ada pada persamaan (5.10) memiliki ekspresi matematis berbentuk  $\text{sign}(D, k_{priv}) = \text{sign}(D, s) = D^s \pmod{N} = Sig$ , dengan  $Sig$  adalah tanda tangan yang dihasilkan. Kemudian fungsi verifikasi ver yang ada pada persamaan (5.11) dapat dinyatakan sebagai  $\text{ver}(D, Sig, k_{pub}) = \text{ver}(D, Sig, v) = (Sig^v \equiv D \pmod{N})$ . Perhatikan bahwa jika  $Sig$  adalah tanda tangan yang autentik untuk dokumen  $D$ , maka  $Sig \equiv D^s \pmod{N}$  dan akibatnya kita memiliki

$$Sig^v = (D^s)^v = D^{sv} \equiv D \pmod{N} \quad (5.12)$$

Bukti dari kongruensi (5.12) analog dengan bukti untuk kondisi (5.9) yang telah dijelaskan sebelumnya pada Subtopik 5.4.1.

Contoh numerik dari Protokol 4 yang diadaptasi dari [2, Contoh 4.5] dapat dijelaskan sebagai berikut. Misalkan Samantha sebagai pemilik dokumen memilih  $p = 1223$  dan  $q = 1987$ . Samantha menghitung  $N = pq = 2430101$ ,  $\phi(N) = (p-1)(q-1) = 2426892$ , dan memilih pangkat verifikasi  $v = 948047$ . Kita dapat memeriksa bahwa  $\gcd(v, \phi(N)) = 1$ . Nilai  $(N, v) = (2430101, 948047)$  diketahui secara publik.

Misalkan Samantha memiliki dokumen (atau nilai hashnya) yang dinyatakan dalam  $D = 1070777$ . Untuk menandatangani  $D$  Samantha menghitung nilai  $s$  yang memenuhi  $sv \equiv 1 \pmod{\phi(N)}$ . Hal ini dapat dilakukan dengan *extended Euclid's algorithm* atau perintah `xgcd` pada SageMath. Samantha memperoleh nilai  $s = 1051235$  yang sifatnya rahasia dan hanya diketahui olehnya. Tanda tangan dari  $D$  adalah  $Sig = D^s = 1070777^{1051235} \equiv 153337 \pmod{2430101}$ . Samantha menginformasikan nilai  $D$  dan  $Sig$  kepada penerima dokumen.

Misalkan Victor menerima pasangan  $(D, Sig)$ , dengan  $D = 1070777$  dan  $Sig = 153337$ . Untuk memeriksa keautentikan dokumen  $D$  dan tanda tangan  $Sig$  yang terkait dengannya, Victor memanfaatkan nilai  $v = 948047$  yang diketahui secara publik dan menghitung  $Sig^v = 153337^{948047} \equiv 1070777 \pmod{2430101}$ . Karena Victor memperoleh kondisi  $Sig^v \equiv D \pmod{N}$

---

maka ia menyimpulkan bahwa pasangan dokumen dan tanda tangan  $(D, \text{Sig})$  yang diperolehnya benar-benar dikirim/dibuat oleh Samantha.

Pada dasarnya simulasi skema tanda tangan digital menggunakan RSA pada SageMath hampir serupa dengan simulasi enkripsi dan dekripsi pesan pada RSA yang telah dijelaskan pada Subtopik 5.4.1. Misalkan Samantha memiliki sebuah dokumen  $D$  yang tanda tangannya akan diverifikasi dengan eksponen verifikasi  $v$  serta dua bilangan prima rahasia  $p$  dan  $q$ , maka proses pembuatan tanda tangan digital untuk  $D$ , yaitu  $\text{Sig}$ , dapat diperoleh dengan skrip berikut:

```
def signRSA(D, v, p, q):
    # proses membuat tanda tangan untuk D
    # nilai v adalah eksponen verifikasi yang diketahui secara publik
    # nilai p dan q hanya diketahui oleh Samantha
    N = p*q
    phiN = euler_phi(N)
    s = xgcd(v, phiN)[1]
    Sig = pow(D, s, N) # Sig adalah nilai tanda tangan untuk D
    return(D, Sig)
```

Fungsi `signRSA(D, v, p, q)` menggunakan nilai  $D$  dan  $v$  yang sifatnya publik (diketahui oleh semua orang) serta nilai  $p$  dan  $q$  yang hanya diketahui oleh Samantha. Hasil dari fungsi ini adalah pasangan  $(D, \text{Sig})$  dengan  $\text{Sig}$  adalah tanda tangan digital untuk dokumen  $D$ . Sebagai contoh, untuk menandatangani sebuah pesan  $D = 12332111999888777$  yang tanda tangannya akan diverifikasi dengan eksponen verifikasi  $v = 10403$  pada ring  $\mathbb{Z}_N$  dengan  $N = pq$  serta  $p = 999999667$  dan  $q = 999999337$  kita memakai perintah

```
signRSA(12332111999888777, 10403, 999999667, 999999337)
```

Hasil dari skrip ini adalah sebuah pasangan  $(D, \text{Sig})$  dengan  $D = 12332111999888777$  dan  $\text{Sig} = 434267244747427400$ . Simulasi proses verifikasi dari pasangan dokumen dan tanda tangan  $(D, \text{Sig})$  menggunakan pangkat verifikasi  $v$  dan nilai  $N = pq$  yang diketahui secara publik dapat dilakukan dengan skrip berikut:

```
def verRSA(D, Sig, v, N):
    # proses verifikasi tanda tangan Sig untuk D
    # semua nilai parameter diketahui secara publik
    check = pow(Sig, v, N)
    if (check == D):
        return 'tanda tangan valid'
    else:
        return 'tanda tangan tidak valid'
```

Sebagai contoh untuk memeriksa apakah pasangan  $(D, \text{Sig})$  merupakan dokumen dan tanda tangan yang autentik dihasilkan oleh Samantha dengan pangkat verifikasi  $v$  pada ring  $\mathbb{Z}_N$  untuk  $D = 12332111999888777$ ,  $\text{Sig} = 434267244747427400$ ,  $v = 10403$ , dan  $N = 999999004000220779$  kita dapat menggunakan perintah berikut:

```
verRSA(12332111999888777, 434267244747427400, 10403, 999999004000220779)
```

Hasil dari skrip ini adalah string '`tanda tangan valid`', yang berarti dokumen  $D = 12332111999888777$  memang benar-benar dibuat oleh Samantha. Misalkan kita memiliki sebuah dokumen lain, yaitu  $D' = 12332111999888666$  dan tanda tangan  $\text{Sig}' = 434267244747427401$ , kita dapat menjalankan perintah berikut untuk memeriksa keautentikan pasangan dokumen dan tanda tangan  $(D', \text{Sig}')$ :

```
verRSA(12332111999888666, 434267244747427401, 10403, 999999004000220779)
```

Hasil dari skrip tersebut adalah string '`tanda tangan tidak valid`' yang berarti pasangan dokumen dan tanda tangan  $(D', \text{Sig}')$  bukan berasal dari Samantha.

**Latihan 5.4.3** Pada dasarnya kita dapat menggabungkan protokol enkripsi dan dekripsi RSA pada Protokol 3 dengan skema tanda tangan digital berbasis RSA yang dijelaskan pada Protokol

---

4. Protokol yang dihasilkan memungkinkan Alice maupun Bob untuk saling berkirim pesan secara aman dan terautentikasi. Salah satu metodenya dijelaskan sebagai berikut.

1. **Pembuatan kunci.** Pembuatan kunci dilakukan oleh Alice maupun Bob dengan kriteria yang sama seperti pada Protokol 3 maupun Protokol 4, yaitu:

- (a) Alice memilih dua bilangan prima  $p_A$  dan  $q_A$  dengan  $p_A \neq q_A$  serta mendefinisikan  $N_A = p_A q_A$  dan  $\phi(N_A) = (p_A - 1)(q_A - 1)$ . Alice juga memilih sebuah bilangan  $e_A = v_A$  yang memenuhi  $\gcd(e_A, \phi(N_A)) = 1$ . Nilai  $(N_A, e_A) = (N_A, v_A)$  diumumkan kepada publik. Nilai  $e_A = v_A$  digunakan oleh siapa pun yang ingin mengirim pesan kepada Alice atau memeriksa keautentikan pesan yang dikirimkan oleh Alice.
- (b) Bob memilih dua bilangan prima  $p_B$  dan  $q_B$  dengan  $p_B \neq q_B$  serta mendefinisikan  $N_B = p_B q_B$  dan  $\phi(N_B) = (p_B - 1)(q_B - 1)$ . Bob juga memilih sebuah bilangan  $e_B = v_B$  yang memenuhi  $\gcd(e_B, \phi(N_B)) = 1$ . Nilai  $(N_B, e_B) = (N_B, v_B)$  diumumkan kepada publik. Nilai  $e_B = v_B$  digunakan oleh siapa pun yang ingin mengirim pesan kepada Bob atau memeriksa keautentikan pesan yang dikirimkan oleh Bob.

2. **Penandatanganan pesan.** Baik Alice maupun Bob harus menandatangani pesan yang belum dienkripsi dengan cara berikut:

- (a) Alice menentukan nilai  $s_A = d_A$  yang memenuhi  $e_A d_A \equiv 1 \pmod{\phi(N_A)}$  untuk nilai  $e_A$  yang dijelaskan pada langkah pembuatan kunci. Untuk menandatangani  $m_A$  Alice menghitung  $Sig_A = m_A^{s_A} \pmod{N_A}$ . Nilai  $(m_A, Sig_A)$  adalah pasangan pesan dan tanda tangannya yang absah. Nilai  $m_A$  dan  $Sig_A$  haruslah memenuhi  $1 < m_A, Sig_A < N_A$  agar pesan maupun tanda tangannya dapat diverifikasi menggunakan  $v_A$ . Selain itu nilai  $m_A$  dan  $Sig_A$  haruslah memenuhi  $1 < m_A, Sig_A < N_B$  agar pesan maupun tanda tangannya dapat dienkripsi menggunakan  $e_B$ .
- (b) Bob menentukan nilai  $s_B = d_B$  yang memenuhi  $e_B d_B \equiv 1 \pmod{\phi(N_B)}$  untuk nilai  $e_B$  yang dijelaskan pada langkah pembuatan kunci. Untuk menandatangani  $m_B$  Bob menghitung  $Sig_B = m_B^{s_B} \pmod{N_B}$ . Nilai  $(m_B, Sig_B)$  adalah pasangan pesan dan tanda tangannya yang absah. Nilai  $m_B$  dan  $Sig_B$  haruslah memenuhi  $1 < m_B, Sig_B < N_B$  agar pesan maupun tanda tangannya dapat diverifikasi menggunakan  $v_B$ . Selain itu nilai  $m_B$  dan  $Sig_B$  haruslah memenuhi  $1 < m_B, Sig_B < N_A$  agar pesan maupun tanda tangannya dapat dienkripsi menggunakan  $e_A$ .

3. **Enkripsi.** Proses enkripsi dilakukan baik oleh Alice maupun Bob ketika akan mengirimkan pesan dan tanda tangannya. Enkripsi dilakukan baik untuk pesan maupun tanda tangannya yang bersesuaian.

- (a) Alice mengenkripsi pesan  $m_A$  menggunakan  $e_B$  dan mendefinisikan teks sandi  $c_A = m_A^{e_B} \pmod{N_B}$ . Alice juga mengenkripsi tanda tangan  $Sig_A$  menggunakan  $e_B$  dan mendefinisikan  $t_A = Sig_A^{e_B} \pmod{N_B}$ . Alice mengirimkan pasangan  $(c_A, t_A)$  kepada Bob sebagai pasangan teks sandi dan tanda tangan yang terenkripsi.
- (b) Bob mengenkripsi pesan  $m_B$  menggunakan  $e_A$  dan mendefinisikan teks sandi  $c_B = m_B^{e_A} \pmod{N_A}$ . Bob juga mengenkripsi tanda tangan  $Sig_B$  menggunakan  $e_A$  dan mendefinisikan  $t_B = Sig_B^{e_A} \pmod{N_A}$ . Bob mengirimkan pasangan  $(c_B, t_B)$  kepada Alice sebagai pasangan teks sandi dan tanda tangan yang terenkripsi.

4. **Dekripsi.** Proses dekripsi dilakukan oleh penerima pesan. Pesan yang diterima merupakan sebuah pasangan teks sandi dan tanda tangan yang terenkripsi.

- 
- (a) Untuk mendapatkan pesan dari Bob, Alice menghitung  $m'_B = c_B^{d_A} \pmod{N_A}$  dan  $Sig'_B = t_B^{d_A} \pmod{N_A}$ . Selanjutnya pasangan pesan dan tanda tangan  $(m'_B, Sig'_B)$  akan diperiksa keautentikannya.
- (b) Untuk mendapatkan pesan dari Alice, Bob menghitung  $m'_A = c_A^{d_B} \pmod{N_B}$  dan  $Sig'_A = t_A^{d_B} \pmod{N_B}$ . Selanjutnya pasangan pesan dan tanda tangan  $(m'_A, Sig'_A)$  akan diperiksa keautentikannya.
5. **Verifikasi pesan dan tanda tangannya.** Baik Alice maupun Bob melakukan verifikasi untuk mengetahui keautentikan dari pesan dan tanda tangan yang diterima.
- (a) Alice yang menerima  $(m'_B, Sig'_B)$  dari Bob menggunakan nilai  $v_B = e_B$  yang diberikan Bob untuk memeriksa apakah  $Sig'^{v_B}_B \equiv m'_B \pmod{N_B}$ . Jika hal ini berlaku maka pesan  $m'_B$  benar-benar dikirimkan oleh Bob.
- (b) Bob yang menerima  $(m'_A, Sig'_A)$  dari Alice menggunakan nilai  $v_A = e_A$  yang diberikan Alice untuk memeriksa apakah  $Sig'^{v_A}_A \equiv m'_A \pmod{N_A}$ . Jika hal ini berlaku maka pesan  $m'_A$  benar-benar dikirimkan oleh Alice.

Berdasarkan penjelasan protokol yang telah diberikan, buatlah simulasi pada SageMath yang memberikan ilustrasi protokol RSA yang menggabungkan penyandian dan autentikasi pesan.

### 5.4.3 Beberapa Serangan Elementer pada RSA

Pada protokol penyandian pesan menggunakan RSA yang dijelaskan pada Protokol 3 kita melihat bahwa sebuah pesan  $m$  dapat diperoleh dari teks sandi  $c$  menggunakan nilai publik  $(N, e)$  sebagaimana dijelaskan pada Permasalahan 5.4.1. Ketika  $N$  dapat difaktorkan dengan mudah, maka kita dapat memperoleh nilai  $m$  menggunakan teknik pada Algoritma 4.

---

**Algoritma 4** Pencarian solusi dari  $x^e \equiv c \pmod{N}$  untuk  $N = pq$ .

---

**Masukan:** Kongruensi  $x^e \equiv c \pmod{N}$  dengan  $N$  yang dapat difaktorkan dengan mudah.

Nilai  $e$ ,  $c$ , dan  $N$  diketahui.

**Keluaran:** Nilai  $x$  yang memenuhi  $x^e \equiv c \pmod{N}$ .

1. Lakukan faktorisasi dari nilai  $N$  sehingga diperoleh dua bilangan prima  $p$  dan  $q$  yang memenuhi  $N = pq$ .
  2. Tentukan nilai  $\phi(N) = (p - 1)(q - 1)$ .
  3. Tentukan nilai  $d$  yang memenuhi  $ed \equiv 1 \pmod{\phi(N)}$  menggunakan *extended Euclid's algorithm*.
  4. Hitung nilai  $x = c^d \pmod{N}$ .
- 

**Latihan 5.4.4** Buatlah sebuah fungsi `modularroot(e, c, N)` pada SageMath untuk mencari nilai  $x$  yang memenuhi  $x^e \equiv c \pmod{N}$  untuk nilai  $N = pq$  yang mudah difaktorkan dengan menerapkan Algoritma 4. Lakukan pengujian pada program Anda dengan menuliskan perintah-perintah berikut:

```
print(modularroot(17389, 43927, 64349))
print(modularroot(10403, 2041895125359412, 999999004000220779))
print(modularroot(999999323, 16827134532710049, 999999004000220779))
```

Keluaran dari skrip ini adalah:

```
14458
59060322328768723
504736708736577
```

---

Perhatikan bahwa pencarian solusi  $x$  yang memenuhi  $x^e \equiv c \pmod{N}$  pada Algoritma 4 bergantung pada penyelesaian lain, yaitu faktorisasi dari  $N$ . Jika pemfaktoran dari  $N$  sulit dilakukan, maka kita tidak dapat mencari nilai  $x$  yang diinginkan menggunakan Algoritma 4. Pencarian  $x$  dengan cara *brute-force* jelas tidak efisien karena kita perlu mencoba  $N - 2$  nilai yang berbeda ketika  $c \notin \{0, 1\}$ . Hingga saat ini algoritma yang efisien untuk mencari solusi Permasalahan 5.4.1 belum ada.

Pada praktiknya nilai  $p$  maupun  $q$  yang dipakai pada RSA keduanya harus cukup besar. Nilai  $p$  dan  $q$  yang disarankan minimal terdiri dari 100 digit agar faktorisasi sulit dilakukan. Ketika salah satu dari nilai  $p$  atau  $q$  cukup kecil, kita dapat melalukan *trial division* untuk memperoleh salah satu faktor dari  $N$ . Hal ini diilustrasikan sebagai berikut. Misalkan  $N = pq$  dengan salah satu dari  $p$  atau  $q$  nilainya kurang dari  $10^6$ . Akibatnya kita dapat menghitung nilai dari  $N/p_i$  untuk semua bilangan prima  $p_i$  pada rentang  $[1, 10^6]$ . Jika nilai  $N/p_i$  adalah bilangan bulat, maka  $p_i$  diketahui dan akibatnya faktor dari  $N$  yang lain juga dapat dicari dengan mudah berdasarkan sifat  $N = pq$ .

Serangan juga dapat dilakukan apabila bilangan prima yang digunakan adalah  $p$  dan  $q$  dengan selisih yang kecil. Misalkan  $p$  dan  $q$  adalah bilangan prima dengan  $p < q$  dan nilai  $r = q - p$  yang cukup kecil. Perhatikan bahwa kondisi  $N = pq$  dengan  $p < q$  mengakibatkan  $p < \sqrt{N} < q$ . Akibatnya  $\sqrt{N} - p < q - p = r$  dan kita memperoleh  $\sqrt{N} - r < p < \sqrt{N}$ . Jika Kita mengetahui bahwa selisih dari dua bilangan prima yang digunakan tidak lebih dari  $r$ , maka kita dapat mencari semua bilangan prima pada rentang  $[\sqrt{N} - r, \sqrt{N}]$  untuk mendapatkan salah satu faktor dari  $N$ .

Kita juga dapat memanfaatkan informasi mengenai selisih dari  $p$  dan  $q$  untuk memperoleh faktorisasi dari  $N = pq$  dengan cara yang lain. Ketika  $p$  dan  $q$  keduanya adalah bilangan prima yang cukup besar namun memiliki selisih  $|p - q|$  yang cukup kecil, maka pertama kita dapat menuliskan  $N$  sebagai

$$\begin{aligned} N &= \left( \frac{p+q}{2} + \frac{p-q}{2} \right) \left( \frac{p+q}{2} - \frac{p-q}{2} \right) \\ &= \left( \frac{p+q}{2} \right)^2 - \left( \frac{p-q}{2} \right)^2. \end{aligned}$$

Ketika  $p$  dan  $q$  adalah bilangan prima yang cukup besar maka keduanya adalah bilangan ganjil sehingga  $\frac{p+q}{2}$  dan  $\frac{p-q}{2}$  adalah bilangan bulat. Misalkan selisih dari  $p$  dan  $q$  tidak lebih dari  $r$ , maka nilai dari  $\left(\frac{p-q}{2}\right)^2 \leq \frac{r^2}{4}$ . Kita dapat mencari nilai  $\left(\frac{p+q}{2}\right)^2$  dengan menghitung nilai  $M = N - \frac{s^2}{4}$  untuk semua nilai  $s$  pada rentang  $[1, r]$ . Jika  $M$  adalah bilangan bulat, maka kita memperoleh nilai  $\left(\frac{p+q}{2}\right)$ . Nilai  $p + q$  dapat diperoleh dengan cukup mudah. Selanjutnya misalkan  $S = p + q$ . Karena  $N = pq$ , maka kita memperoleh persamaan  $p^2 - Sp + N = 0$ . Nilai  $p$  yang positif merupakan salah satu faktor dari prima  $N$  dan faktor yang lain dapat diperoleh dengan membagi  $N$  dengan faktor prima dari  $N$  tersebut.

Serangan elementer terakhir yang dibahas pada subtopik ini adalah serangan *broadcast* Håstad. Ilustrasi yang diberikan di sini diadaptasi dari [26]. Misalkan Alice mengirim pesan kepada tiga penerima berbeda, yaitu  $P_1$ ,  $P_2$ , dan  $P_3$ , menggunakan kunci publik  $(N_1, e_1)$ ,  $(N_2, e_2)$ , dan  $(N_3, e_3)$ , dengan  $(N_i, e_i)$  adalah kunci publik dari  $P_i$ . Misalkan kita meninjau kondisi ketika  $e_1 = e_2 = e_3 = 3$ . Dalam kasus ini teks sandi yang dikirim Alice kepada  $P_1$ ,  $P_2$ , dan  $P_3$  berturut-turut adalah  $c_1 \equiv m^3 \pmod{N_1}$ ,  $c_2 \equiv m^3 \pmod{N_2}$ , dan  $c_3 \equiv m^3 \pmod{N_3}$ . Kita akan meninjau serangan ketika  $\gcd(N_i, N_j) = 1$  untuk  $i$  dan  $j$  yang berbeda. Hal ini dilakukan karena apabila  $\gcd(N_i, N_j) = p \neq 1$  maka  $p$  adalah faktor prima dari  $N_i$  dan  $N_j$  serta akibatnya kita dapat memfaktorkan  $N_i$  dan  $N_j$ .

Tinjau bahwa karena  $\gcd(N_i, N_j) = 1$  untuk  $i$  dan  $j$  yang berbeda maka kita dapat menggunakan Teorema Sisa Tiongkok (Teorema 2.3.4) untuk mencari  $c \in [0, N_1 N_2 N_3 - 1]$  yang

---

memenuhi

$$\begin{aligned} c &\equiv c_1 \pmod{N_1} \equiv m^3 \pmod{N_1} \\ c &\equiv c_2 \pmod{N_1} \equiv m^3 \pmod{N_2} \\ c &\equiv c_3 \pmod{N_1} \equiv m^3 \pmod{N_3}. \end{aligned} \tag{5.13}$$

Solusi dari sistem kongruensi (5.13) adalah  $c \equiv m^3 \pmod{N_1 N_2 N_3}$ . Perhatikan bahwa karena  $1 < m < N_i$  untuk setiap  $i = 1, 2, 3$ , akibatnya  $1 < m^3 < N_1 N_2 N_3$ . Nilai  $c$  yang memenuhi  $c \equiv m^3 \pmod{N_1 N_2 N_3}$  dapat dicari dengan mudah karena  $c = m^3$  berlaku pada ring bilangan bulat. Kita memiliki  $m = \sqrt[3]{c}$ .

## 5.5 Sistemkripto Elgamal

Sistemkripto RSA yang dijelaskan pada Subtopik 5.4 merupakan sistemkripto asimetrik pertama yang dipublikasikan secara luas. Meskipun demikian, sebagaimana telah kita lihat pada Subtopik 5.4, protokol penyandian maupun skema tanda tangan digital berbasis RSA tidak dikonstruksi berdasarkan masalah logaritma diskrit yang sebenarnya sudah diperkenalkan sejak tahun 1976. Sistemkripto Elgamal yang diperkenalkan oleh Taher Elgamal pada tahun 1985 merupakan sistemkripto asimetrik pertama yang keamanannya didasarkan pada masalah logaritma diskrit [28].

Pada subtopik ini kita akan membahas sistemkripto Elgamal sebagaimana dijelaskan pada [2, Tabel 2.3] maupun [3, Sistemkripto 7.1]. Pada sistemkripto Elgamal kita menggunakan grup siklis  $\mathbb{F}_p^*$  yang terkadang juga dinotasikan dengan  $\mathbb{Z}_p^*$ . Skenario yang kita tinjau serupa dengan yang sebelumnya dijelaskan untuk proses penyandian berbasis RSA pada Subtopik 5.4.1. Misalkan terdapat dua pihak, yaitu Alice dan Bob. Alice ingin mengirimkan pesan kepada Bob menggunakan kunci publik yang disediakan oleh Bob. Proses enkripsi pada dasarnya dapat dilakukan oleh semua pihak yang hendak mengirimkan pesan kepada Bob. Untuk membuka pesan tersandi Bob menggunakan kunci dekripsi yang hanya diketahui olehnya saja. Protokol penyandian dengan sistemkripto Elgamal dijelaskan pada Protokol 5.

**Protokol 5 (Protokol sistemkripto Elgamal yang dirangkum dari Tabel 2.3 pada [2])** Protokol sistemkripto Elgamal untuk penyandian pesan dijelaskan sebagai berikut:

1. **Pembuatan parameter publik.** Pihak yang terpercaya memilih sebuah bilangan prima  $p$  yang cukup besar dan mengumumkan field  $\mathbb{F}_p$  yang dipakai. Selain itu ia juga memilih sebuah elemen  $g \in \mathbb{F}_p^*$  yang merupakan pembangkit dari  $\mathbb{F}_p^*$ .
2. **Pembuatan kunci.** Bob memilih sebuah bilangan bulat rahasia  $b \in [2, p - 2]$  yang menjadi kunci privatnya. Kemudian Bob juga menghitung  $B = g^b \pmod{p} \in \mathbb{F}_p^*$  sebagai kunci publiknya. Nilai  $B$  dapat diketahui siapa pun yang ingin mengirim pesan secara aman kepada Bob.
3. **Enkripsi.** Misalkan Alice ingin mengirim pesan  $m \in \mathbb{F}_p^*$  menggunakan kunci  $B$  kepada Bob. Pertama ia memilih sebuah bilangan bulat  $k \in [2, p - 2]$  secara acak dan lalu menghitung nilai  $c_1 = g^k \pmod{p}$  dan  $c_2 = mB^k \pmod{p}$ . Teks sandi dari pesan  $m$  adalah pasangan  $(c_1, c_2)$  yang dijelaskan sebelumnya. Nilai  $(c_1, c_2)$  selanjutnya dikirimkan kepada Bob.
4. **Dekripsi.** Untuk memperoleh teks pesan dari teks sandi  $(c_1, c_2)$  Bob menghitung nilai  $m' = (c_1^b)^{-1} \cdot c_2 \pmod{p}$ . Secara matematis nilai  $m'$  sama dengan pesan  $m$  yang dikirimkan oleh Alice.

---

Pada sistemkripto Elgamal, enkripsi dari pesan  $m$  dengan kunci publik  $B$  pada dasarnya adalah fungsi  $\text{enc}(m, B) = (g^k, mB^k)$  untuk suatu bilangan bulat acak  $k \in [2, p - 2]$  yang semua operasi aritmetikanya ditinjau pada  $\mathbb{F}_p^*$ . Dari hasil ini kita melihat bahwa pesan pada sistemkripto Elgamal adalah sebuah elemen  $\mathbb{F}_p^*$  sedangkan teks sandi dari pesan yang bersesuaian adalah sebuah pasangan terurut  $(c_1, c_2) \in \mathbb{F}_p^* \times \mathbb{F}_p^*$ . Ini berarti *ukuran hasil penyandian pesan dua kali lebih besar daripada ukuran pesan awal*. Dalam hal ini kita mengatakan bahwa sistemkripto Elgamal memiliki sifat ekspansi pesan 2 banding 1 (*2 to 1 message expansion*).

Dekripsi dari suatu sandi  $(c_1, c_2)$  yang dilakukan menggunakan kunci rahasia  $b$  pada sistemkripto Elgamal merupakan sebuah fungsi  $\text{dec}(c_1, c_2, b) = (c_1^b)^{-1} \cdot c_2$  yang semua operasi aritmetikanya ditinjau pada  $\mathbb{F}_p^*$ . Untuk membuktikan keabsahan sistemkripto Elgamal sebagaimana dijelaskan pada Protokol 5, kita perlu membuktikan bahwa jika  $\text{enc}(m, B) = (c_1, c_2)$ , maka  $\text{dec}(c_1, c_2, b) = m$ . Dari Protokol 5 kita mengetahui bahwa  $\text{enc}(m, B) = (g^k, mB^k)$  untuk suatu bilangan bulat acak  $k \in [2, p - 2]$ . Akibatnya kita cukup membuktikan bahwa  $\text{dec}(g^k, mB^k, b) = m$ . Kita akan membuktikan hal ini dengan memanfaatkan definisi fungsi dekripsi. Tinjau bahwa

$$\begin{aligned}\text{dec}(g^k, mB^k, b) &= ((g^k)^b)^{-1} \cdot mB^k \quad (\text{definisi fungsi dekripsi}) \\ &= (g^{kb})^{-1} \cdot m(g^b)^k \quad (\text{karena } B = g^b) \\ &= (g^{kb})^{-1} \cdot m(g^{bk}) \\ &= g^{-kb} \cdot g^{kb} \cdot m \\ &= m \quad (\text{karena } g^{-a} \cdot g^a = 1 \text{ untuk } a \neq 0)\end{aligned}\tag{5.14}$$

Dari hasil pada (5.14) kita melihat bahwa nilai  $k$  yang digunakan Alice tidak berpengaruh pada hasil dekripsi pesan tersebut. Pada sistemkripto Elgamal teks pesan  $m$  “ditutupi” (*masked*) dengan nilai  $B^k$  dalam proses dekripsi. Alice juga mengirim nilai  $g^k$  yang nantinya digunakan Bob untuk membuka “tutup” (*mask*) ini. Proses dekripsi pada dasarnya merupakan kalkulasi  $\frac{mB^k}{(g^k)^b} = m$  yang dilakukan pada  $\mathbb{F}_p^*$ .

Berikut adalah ilustrasi numerik dari penyandian dengan sistemkripto Elgamal yang diaadaptasi dari [3, Contoh 7.1]. Misalkan pihak yang terpercaya mengumumkan penggunaan  $\mathbb{F}_{2579}^*$  dan elemen pembangkit  $g = 2$ . Bob memilih bilangan  $b = 765 \in [2, 2577]$  sebagai kunci rahasianya dan mengumumkan nilai  $B = g^b = 2^{765} = 949$  sebagai kunci publiknya.

Misalkan Alice ingin mengirim sebuah pesan  $m = 1299$  kepada Bob. Pertama Alice memilih bilangan bulat  $k \in [2, 2577]$  secara acak. Misalkan Alice memilih  $k = 853$ . Alice menghitung  $c_1 = g^k = 2^{853} = 435$  dan  $c_2 = mB^k = 1299 \cdot 949^{853} = 2396$ . Alice mengirimkan nilai  $(c_1, c_2) = (435, 2396)$  sebagai teks sandinya kepada Bob. Untuk memperoleh pesan Alice, Bob melakukan dekripsi dengan menghitung  $(c_1^b)^{-1}c_2 = (435^{765})^{-1}2396 = 1299$ . Semua aritmetika dilakukan dalam  $\mathbb{F}_{2579}^*$ .

Pada sistemkripto Elgamal pembuatan parameter publik sebenarnya dapat dilakukan oleh Bob. Pada kondisi ini *field*  $\mathbb{F}_p^*$  yang digunakan serta elemen pembangkit  $g \in \mathbb{F}_p^*$  dikonstruksi oleh Bob dan dipublikasikan bersama dengan nilai  $B = g^b$ . Nilai  $b \in [2, p - 2]$  dibangkitkan secara acak oleh Bob dan bersifat rahasia. Simulasi pembuatan parameter publik dan kunci publik untuk sistemkripto Elgamal pada SageMath dapat dilakukan dengan skrip berikut:

---

```

import random
# fungsi untuk mengonstruksi parameter dan kunci publik pada sistem kripto Elgamal
# pertama kita akan membangkitkan sebuah bilangan prima pada rentang [Min,Max]
print('pertama kita bangkitkan sebuah bilangan prima pada selang [Min,Max]')
Min, Max = map(Integer, input('masukkan Min dan Max, dipisahkan dengan spasi: ').split())
print('Kita memakai fungsi prime_list(Min,Max) yang telah dibuat sebelumnya.')
p = random.choice(prime_list(Min,Max))
print('Field hingga yang dipakai.')
F = GF(p); print(F)
g = F.primitive_element()
print('Elemen pembangkit yang dipakai: ',g)
# konstruksi kunci rahasia untuk Bob
b = random.randint(2,p-1)
print('Kunci rahasia Bob, hanya diketahui Bob: ',b)
# konstruksi kunci publik dari Bob
B = g^b
print('Kunci publik dari Bob, B: ',B)

```

Pada skrip ini kita memakai fungsi `prime_list(Min,Max)` untuk membangkitkan sebuah *list* yang isinya adalah seluruh bilangan prima pada rentang  $[Min, Max]$ . Fungsi ini dijelaskan pada Subtopik 2.1.3. Sebenarnya kita dapat menggunakan metode lain yang dijelaskan pada Subtopik 2.1.3, misalnya fungsi `random_prime`. Kemudian pada skrip ini kita memakai *library* `random` pada Python agar fungsi `random.choice(L)` dapat dipakai untuk mengambil sebuah elemen dari *list*  $L$  secara acak. *Library* ini juga dipakai untuk menjalankan fungsi `random.randint(m,n)` yang berguna untuk membangkitkan sebuah bilangan bulat secara acak pada rentang  $[m,n]$ . Kita memakai fungsi ini untuk membangkitkan bilangan prima  $p$  secara acak pada rentang  $[Min, Max]$ . Kunci rahasia Bob merupakan sebuah bilangan bulat  $b$  yang dibangkitkan secara acak pada rentang  $[2, p - 2]$ . Misalkan dari interaksi yang terjadi berdasarkan skrip sebelumnya kita memilih  $Min = 999200000$  dan  $Max = 999300000$  (salah satu tujuannya adalah mengonstruksi bilangan prima yang terdiri atas 9 digit), maka salah satu keluaran yang mungkin dihasilkan dari skrip tersebut adalah:

```

pertama kita bangkitkan sebuah bilangan prima pada selang [Min,Max]
masukkan Min dan Max, dipisahkan dengan spasi: 999200000 999300000
Kita memakai fungsi prime_list(Min,Max) yang telah dibuat sebelumnya.
Field hingga yang dipakai.
Finite Field of size 999209557
Elemen pembangkit yang dipakai: 13
Kunci rahasia Bob, hanya diketahui Bob: 99568227
Kunci publik dari Bob, B: 246109803

```

Di sini kita memakai parameter publik  $\mathbb{F}_p^*$  untuk  $p = 999209557$  dan  $g = 13 \in \mathbb{F}_p^*$  sebagai pembangkit. Kunci rahasia yang digunakan Bob adalah  $b = 99568227$  sedangkan kunci publik yang dihasilkan adalah  $B = 246109803$ . Proses enkripsi untuk suatu pesan  $m \in \mathbb{F}_p^*$  dilakukan menggunakan nilai  $k \in [2, p - 2]$  yang diambil secara acak dan kunci publik  $B$ . Hal ini dapat disimulasikan menggunakan skrip berikut pada SageMath:

```

import random
# enkripsi pada protokol Elgamal
# Di sini kita mengenkripsi pesan m dan kunci publik B
def encElgamal(m,B):
    k = random.randint(2,p-2)
    c1 = F(g)^k
    c2 = F(m)*F(B)^k
    return (c1,c2)

```

Fungsi `encElgamal(m,B)` mengenkripsi pesan  $m \in \mathbb{F}_p^*$  menggunakan kunci publik  $B$  yang diberikan Bob serta nilai pembangkit  $g$  yang merupakan parameter publik. Nilai  $k$  adalah bilangan bulat yang dibangkitkan secara acak pada rentang  $[2, p - 2]$  menggunakan fungsi `random.randint`. Penggunaan nilai  $k$  yang acak ini mengakibatkan hasil enkripsi dari suatu pesan  $m$  menggunakan kunci publik  $B$  yang sama dapat berbeda-beda.

**Latihan 5.5.1** Gunakan fungsi `encElgamal(m,B)` yang telah dijelaskan sebelumnya untuk

---

menjalankan beberapa eksekusi fungsi tersebut untuk nilai  $m = 123456789$  dan  $B = 246109803$ . Parameter publik yang digunakan adalah  $\mathbb{F}_p^*$  dengan  $p = 999209557$  dan  $g = 13 \in \mathbb{F}_p^*$ . Lakukan setidaknya lima eksekusi yang berbeda.

Misalkan kita ingin mengenkripsi  $m = 123456789 \in \mathbb{F}_{999209557}^*$  menggunakan kunci publik  $B = 246109803$ , maka kita dapat menjalankan perintah `encElgamal(123456789, 246109803)`. Salah satu hasil eksekusi yang mungkin memberikan nilai  $(c_1, c_2) = (192981701, 24127824)$ .

Untuk melakukan dekripsi pada sistemkripto Elgamal, kita memerlukan informasi tentang nilai  $b$  yang sebelumnya dibangkitkan secara acak. Dekripsi juga dilakukan menggunakan parameter publik  $\mathbb{F}_p^*$  yang telah diberikan sebelumnya. Salah satu skrip SageMath yang dapat digunakan untuk simulasi dekripsi sistemkripto Elgamal adalah:

```
# dekripsi pada protokol Elgamal
# Di sini kita mendekripsi (c1,c2) menggunakan kunci rahasia b.
def decElgamal(c1,c2,b):
    message = (F(c1)^b)^-1 * F(c2)
    return message
```

Dekripsi dari teks sandi  $(192981701, 24127824)$  menggunakan kunci rahasia  $b = 99568227$  dilakukan dengan eksekusi perintah `decElgamal(192981701, 24127824, 99568227)`. Hasil dari dekripsi ini adalah pesan  $123456789$ . Pada sistemkripto Elgamal kita dapat melihat bahwa sebuah pesan dapat dienkripsi menjadi beberapa teks sandi yang berbeda-beda (sesuai dengan keacakan nilai  $k$ ). Dengan alasan ini kita mengatakan bahwa *enkripsi pada sistemkripto Elgamal bersifat probabilistik*. Namun, meskipun hasil enkripsi tersebut berbeda-beda, hasil dekripsi dari teks sandi yang berbeda-beda tersebut selalu sama.

Pada dasarnya struktur  $\mathbb{F}_p^*$  yang digunakan pada sistemkripto Elgamal dapat diganti dengan grup siklis apapun. Jika kita mengganti  $\mathbb{F}_p^*$  dengan suatu kurva eliptik, maka kita memperoleh sistemkripto yang dinamakan sistemkripto Elgamal eliptik (*elliptic Elgamal cryptosystem*). Kemudian, sebagaimana perluasan skema tanda tangan digital berbasis RSA dari sistemkripto RSA yang dijelaskan pada Subtopik 5.4.2, kita juga dapat mengonstruksi skema tanda tangan digital dari sistemkripto Elgamal. Lebih jauh, konstruksi skema tanda tangan digital ini juga dapat dilakukan dengan meninjau kurva eliptik sebagai grup siklis yang dipakai.

### Latihan 5.5.2

- Buatlah sebuah fungsi `encElgamalstring(M, B, g, p)` yang melakukan enkripsi pada string masukan  $M$  yang merepresentasikan teks pesan (*plaintext*) dalam format string dengan kunci publik  $B$  dan parameter publik  $g \in \mathbb{F}_p^*$ . Terapkan Algoritma 2 yang dijelaskan pada Subtopik 5.3. Hasil enkripsi adalah pasangan terurut pada  $\mathbb{F}_p^* \times \mathbb{F}_p^*$  yang dapat berbeda-beda untuk masukan yang sama. Nilai  $k$  dibangkitkan di dalam fungsi enkripsi. Sebagai ilustrasi perintah-perintah berikut:

```
print(encElgamalstring('SageMath', 934850638599338434, 2, 999999999999999989))
print(encElgamalstring('SageMath', 934850638599338434, 2, 999999999999999989))
print(encElgamalstring('SageMath', 934850638599338434, 2, 999999999999999989))
```

dapat memberikan keluaran:

```
(61833697307401106, 344545329178341997)
(838282326411722306, 866936018080129275)
(638613178470605421, 241592589281083638)
```

Jawaban yang diperoleh pada proses enkripsi ini selanjutnya akan didekripsi menggunakan skrip selanjutnya.

- Buatlah sebuah fungsi `decElgamalstring(c1, c2, b, p)` yang melakukan dekripsi pada pasangan terurut  $(c_1, c_2)$  yang merepresentasikan teks sandi pada  $\mathbb{F}_p^* \times \mathbb{F}_p^*$  menggunakan kunci rahasia  $b \in [2, p - 2]$ . Hasil dekripsi adalah string yang merepresentasikan

---

pesan yang sebelumnya dienkripsi. Terapkan Algoritma 3 yang telah dikaji pada Subtopik 5.3. Gunakan hasil enkripsi dari `encElgamalstring (M, B, g, p)` sebagai masukan. Sebagai ilustrasi, masukan berikut:

```
print(decElgamalstring(618336973074011106, 344545329178341997,1000000007,99999999999999989))  
print(decElgamalstring(838282326411722306, 866936018080129275,1000000007,99999999999999989))  
print(decElgamalstring(638613178470605421, 241592589281083638,1000000007,99999999999999989))
```

semuanya memberikan keluaran berupa string SageMath.

# Lampiran A

## Repositori Skrip SageMath

Skrip SageMath yang digunakan dalam tutorial ini tersedia dalam format .ipynb dan dapat diakses pada tautan <https://github.com/csmarz/SageMathTutorial>. Seluruh skrip yang terkait dengan Topik  $i$  ( $1 \leq i \leq 5$ ) tersedia pada berkas dengan format penamaan Tutorial-Topik*<i>*.ipynb. Untuk memakai berkas .ipynb tersebut dengan optimal, pembaca mungkin perlu me-restart *Kernel* SageMath setelah menyelesaikan satu atau lebih rangkaian skrip pada satu subtopik. Hal ini dilakukan untuk membersihkan *cache* memori dari nama-nama variabel, definisi fungsi, maupun definisi-definisi lain yang mungkin dideskripsikan secara redundan. Hal ini dapat dilakukan dengan memilih *Kernel* pada menu bar dan memilih *Restart*. Pembaca juga dapat me-restart *Kernel* dan menghilangkan semua keluaran dengan memilih *Restart & Clear Output*.

# Penutup

Penulis menyadari bahwa penjelasan yang ada pada tutorial ini belum lengkap dan memiliki banyak kekurangan. Pada edisi berikutnya penulis berencana untuk menambah kajian-kajian berikut yang cukup penting namun belum dicantumkan pada edisi ini:

1. Pada Topik 1 rencananya penulis akan menambahkan kajian mengenai struktur data *set* (representasi himpunan pada Python) dan *dictionary*. Selain itu mengingat SageMath juga memakai pendekatan berorientasi objek (seperti kelas dan *method*), maka penulis juga berencana menambahkan sedikit bahasan mengenai pemrograman berorientasi objek pada Python yang relevan dengan SageMath.
2. Pada Topik 2 penulis berencana untuk menambahkan kajian mengenai *lattice*, terutama *Euclidean lattice*. *Euclidean lattice* merupakan subgrup dari  $\mathbb{R}^n$  yang isomorfik dengan grup penjumlahan  $\mathbb{Z}^n$ . Sebuah *lattice* di  $\mathbb{R}^n$  untuk himpunan basis  $B = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$  dari  $\mathbb{R}^n$  dapat dituliskan sebagai himpunan seluruh kombinasi linier dari vektor-vektor pada  $B$  dengan koefisien-koefisien bilangan bulat, atau secara matematis  $\{\sum_{i=1}^n a_i \mathbf{v}_i\}$  dengan  $a_i \in \mathbb{Z}$  untuk setiap  $1 \leq i \leq n$ . Lattice merupakan salah satu struktur matematika yang digunakan pada kriptografi pasca-kuantum (yang tahan terhadap serangan komputer kuantum).
3. Pada Topik 3 penulis berencana untuk menambahkan kajian mengenai kriptanalisis pada sandi Hill, sandi Vigenére, maupun sandi transposisi.
4. Pada Topik 4 penulis berencana untuk memberi deskripsi dari SDES dan mini AES yang lebih rinci sesuai dengan [17] untuk SDES dan [18] untuk mini AES. Penulis juga berencana untuk menambahkan deskripsi mengenai DES dan AES secara singkat.
5. Pada Topik 5 penulis berencana untuk menambahkan penjelasan mengenai sistemkripto Elgamal dengan kurva eliptik (EC Elgamal), skema tanda tangan digital berbasis sistemkripto Elgamal, penjelasan mengenai *Digital Signature Algorithm* (DSA), dan varian dari DSA menggunakan kurva eliptik (yang juga disebut sebagai ECDSA).

# Daftar Pustaka

- [1] “SAGEmath Reference Documentation,” <https://doc.sagemath.org/pdf/en/reference/>, diakses: 2021-07-19.
- [2] J. Hoffstein, J. C. Pipher, and J. H. Silverman, *An introduction to mathematical cryptography*, 2nd ed. Springer, 2014.
- [3] D. R. Stinson and M. B. Paterson, *Cryptography: theory and practice, 4th Edition*. Chapman and Hall/CRC, 2018.
- [4] W. Trappe and L. C. Washington, *Introduction to cryptography with coding theory, 3rd Edition*. Pearson Education, 2020.
- [5] “SageMath-Wikipedia,” <https://en.wikipedia.org/wiki/SageMath>, diakses: 2021-07-19.
- [6] “Sage Installation Guide Release 9.4,” <https://doc.sagemath.org/pdf/en/installation/installation.pdf>, diakses: 2021-07-19.
- [7] “The Python Tutorial,” <https://docs.python.org/3/tutorial/>, diakses: 2021-08-1.
- [8] M. E. O’Sullivan, “SDSU Sage tutorial,” <https://mosullivan.sdsu.edu/Teaching/sdsu-sage-tutorial/>, diakses: 2021-07-27.
- [9] R. A. Beezer, *Sage for Abstract Algebra*. Department of Mathematics and Computer Science, University of Puget Sound, 2011.
- [10] T. W. Judson, *Abstract algebra: theory and applications*, 2020.
- [11] W. J. Gilbert and W. K. Nicholson, *Modern algebra with applications*. John Wiley & Sons, 2004, vol. 66.
- [12] J. R. Durbin, *Modern algebra: An introduction*. John Wiley & Sons, 2008.
- [13] A. Arifin, *Aljabar*. ITB Press, 2000.
- [14] K. H. Rosen, *Discrete Mathematics and Its Applications, 8th Edition*. McGraw Hill Education, 2019.
- [15] H. Anton and C. Rorres, *Elementary linear algebra: applications version*. John Wiley & Sons, 2013.
- [16] E. A. Lamagna, “Simplified DES,” <https://www.cs.uri.edu/cryptography/dessimplified.htm>, diakses: 2021-08-3.
- [17] E. F. Schaefer, “A simplified data encryption standard algorithm,” *Cryptologia*, vol. 20, no. 1, pp. 77–84, 1996.

- 
- [18] R. C.-W. Phan, “Mini advanced encryption standard (mini-AES): a testbed for cryptanalysis students,” *Cryptologia*, vol. 26, no. 4, pp. 283–306, 2002.
  - [19] W. Diffie and M. E. Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory*, vol. IT.22 No. 6, pp. 644–654, 1976.
  - [20] I. Ingemarsson, D. T. Tang, and C. K. Wong, “A Conference Key Distribution System,” *IEEE Transactions on Information Theory*, vol. 28, no. 5, pp. 714–720, 1982.
  - [21] M. Burmester and Y. Desmedt, “A Secure and Efficient Conference Key Distribution System,” in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1994, pp. 275–286.
  - [22] M. Steiner, G. Tsudik, and M. Waidner, “Diffie-Hellman Key Distribution Extended to Group Communication,” in *Proceedings of the 3rd ACM Conference on Computer and Communications Security*. ACM, 1996, pp. 31–37.
  - [23] R. Dewoprabowo, M. Arzaki, and Y. Rusmawati, “On Generalized Divide and Conquer Approach for Group Key Distribution: Correctness and Complexity,” in *2018 6th International Conference on Information and Communication Technology (ICoICT)*. IEEE, 2018, pp. 416–424.
  - [24] R. Shirey, “Internet Security Glossary, Version 2,” <https://datatracker.ietf.org/doc/html/rfc4949>, 2007, diakses: 2021-8-24.
  - [25] D. R. Kohel, “Cryptography,” *Institut de Mathématiques de Luminy, Universite de la Méditerranee, Marseille*, 2008.
  - [26] “Coursera: Number Theory and Cryptography,” <https://www.coursera.org/learn/number-theory-cryptography/>, 2021, diakses: 2021-8-5.
  - [27] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
  - [28] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.