K R I P T O G R A F I

P Y T H O N

*Sebuah nilai dan inspirasi untuk Indonesia, Tanah Airku
Dilengkapi dengan topik dan projek pembahasan

Matius Celcius Sinaga

Perhatian dan Peringatan

1. Jika Anda membaca hal ini berarti anda sudah mengunduh ebook **Kriptografi Python** yang ditulis oleh **Matius Celcius Sinaga**, tetaplah menyertakan narasumber, sumber dan nama penulis jika anda ingin melakukan penulisan ulang/repost/reshare sebagai bentuk menghargai karya orang lain.

2. Anda berhak untuk berbagi dalam bentuk salinan, baik wujud fisik maupun media elektronik lainnya dimana ebook adalah sebagai wujud asli/bentuk pertama dalam publikasi dengan tetap mencantumkan nama penulis baik berupa potongan kalimat/referensi atau source code/sumber kode namun **tidak untuk komersial atau bahkan sebagai bentuk meraih keuntungan pribadi/golongan**.

3. Segala kerusakan dan hal yang mempengaruhi, mengubah dan menambah hal lain yang terjadi pada komputer pribadi anda maupun konsol dekstop dan telepon genggam akibat dari anda menjalankan atau mengikuti yang dimaksud di dalamnya adalah source code yang anda kompilasi menghasilkan error atau kerusakan diluar tanggung jawab penulis, tetaplah berhati-hati dan bertanggung jawab.

4. Dalam topik-topik dalam buku ini masih memiliki banyak kekurangan dalam berbagai hal, tetaplah senantiasa memberitahu saya untuk perkembangan selanjutnya melalui surat elektronik dan media sosial yang memungkinkan komunikasi secara langsung dan tetap bertanggung jawab.Saya sangat membutuhkan kritik dan saran dari Anda

Hubungi saya : matiussinaga29@gmail.com
matiuscelciussinaga@merahputih.id
Matius Celcius Sinaga (Facebook, Linkedin)
Matius Sinaga (Academia)

Coba fikirkan hal apa yang tidak dapat diretas hari ini ? Apakah demikian juga halnya dahulu ? apakah esok dan masa depan kita masih akan terus mengalami semua kekacauan ini ? Coba jawab dalam diri anda berapa banyak informasi yang anda ijinkan untuk diketahui dan tidak oleh oranglain, ada beberapa cara agar tetap bisa menjaga rahasia yang kita miliki dalam era teknologi serba canggih ini. Tidak ada yang benar-benar aman di dunia maya, sudah marak terjadi aksi pembobolan bank melalui saluran internet, peretasan akun internet juga hal-hal yang bersifat pribadi dengan mudah dicuri dan disebarkan oleh pihak-pihak yang tidak bertanggung jawab secara sengaja dan tidak. Lalu apakah kita akan membiarkan hal ini terus terjadi ? Tetaplah berhati-hati, pada buku ini akan dibahas hal mengenai sandi namun penuh dengan kata-kata/kalimat yang mudah dimengerti, bagaimana anda dapat membuat sandi anda sendiri, merusak dan bahkan meretasnya. Dan untuk yang lebih menarik akan dibahas mengenai apa dan bagaimana sandi RSA melakukan enkripsinya.

TERIMAKASIH Ayah, terimakasih Ibu dan terimakasih untuk yang awal dan akhir, seluruh jagat raya akan selalu memuji dan menyembahMu, Ya Allah Bapa Ya Tuhanku. Puji Tuhan. Halleluya.

Amin.

TENTANG BUKU INI

Ada banyak cara untuk seseorang belajar sesuatu, dimulai dengan membaca buku, melakukan praktik, belajar dari pengalaman orang lain, diajari oleh orang lain atau bahkan dengan pengalaman anda sendiri. Saya berharap buku ini dapat membantu anda.

Ini adalah buku kedua dari buku yang saya tuliskan tahun ini (September, 2017) dan menjadi buku kedua yang saya tulis. Ini adalah mimpi saya untuk tetap dapat berbagi kepada orang lain menjadi orang yang berguna setidaknya bagi bangsa dan negara. Setidaknya saya menghabiskan waktu kurang lebih 3 bulan untuk benar-benar yakin bahwa apa yang saya tulis adalah hal yang berguna nantinya bagi orang lain. Buku kedua ini adalah buku perjanjian bagi saya, perjanjian dimana yang dimaksudkan adalah janji pada diri untuk mengajarkan sesuatu pada orang lain. Saya belajar mengenai pemograman dan hal lainnya dimulai pada usia 15 tahun dengan membaca artikel di Internet atau dahulu disebut dengan WARNET (Warung Internet) hingga akhirnya dapat belajar otodidak mempelajari bahasa pemograman dan pengkodean. Ada banyak kisah tentang pemograman yang saya miliki dan bahkan itu terlalu panjang untuk diungkapkan dalam tulisan ini namun kelak saya berharap dapat berbagi. Buku Perjanjian ini juga adalah janji saya untuk 21 tahun lebih menjalani kehidupan yang indah dari Tuhan Allah Yang Maha Esa dan esok adalah hari baru bagi saya menjalani usia 22 Tahun. Saya sangat berharap dengan buku ini seseorang terbantu dan menjadi kado ulang tahun yang lebih dari cukup dikenang nantinya.

Tetaplah menulis dan berbagi dan jangan menyerah karena keterbatasan, saya pun menulis buku ini dengan meminjam laptop orang lain dan tetap tertawa, berterima kasih pada pencipta.

Medan, 26 September 2017

"… Seseorang boleh pandai dan terkenal, namun jika ia tidak menulis Ia akan tenggelam dalam masyarakat dan sejarah".

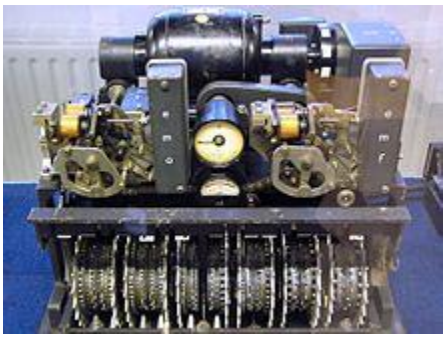
Kriptografi dan Python

DAFTAR ISI

Bagian 1

KRIPTOGRAFI

Terminologi, Sejarah kriptografi dan kriptanalisis, Kriptografi Moderen



Alat kriptografi Lorenz yang dipakai di Jerman saat perang dunia II

Kriptografi (atau **kriptologi**; dari bahasa Yunani κρυπτός *kryptós*, "tersembunyi, rahasia"; dan γράφειν *graphein*, "menulis", atau -λογία **logi**, "ilmu") merupakan keahlian dan ilmu dari cara-cara untuk komunikasi aman pada kehadirannya di pihak ketiga. Secara umum, kriptografi ialah mengenai mengkonstruksi dan menganalisis protokol komunikasi yang dapat memblokir lawan; berbagai aspek dalam keamanan informasi seperti data rahasia, integritas data, autentikasi, dan non-repudansi merupakan pusat dari kriptografi modern. Kriptografi modern terjadi karena terdapat titik temu antara disiplin ilmu matematika, ilmu komputer, dan teknik elektro. Aplikasi dari kriptografi termasuk ATM, password komputer, dan E-commerce.

Kriptografi sebelum pada termodernisasi merupakan sinonim dari "enkripsi", konversi dari kalimat-kalimat yang dapat dibaca menjadi kelihatan tidak masuk akal. Pembuat dari pesan enkripsi membagi teknik pemecahan sandi yang

dibutuhkan untuk mengembalikan informasi asli jika hanya dengan penerima yang diinginkan, sehingga dapat mencegah orang yang tidak diinginkan melakukan hal yang sama. Sejak Perang Dunia I dan kedatangan komputer, metode yang digunakan untuk mengelola kriptologi telah meningkat secara komplek dan pengaplikasiannya telah tersebar luar.

Kriptografi modern sangat didasari pada teori matematis dan aplikasi komputer; algoritma kriptografi didesain pada asumsi ketahanan komputasional, membuat algoritma ini sangat sulit dipecahkan oleh musuh. Secara teoretis, sangat sulit memecahkan sistem kriptografi, namun tidak layak melakukannya dengan cara-cara praktis. Oleh karena itu skema ini disebut sangat aman secara komputasional; kemajuan teoretis dapat meningkatkan algoritma faktorisasi integer, dan meningkatkan teknologi komputasi yang membutuhkan solusi untuk diadaptasi terus-menerus. Terdapat skema keamanan informasi yang benar-benar tidak boleh dapat ditembus bahkan dengan komputasi yang tak terbatas namun skema ini sangat sulit diimplementasikan.

Teknologi yang berhubungan dengan kriptologi memiliki banyak masalah legal. Di Inggris, penambahan Regulasi Penyelidikan Aksi Wewenang membutuhkan kriminal yang tertuduh harus menyerahkan kunci dekripsinya jika diminta oleh penegah hukum. Jika tidak pengguna akan menghadapi hukum pidana. **Electronic Frontier Foundation** (**EFF**) terlibat dalam sebuah kasus di Amerika Serikat yang mempertanyakan jika seorang tersangka harus menyerahkan kunci dekripsi mereka kepada pengak hukum merupakan inkonstitusionil. EFF memperdebatkan bahwa regulasi ini merupakan pelanggaran hak untuk tidak dipaksa mencurigai dirinya sendiri, seperti dalam Amendemen Kelima Konsitusi Amerika.

TERMINOLOGI

HINGGA zaman modern kriptografi mengacu hampir secara ekslusif pada enkripsi, yang merupakan proses mengkonversikan informasi biasa menjadi teks yang tak dapat dipahami (disebut teks sandi). Deskripsi merupakan kebalikan, dengan kata lain, memindahkan teks sandi yang tidak dapat dibaca menjadi teks yang dapat dibaca. *sandi* atau (***cypher***) merupakan sepasang algoritma yang menciptakan enkripsi dan membalikan dekripsi. Operasi yang lebih mendalam dari sandi diatur baik oleh algoritma dan pada setiap permintaan dekripsi dengan kunci. Kunci ini bersifat rahasia (yang biasanya diketahui hanya oleh orang yang berkomunikasi), dan biasanya terdiri dengan karakter string singkat, yang dibutuhkan untuk mendekripsi teks sandi. Sebelumnya dinamakan "kriptosistem" yang merupakan daftar teratur dari elemen-elemen teks terbatas, teks sandi terbatas, kunci terbatas, dan algoritma dekripsi dan enkripsi yang berkoresponden pada setiap kunci. Kunci sangat penting baik pada penggunaan secara teoretis maupun sebenarnya, di mana sandi tanpa kunci variabel dapat dengan mudah rusak dengan hanya pengetahuan yang digunakan dari sandi dan dengan kemungkinan tidak berguna (atau malam tidak produktif) untuk banyak tujuan. Secara historis, sandi sering digunakan secara langsung untuk enkripsi atau deskripsi tanpa prosedur tambahan seperti *autentikasi* atau *pengecekan integritas.*

Dalam penggunaan bahasa sehari-hari, istilah "sandi" sering digunakan untuk menunjukkan setiap metode enkripsi atau penyembunyian arti. Bagaimanapun, dalam kriptografi, sandi telah memiliki arti yang lebih spesifik. Itu berarti pemindahan unit teks (contoh kata atau frasa yang berarti) dengan sebuah kata sandi (sebagai contoh, *"wallaby"* berarti *"menyerang saat fajar")*. Sandi tidak lagi digunakan pada kriptografi serius-kecuali sesekali untuk beberapa hal yang

menyangkut istilah tertentu-sejak sandi yang dipilih secara tepat lebih praktis dan lebih aman daripada sandi terbaik dan juga dapat diadaptasikan pada komputer. Kriptanalisis merupakan istilah yang digunakan untuk mempelajari metode untuk memperoleh arti dari informasi enkripsi tanpa mengakses sandi secara normal yang dibutuhkan untuk melakukannya; sebagai contoh ilmu yang mempelajari cara untuk memecahkan algoritma enkripsi atau implementasinya. Beberapa kegunaan dari istilah kriptografi dan kriptologi selalu berubah di Bahasa Inggris, sedang lainnya menggunakan kriptografi untuk merujuk secara spesifik pada penggunakan dan pengaplikasikan dari teknik kriptografi dan kriptologi untuk merujuk pada ilmu kombinasi dari kriptografi dan kriptanalisis. Bahasa Inggris lebih fleksibel dari istilah umum yang digunakan pada beberapa bahasa lain yang dimana kriptologi (dilakukan oleh kriptolog) selalu digunakan pada arti kedua di atas.
Ilmu karateristik dari bahasa yang memiliki aplikasi pada kriptografi (atau kriptologi) (seperti data frekuensi, kombinasi surat, pola universal, dll.) disebuh kriptolinguistik.

SEJARAH KRIPTOGRAFI DAN KRIPTANALISIS

SEBELUM zaman modern, kriptografi dilihat hanya semata-mata berhubungan dengan pesan rahasia (seperti enkripsi)-konversi pesan dari bentuk dapat dipahami menjadi bentuk yang tak dapat dipahami dan kembali lagi satu dengan yang lain, menjadikannya tak dapat dibaca oleh pencegat atau penyadap tanpa ilmu khusus (di mana sandi dibutuhkan untuk dekripsi pesan itu). Enkripsi digunakan untuk menyakinkan kerahasiaan di komunikasi, termasuk teknik untuk pemeriksaan integritas pesan, autentikasi identitas pengirim/penerima, tanda-

tangan digital, bukti interaktif dan komputasi keamanan, serta banyak lagi yang lain.

KRIPTOGRAFI KLASIK

scytale Yunani yang direkonstruksi kembali, alat sandi pertama kali

BENTUK awal dari penulisan rahasia membutuhkan lebih sedikit dari implementasi penulisan sejak banyak orang tidak dapat membaca. lawan yang lebih terpelajar, membutuhkan kriptografi yang nyata. Tipe sandi klasik utama ialah *sandi transposisi*, di mana mengatur aturan huruf pada pesan (contoh '*hello world*' menajdi '*ehlol owrdl*' pada skema pengubahan sederhana ini), dan sandi subtitusi, di mana secara sistematis mengganti huruf atau grup kata dengan kata lainnya dari grup kata (contoh '*fly at once*' menjadi '*gmz bu podf*' dengan mengganti setiap huruf dengan yang lain di alfabet Latin. Substitusi sandi pada awalnya disebut **sandi Caesar**, di mana setiap kata pada teks diganti degan huruf dari jumlah tetap pada posisi di alfabet. Laporan ***Suetonius*** menyebutkan **Julius Caesar** mengunakannya untuk berkomunikasi dengan jendral-jendralnya. *Atbash* merupakan contoh dari sandi Ibrani pada mulanya. Penggunaan awal kriptografi yang diketahui merupakan teks sandi yang diukir pada batu di Mesir (1900 sebelum Masehi), namun teks sandi ini digunakan hanya sebagai hiburan untuk pengamat terpelajar daripada cara untuk menyimpan informasi.

Yunani kuno menyebutkan telah mengetahui sandi (contoh sandi transposisi *scytale* yang diklaim telah digunakan oleh militer Sparta.*Steganografi* (menyembunyikan kehadiran pesan sehingga pesan tersebut menjadi rahasia) juga pertama kali diperkenalkan pada masa kuno. Contoh awal seperti, dari *Herodotus*, menyembunyikan pesan - sebuah tato pada kepala budaknya - di bawah rambut yang kembali tumbuh. Contoh yang lebih modern dari steganografi termasuk penggunaan tinta tak tampak, mikrodot, dan tanda air digital untuk menyembunyikan informasi.

Di India, Kamasutra dari *Vātsyāyana* yang berumur 2000 tahun berbicara dengan dua jenis sandi yang berbeda yang disebut Kautiliyam dan Mulavediya. Di Kautiliyam, substitusi kata sandi berdasarkan relasi fonetik, seperti vokal menjadi konsonan. Di Mulavediya, alfabet sandi terdiri dari kata-kata yang berpasangan dan bertimbal-balik.

Lembar pertama dari buku Al-Kindiyang menjelaskan bagaimana mengenkripsikan pesan

Teks sandi yang dihasilkan dengan sandi klasik (dan beberapa sandi modern) selalu mengungkapkan informasi statistik tentang teks awal, yang sering dapat digunakan untuk memecahkannya. Setelah ditemukannya analisis frekuensi oleh matematikawan Arab dan *polymath* Al-Kindi (juga dikenal sebagai *Alkindus*) pada abad ke-9, hampir semua jenis sandi menjadi lebih sulit dipecahkan oleh penyerang yang memiliki informasi tersebut. Seperti sandi klasik

yang masih populer hingga saat ini, meskipun lebih banyak dalam bentuk *puzzle*. Al-Kindi menuliskan buku kriptografi yang berjudul *Risalah fi Istikhraj al-Mu'amma* (*Risalah untuk Mnejermahkan Pesan Kriptografi*), yang menjelaskan teknik analisis frekuensi kriptanalisis yang pertama kalinya.

Mesin sandi berbentuk buku pada abad ke-16 milik Perancis, ditangan Henri II dari Perancis

Surat terenkripsi dari *Gabriel de Luetz d'Amaron, Duta Besar Perancis untuk Kerajaan Ottoman*, setelah 1546, dengan penguraian parsial

Pada dasarnya semua sandi tetap rentan kepada kriptanalisis menggunakan teknik analisis frekuensi hingga pengembangan dari sandi *polyalphabetic*, yang dijelaskan oleh *Leon Battista Alberti* sekitar tahun 1467, meskipun terdapat beberapa indikasi bahwa hal ini telah terlebih dahulu diketahui oleh Al-Kindi. Penemuan Alberti menggunakan sandi yang berbeda (seperti subtitusi alfabet) untuk beberapa bagian pesan (mungkin untuk setiap teks surat berturut-turut hingga akhir). Dia juga menemukan apa yang mungkin menjadi alat sandi otomatis untuk pertama kalinya, roda yang menerapkan pelaksanaan dari penemuannya. Pada sandi Vigenère polyalphabetic, enkripsi menggunakan kata kunci, yang mengatur substitusi surat berdasarkan surat mana dari kata kunci

yang digunakan. Pada pertengahan abad ke-19 Charles Babbage menunjukkan bahwa sandi *Vigenère* sangat rentan terhadap *pemeriksaan Kasiski*, namun hal ini diterbitkan pertama sekali kira-kira sepuluh tahun kemudian oleh Friedrich Kasiski.

Walaupun analisis frekuensi dapat sangat kuat dan menjadi teknik umum melawan banyak sandi, enskripsi masih sangat efektif dalam penerapannya, sebagaimana banyak kriptanalisis masih khawatir akan penerapannya. Memecahkan pesan tanpa menggunakan analisis frekuensi pada dasarnya membutuhkan pengetahuan sandi dan mungkin kunci yang digunakan, sehingga membuat spionase, penyuapan, pencurian, dll. Hal ini secara tegas mengakui kerahasiaan algoritma sandi pada abad 19 sangat tidak peka dan tidak menerapkan praktik keamanan pesan; faktanya, hal ini lebih lanjut disadari bahwa setiap skema kriptografi yang memadai (termasuk sandi) harus tetap aman walaupun musuh benar-benar paham tentang algoritma sandi itu sendiri. Keamanan kunci yang digunakan harus dapat menjamin keamanan pemegang kunci agar tetap rahasia bahkan ketika diserang sekalipun. Prinsip fundamental ini pertama kali dijelaskan pada tahun 1883 oleh *Auguste Kerckhoffs* dan secara umum dikenal dengan *Prinsip Kerckhoff*; secara alternatif dan blak-blakan, hal ini dijelaskan kembali oleh *Claude Shannon*, penemu teori informasi dan fundamental dari teori kriptografi, seperti pribahasa Shanon - 'musuh mengetahui sistemnya'.

Alat-alat bantu yang berbeda telah banyak digunakan untuk membantu sandi. Salah satu alat paling tua yang dikenali merupakan scytale dari Yunani, tangkai yang digunakan oleh **Spartan** sebagai alat bantuk untuk memindahkan sandi (lihat gambar di atas). Pada zaman pertengahan, alat bantu lainnya ditemukan seperti *jerejak sandi*, yang juga dikenal sebagai jenis steganografi. Dengan penemuan polialfabetik, sandi menjadi lebih mutakhir dengan bantuan disk sandi

milik Alberti, skema *tabula recta Johanner Trithemius*, dan silinder multi *Thomas Jefferson* (tidak banyak diketahui, dan ditemukan kembali oleh *Bazeries* sekitar tahun 1900. Banyak alat mekanik enkripsi/dekripsi ditemukan pada awal abad ke-20, dan beberapa telah dipatenkan, di antaranya *mesin rotor* yang dikenal dengan nama mesin Enigma digunakan oleh pemerintah dan militer Jerman dari akhir tahun 1920-an dan selama Perang Dunia II.

ERA KOMPUTER

KRIPTANALISIS dari alat mekanis baru terbukti lebih sulit dan melelahkan. Di Inggris, usaha kriptanalisis di *Bletchley Park* selama Perang Dunia II memacu perkembangan alat yang lebih efisien untuk membawa tugas yang berulang-ulang. Hal ini berujung pada pengembangan *Colossus*, komputer digital pertama sekali yang bekerja penuh secara elektronik, yang membantu penyandi untuk mendekripsikan mesin *Lorenz SZ40/42* milik tentara Jerman.

Seperti juga pengembangan komputer digital dan elektronik, kriptanalisis juga berkembang menjadi lebih kompleks. Lebih jauh lagi, komputer dapat mengenkripsi setiap jenis data yang mungkin dalam bentuk biner, tidak seperti chiper klasik yang hanya mengenkripsi teks bahasa tertulis; hal ini sangat baru dan signifikan. Penggunaan komputer telah menggantikan kirptografi bahasa, baik untuk desain chiper dan kriptanalisis. Banyak chiper komputer dapat dikategorikan dengan operasi mereka pada urutan biner (kadang dalam grup atau blok), tidak seperti skema mekanikal dan klasik, yang biasa memanipulasikan karakter tradisional (seperti surat dan digit) secara langsung. Bagaimanapun, komputer juga membantu kriptanalisis, yang mengimbangi hingga batas tertentu untuk kompleksitas chiper yang lebih tinggi. Namun, chiper modern yang baik telah mengungguli kriptanalisis; kasus ini biasa menggunakan kualitas chiper yang lebih efisien (seperti membutuhkan sumber daya yang

sedikit dan cepat, seperti memori atau kapabilitas CPU), sedang mendekripsikannya membutuhkan udaha dengan urutan yang lebih besar, dan lebih luas dibandingkan chiper yang lebih klasik, membuat kriptanalsis menjadi lebih tidak efisien dan tidak berguna.

Riset akademik kriptografi yang terbuka luas masih relatif baru; dimulai pada pertengahan 1970an. Pada saat ini, personel IBM mendesain algoritma yang menjadi standar enkripsi data Federal; Whitfield Diffie dan Martin Hellman mempublikasikan algoritma perjanjian mereka. dan algoritma RSA yang dipublikasikan oleh kolumnis Amerika Martin Gardner. Sejak saat itu, kriptografi telah sangat luas digunakan dalam dunia komunikasi, jaringan komputer, dan keamanan komputer secara umum. Beberapa teknik kriptografi modern dapat menyimpan kunci rahasianya jika menggunakan masalah matematika rumit, seperti faktorisasi integer atau masalah logaritma diskrit, jadi terdapat hubungan yang mendalam dengan matematika abstrak. Tidak ada bukti pasti bahwa teknik kriptografi aman. Saat terbaik, terdapat bukti bahwa beberapa teknik aman jika beberapa masalah komputasional sulit untuk dipecahkan, atau asumsi tertentu mengenai implementasi atau penggunakan praktikal bertemu.

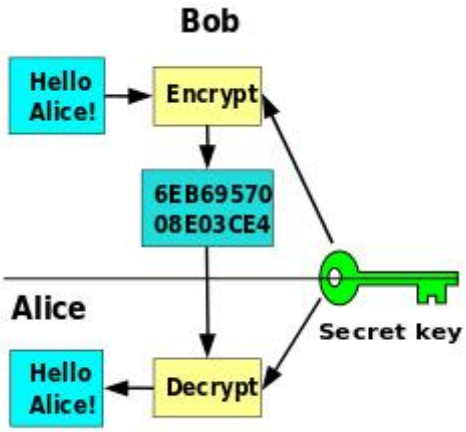
Semakin mengetahui sejarah kriptografi, algoritma kriptografi dan desainer sistem harus juga bijaksana mempertimbangkan pengembangan masa depan saat bekerja dengan desain mereka. Sebagai contoh, pengembangan kontinu pada computer processing power telah meningkatkan cakupan ***brute-force attack***, jadi ketika menentukan panjang kunci, diharuskan memilih kunci yang sulit. Efek potensial dari komputer kuantum telah dipertimbangkan oleh beberapa sistem desainer kriptografi; implementasi kecil dari mesin ini yang akan segera diumumkan mungkin akan membutuhkan perhatian khusus ketimbang hanya spekulatif.

Pada Dasarnya, pada awal abad ke-20, kriptografi secara singkat berkenaan dengan bahasa dan pola *lexicographic*. Sejak saat itu tekanannya telah berubah, dan kriptografi saat ini lebih luas menggunakan matematika, termasuk aspek teoriinformasi, kompleksitaskomputasional, statistik, kombinasi, aljabar, teori bilangan, dan matematika diskrit secara umum. Kriptografi juga merupakan cabang teknik, namun tidak biasa sebab hal ini berkaitan dengan pertentangan yang aktif, pintar, dan jahat; teknik jenis lain (seperti teknik kimia dan sipil) hanya membutuhkan gaya natural netral. Terdapat juga riset berkembang yang memeriksa hubungan antara masalah kriptografi dan fisika kuantum.

KRIPTOGRAFI MODEREN

BIDANG kriptografi yang lebih modern dapat dibagi dalam beberapa area studi. Beberapa yang paling penting akan dibahas di sini.

KRIPTOGRAFI KUNCI-SIMETRIS



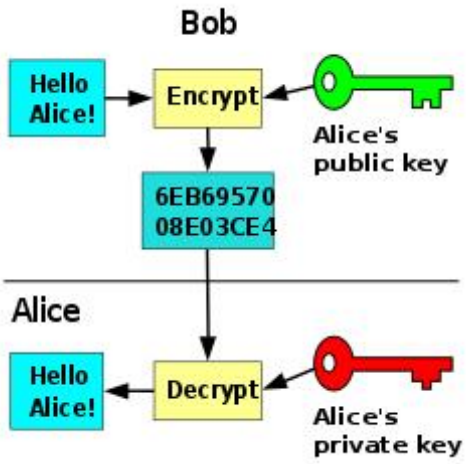
Kriptografi kunci-simetris, di mana satu kunci di gunakan untuk enkripsi dan dekripsi

KRIPTOGRAFI kunci-simetris merujuk pada metode enskripsi di mana kedua pengirim dan penerima membagi kunci yang sama (atau, walaupun kuncinya tidak mirip, namun dapat berhubungan dengan cara komputasi sederhana). Hal

ini menjadi satu-satunya jenis enkripsi yang ketahui publik hingga Juni 1976.*International Data Encryption Algorithm*, digunakan pada beberapa versi ***PGP (Pretty Good Privacy)*** untuk enkripsi tingkat tinggi, seperti e-mail Chiper kunci simetris diimplementasikan baik itu sebagai chiper blok atau chiper stream. Sebuah block chiper enchiper masukan pada blok plainteks sebagai lawanan untuk karakter individual, bentuk masukan yang digunakan oleh chiper aliran. *Standar Enkripsi Data (SED)* dan *Standar Enkripsi Lanjutan (SEL)* merupakan desain chiper blok yang telah ditunjuk sebagai *standar kriptografi* oleh pemerintah Amerika (walaupun penunjukan SED pada akhirnya ditarik setelah SEL diadopsi). Walaupun penarikannya sebagai standar resmi, SED (masih menjadi varian yang masih terbukti dan lebih aman) masih cukup terkenal; Hal ini digunakan oleh banyak penerapan dari enskripsi ATM hingga keamanan e-mail dan akses remote aman.Banyak chiper blok lainnya telah didesain dan dirilis, dengan kualitas yang bervariasi. Banyak telah juga yang dihancurkan, seperti *FEAL*. Beberapa chiper, yang berbeda dengan tipe 'blok', membuat berkas panjang material kunci yang panjang, di mana dikombinasikan dengan bit-bit teks atau karakter-karakter, sedikit mirip dengan *one-time pad*. Pada chiper aliran, aliran keluarannya diciptakan berdasarkan keadaan internal yang tersembunyi yang berubah saat chiper bekerja. Keadaan internal mulanya diatur menggunakan bahan kunci rahasia. *RC4* sangat luas digunakan sebagai chiper aliran. Chiper blok dapat digunakan sebagai chiper aliran. *Fungsi hash Kriptografi* merupakan algoritma kriptografi tipe ke-tiga. Fungsi ini mengambil segala panjang pesan sebagai input, dan panjang keluaran hash yang pendek dan tetap, yang dapat digunakan sebagai (sebagai contoh) tanda tangan digital. Untuk memiliki fungsi hash yang baik, penyerang tidak dapat mencari dua pesan yang dapat menghasilkan hash yang sama. MD4 merupakan fungsi hash pangjang yang sekarang telah dapat dipecahkan; MD5, varian yang lebih kuat

dari MD4, sudah luas digunakan namun dapat dipecahkan saat beroperasi. Agensi keamanan nasional Amerika mengembangkan serial Algoritma Hash Aman seperti fungsi hash MD5: SHA-0 ialah algoritma cacat yang kemudian ditarik; SHA-1 digunakan secara luas dan lebih aman dari MD5, namun kriptanalisis telah menemukan serangan padanya; keluarga SHA-2 meningkatkan performa SHA-1, namun belum secara luas digunakan; dan kewenangan Amerika mengatakan hal ini cukup bijaksana dari sudut pandang keamanan untuk mengembangkan standar baru "toolkit algoritma hash NIST secara keseluruhan untuk peningkatan kekuatan secara signifikan. Sehingga, pada tahun 2012, standar nasional Amerika memilih SHA-3 sebagai standar desain hash yang baru. *Message Authentication Code* (MAC) hampir mirip dengan fungsi hash kriptografi, kecuali terdapat kunci rahasia yang dapat digunakan untuk membuktikan nilai hash melalui serangkaian resep;[4] kerumitan tambahan yang melindungi skema serangan algoritma penyingkat sederhana, dan dianggap cukup menguntungkan.

KRIPTOGRAFI KUNCI-PUBLIK



Kriptogfari kunci- publik, di mana kunci-kunci yang berbeda digunakan untuk enkripsi dan dekripsi

KRIPTOSISTEM kunci-simetris menggunakan kunci yang sama untuk enkripsi dan dekripsi sebuah pesan, walaupun pesan atau kelompok pesan dapat memiliki kunci yang berbeda dari yang lain. Kerugian yang paling signifikan dari chiper simetris ialah kebutuhan *manajerial kunci* untuk menggunakannya secara aman. Setiap sepasang pihak komunikasi yang berbeda harus, idealnya, membagi kunci yang berbeda, dan juga membagi textchiper yang berbeda juga. jumlah kunci yang dibutuhkan meningkat dua kali lipat dari jumlah anggota jaringan, yang sangat cepat membutuhkan skema manajemen kunci komplek untuk menjaga semuanya tetap konsisten dan rahasia. Kesulitan dari menciptakan kunci rahasia yang aman di antara dua pihak yang saling berkomunikasi, ialah, ketika belum adanya jaringan aman di antara keduanya, juga kehadiran *chicken-and-egg problem* yang dianggap menjadi tantangan praktikal untuk pengguna kriptografi di dunia nyata.

Whitfield Diffie dan Martin Hellman, penulis jurnal pertama kriptografi kunci-publik]] Pada jurnal pionir tahun 1976, Whifiled Diffie dan Martin Hellman mengusulkan istilah dari kriptografi*kunci-publik* (juga, secara umum, disebut *kunci asimetris*) pada dua istilah yang berbeda namun secara matematis terdapat kunci yang berhubungan, yaitu kunci *publik* dan kunci *privat*. Sistem kunci pablik dikonstruksikan sangat baik sehingga kalkulasi dari satu kunci ('kunci privat') secara komputasional tidak mirip dengan (kunci 'publik') walaupun secara kebutuhan mereka mirip. Malah, kedua kunci dihasilkan secara rahasia, sebagai pasangan yang tidak berhubungan. Sejarawan David Kahn menjelaskan kriptografi kunci-publik sebagai "konsep baru paling revolusioner dalam bidang ini sejak substitusi polialfabetik yang ditemukan pada masa Renaissance

Dalam ekosistem kunci-publik, kunci publik dapat secara bebas terdistribusi, saat pasangannya kunci privat harus selalu terjaga rahasia. Pada sistem enkripsi kunci-publik, *kunci publik* digunakan untuk enkripsi, sedang *kunci*

privat atau *rahasia* digunakan untuk dekripsi. Sementara Diffie dan Hellman tidak dapat menemukan sistem seperti itu, mereka menunjukkan bahwa kriptografi kunci-publik memang benar mungkin dengan menunjukkan protokol Diffie-Hellman key exchange, sebuah solusi yang sekarang digunakan secara luas dalam komunikasi aman, mengizinkan dua kelompok untuk secara rahasia membagi kunci enkripsi.

Jurnal Diffie dan Hellman menyebar luas pada dunia akademi dalam mencari sistem enkripsi kunci-publik praktis. Lalu pada tahun 1978 **Ronald Rivest, Adi Shamir**, dan **Len Adleman**, menemukan solusi yang kini dikenal sebagai algoritma RSA.

Algoritma Diffie-Hellman dan RSA, sebagai tambahan dalam menciptakan contoh algoritma kunci-publik kualitas tinggi pertama yang dikenal publik, telah sangat luas digunakan. Yang lain termasuk Kriptosistem Cramer-Shoup, Enkripsi ElGamal, dan varian Teknik kurva eliptis.

Lalu, dokumen yang dipublikasikan pada tahun 1997 oleh ***Government Communication Headquarters (GCHQ)***, organisasi rahasia Inggris, mengungkapkan bahwa kriptografer di GCHQ telah mengantisipasi beberapa pengembangan akademik. Dilaporkan, sekitar tahun 1970, James H. Ellis telah memahami prinsip kriptografi kunci asimetris. Pada tahun 1973, Clifford Cocks menemukan sulisi yang esensialnya menyerupai algoritma RSA. Dan pada tahun 1974, Malcom J. Williamson diklaim telah mengembangkan algoritma pertukaran Diffie-Hellman.

icon kunci dari browser web Firefox, yang mengindikasikan TLS, sistem kriptografi kunci-publik, sedang digunakan

Kriptografi kunci-publik dapat juga digunakan untuk mengimplementasikan skema tanda tangan digital. Tanda tangan digital berhubungan dengan tanda tangan pada umumnya; mereka memiliki karateristik yang sama dimana mudah bagi pengguna untuk membuatnya, namun sangat sulit bagi orang lain untuk memalsukannya. Tanda tangan digital dapat juga secara permanen mengikat pada konten pesan yang sedang ditanda tangani; mereka lalu tidak dapat 'dipindahkan' dari satu dokumen ke dokumen yang lain, dan setiap usaha akan dapat terdeteksi. Pada skema tanda tangan digital, terdapat dua algoritma: satu untuk *menandatangani*, di mana kunci rahasia digunakan untuk memproses pesan (atau hash dari pesan, atau keduanya), dan satu untuk *verivikasi,* di mana kunci publik yang sesuai digunakan dengan pesan untuk memeriksa validitas tanda tangan. RSA dan DSA merupakan dua skema tanda tangan digital yang paling terkenal. Tanda tangan digital merupakan pusat dari operasi infrastruktur kunci publik dan banyak skema keamanan jaringan lainnya (seperti Transport Layer Security, VPN, dll).

Algoritma kunci publik paling sering didasari pada teori masalah kompleksitas komputasional, sering dengan teori bilangan. Sebagai contoh, kekuatan RSA berhubungan dengan masalah faktorisasi integer, sedangkan Diffie-Hellman dan DSA berhubungan dengan masalah logaritma diskrit. Baru-baru saja, *kriptografi kurva eliptis* telah ditemukan, sistem di mana keamanan yang didasari pada masalah teoretis bilangan yang melibatkan kurva eliptis. Dikarenakan kesulitan masalah pokok, kebanyakan algoritma kunci-publik melibatkan operasi seperti eksponensial dan perkalian aritmetika modular, di mana teknik ini secara komputasional lebih mahal ketimbang teknik yang digunakan pada banyak chiper blok, khususnya dengan ukuran kunci yang dibutuhkan. Hasilnya, kriptosistem kunci-publik seringnya merupakan *kriptosistem hybrid*, yang merupakan algoritma enkripsi kunci-simetris berkualitas tinggi digunakan untuk pesan itu

sendiri, sedang kunci simetris yang relevan dikirimkan dengan pesan, namun dienkripsikan menggunakan algoritma kunci publik. Hampir sama, skema tanda tangan hybrid sering digunakan, di mana fungsi hash kriptogfari dihitung secara komputer, dan hanya hash hasil yang ditanda tangani secara digital.

KRIPTANALISIS

VARIAN dari mesin Enigma, digunakan oleh militer dan otoritas Jerman pada akhir tahun 1920an saat Perang Dunia II, mengimplementasikan jenis chiper polialfabetik elektro-mekanikal yang rumit. Membaca dan memecahkan chiper Enigma di Biro chiper Polandia, selama 7 tahun sebelum perang, dan dekripsi di Bletchley Park, menjadi faktor penting kemenangan sekutu.

Tujuan dari kriptanalisis ialah untuk menemukan kelemahan dan ketidakamanan skema kriptografi, sehingga memungkinkan peningkatan atau perbaikan.

Terdapat kesalahpahaman umum bahwa setiap metode enkripsi dapat dipecahkan. Hubungan dengan karya Claude Shannon di Bell Labspada Perang Dunia II, membuktikan bahwa chiper *one-time pad tidak dapat dipecahkan, menemukan kunci utama yang acak, tidak pernah dapat digunakan lagi,*

*menyimpan rahasia dari setiap penyerang, dan memiliki panjang yang sama dan lebih besar dari pesan itu sendiri.Kebanyakan chiper, selain one-time pad, dapat dipecahkan dengan cara komputasional brute force attack, namun jumlah usaha yang dibutuhkan dapat sangat lama tergantung pada besar kunci, sebanding dengan usaha yang dibutuhkan untuk membuat chiper. Pada beberapa kasus, keamanan yang efektif dapat dicapai jika terbuktik bahwa usaha yang dibutuhkan (seperti "faktor kerja", dalam istilah Shannon) melebihi kemampuan dari setiap musuh. Ini berarti bahwa tidak boleh ada metode yang efisien (berbanding terbalik dengan metode brute force yang menghabiskan waktu) dapat ditemukan untuk memecahkan chiper. Sejak tidak ada bukti yang ditemukan, metode*one-time-pad *tetap secara teoretis merupakan chiper yang tidak dapat dipecahkan.*

Terdapat banyak jenis variasi serangan kriptanalitis, dan dapat diklasifikasikan dalam beberapa cara yang berbeda. Perbedaan yang sama yang diketahui oleh penyerang dan kemampuan yang tersedia. Pada serangan chiperteks saja, kriptanalis memiliki akses hanya pada chiperteks (kriptosistem modern yang baik biasanya secara efektif kebal terhadap serangan cipherteks). Pada serangan teks yang diketahui, kriptanalis memiliki akses pada chiperteks dan teks yang berhubungan (atau pada banyak pasangan). Pada serangan teks yang terpilih, kriptanalis memilih teks dan mempelajari tekschiper yang berhubungan (mungkin beberapa kali); sebagai contoh, istilah *gardeing* yang digunakan oleh Inggris selama Perang dunia II. Akhirnya, pada serangan chiperteks yang terpilih, kriptanalis dapat *memilih* chiperteks dan belajar teks yang berhubungan.Penting untuk diketahui, walaupun dengan jumlah yang sangat besar, ialah sering terjadinya kesalahan (umumnya pada desain atau penggunaan protokol kriptografis.

Monumen Polandia (*tengah*) kepada kriptologisnya yang memecahkan mesin cipher Enigma Jerman, dimulai pada tahun 1932, dan berakhir pada Perang Dunia II

Kriptanalisis cipher kunci-simetris biasanya melibatkan mencari serangan melawan cipher blok atau cipher aliran yang lebih efisien daripada setiap serangan yang dapat melawan cipher sempurna. Sebagai contoh, serangan brute force sederhana melawan DES membutuhkan satu teks yang diketahui dan dekripsi 2^{55}, membutuhkan kira-kira setengah dari kunci yang mungkin, untuk mencapai titik kemungkinan mengetahui kunci dapat ditemukan atau tidak. Namun hal ini tidak cukup menjadi jaminan; serangan kriptanalis linear terhadap DES membutuhkan 2^{43} teks yang deiketahui dan kira-kira 2^{43} operasi DES. Hal ini dianggap sebagai pengembangan dari serangan brute force.

Algoritma kunci-publik didasari pada kesulitan komputasional pada masalah yang beragam. Kesulitan yang paling terkenal ialah faktorisasi integer (seperti algoritma RSA yang didasari pada masalah yang berhubungan dengan faktor integer), namun masalah logaritma diskrit juga penting. Banyak kriptanalisis kunci-publik berhubungan dengan algoritma numerikal untuk menyelesaikan masalah komputasional ini, dan beberapa darinya, efisien (contoh dalam waktu pengerjaan). Sebagai contoh, algoritma yang paling dikenal untuk menyelesaikan kriptografi kurva eliptik versi logaritma diskrit sangat menghabiskan banyak waktu ketimbang algoritma yang paling dikenal untuk faktorisasi, paling tidak untuk masalah dengan besar yang lebih kurang sama. Oleh karena itu, segala hal harus sama, untuk mencapai kekuatan menahan serangan yang sama, teknik enkripsi berbasis faktor harus menggunakan kunci yang lebih besar dari teknik

kurva eliptik. Untuk alasan ini, kriptosistem kunci-publik yang didasari pada kurva eliptik telah menjadi lebih dikenal sejak penemuannya pada pertengahan tahun 1990an.

Sementara kriptanalisis menggunakan kelemahan pada algoritma itu sendiri, serangan pada kriptosistem lainnya didasari pada penggunaan dari algoritma pada perangkat yang nyata, yang disebut *side-channel attack*. Jika kriptanalis memiliki akses pada, sebagai contoh, jumlah waktu yang dibutuhkan perangkat untuk mengenkripsi jumlah teks atau memberikan laporan kesalahan pada password atau karakter pin, dia mungkin dapat menggunakan serangan waktu untuk memecahkan cipher yang paling tidak tahan pada analisis. Penyerang mungkin juga mempelajari pola dan panjang pesan untuk mendapatkan informasi berharga; hal ini dikenal sebagai *analisis trafik* dan cukup berguna untuk peringatan serangan. Administrasi kriptosistem yang lemah, seperti mengizinkan kunci yang terlalu pendek, akan membuat setiap sistem menjadi mudah diserang, terlepas dari faktor lainnya. Dan, tentu saja, teknik sosial, dan serangan lain melawan personel yang bekerja dengan kriptosistem atau pesan yang mereka pegang (seperti perampokan, pemerasan, blackmail, espionase, penyiksaan, dll) menjadi serangan yang paling produktif dari semuanya jenis serangan.

KRIPTOGFARI SEDERHANA

BANYAK karya teoritikal kriptografi berkaitan dengan kriptografi sederhana-algoritma dengan sifat kriptografi dasar-dan hubungannya pada masalah kriptografi lainnya. Alat kriptografi yang lebih sulit lalu diciptakan dari kriptografi sederhana ini. Kesederhanaan ini menyediakan sifat yang penting, yang digunakan untuk mengembangkan alat yang lebih kompleks yang

disebut *kriptosistem* atau *protokol kriptografi*, yang menjamin sifat keamanan level satu atau lebih tinggi. Bagaimanapun, perbedaan antara kriptografi *sederhana* dan kriptosistem, cukup tipis; sebagai contoh, algoritma RSA kadang disebut kriptosistem, dan kadang sederhana. Contoh tipikal kriptografi sederhana termasuk fungsi *pseudorandom*, fungsi satu-arah, dll.

KRIPTOSISTEM

SATU atau lebih kriptografi sederhana sering digunakan untuk mengembangkan algoritma yang lebih kompleks, disebut sistem kriptografi, atau *kriptosistem*. Kriptosistem (seperti enkripsi ElGamal didesan untuk menyediakan fungsi tertentu (seperti enkripsi kunci publik) sembari menjamin sifat keamanan tertentu keamanan (seperti serangan teks-terpilih) seperti pada model oracle acak. Kriptosistem menggunakan sifat kriptografi sederhana utama untuk mendukung sifat keamanan sistem. Tentu saja, karena perbedaan antara kriptosistem dan kriptografi tidak jelas, kriptosistem yang canggih dapat diperoleh dari kombinasi beberapa kriptosistem sederhana. Pada banyak kasus, struktur kriptosistem melibatkan komunikasi maju mundur di antara dua atau lebih kelompok dalam ruangan. (seperti di antara pengirim dari pesan aman dan penerimanya) atau melewati waktu (seperti data yang dilindungi dengan kriptografi). Kriptosistem yang seperti itu disebut *protokol kriptografi*.

Beberapa kriptosistem yang terkenal termasuk *enkripsi RSA*, tanda tangan Schnorr, enkripsi El-Gamal, PGP, dll. Kriptosistem yang lebih rumit melibatkan sistem uang elektronik, sistem *tanda-tangan enkripsi*, dll. Beberapa kriptosistem *teoritikal* termasuk *sistem pembuktian interaktif*,seperti *pembuktian pengetahuan-nol*), sistem untuk *pembagian rahasia*,dll.

Hingga saat ini, banyak sifat keamanan kriptosistem didemonstrasikan menggunakan teknik empirial atau menggunakan alasan ad hoc. Saat ini, terdapat upaya untuk mengembangkan teknik formal untuk menyelesaikan keamanan kriptosistem; Hal ini secara umum disebut *keamanan terbukti*. Ide umum dari keamanan terbukti ialah untuk memberikan argumen tentang kesulitan komputasional yang dibutuhkan untuk membahayakan aspek keamanan kriptosistem (dari setiap musuh).

Ilmu yang melihat seberapa baik implementasi dan integrasi kriptografi dalam penerapannya pada perangkat lunak disebut bidang tenik kriptografi dan teknik keamanan.

LARANGAN

KRIPTOGRAFI telah lama menjadi perhatian untuk mengumpulkan intelejen dan lembaga penegak hukum. Komunikasi rahasia dapat menjadi kriminal atau bahkan penghianatan. Karena fasilitasinya privasi, dan menurunkan sistem privasinya, kriptografi juga dianggap menarik untuk pendukung hak sipil. Oleh karena itu, terdapat sejarah dari masalah legal yang kontroversial mengenai kriptografi, khususnya sejak kehadiran komputer murah yang memungkinkan akses kriptografi kualitas tinggi.

Pada beberapa negara, bahkan penggunaan kriptografi secara domestik ialah, atau telah, dilarang. Hingga tahun 1999, Perancis telah menlarang penggunaan kriptografi secara domestik, walaupun banyak aturan yang kini telah dilonggarkan. Di Tiongkok, dan Iran, lisensi masih dibutuhkan untuk menggunakan kriptografi. Banyak negara telah melarang ketat penggunaan kriptografi.

Di antaranya Belarusia, Kazakhstan, Mongolia, Pakistan, Singapura, Tunisia, dan Vietnam.

Di Amerika, kriptografi legal digunakan pada kalangan domestik, namun terdapat banyak konflik mengenai masalah legal atau tidaknya kegunaan kriptografi. Satu masalah khas yang penting mengani ekspor kriptografi dan kriptografi perangkat lunak dan perangkat keras. Mungkin karena pentingnya kriptanalisis di Perang Dunia II dan ekspektasi bahwa kriptografi akan terus menjadi penting untuk keamanan nasional, banyak pemerintah Barat memiliki, pada satu titik, mengatur kriptografi secara ketat. Setelah Perang Dunia II, ialah ilegal di Amerika untuk menjual atau mendistribusikan teknologi enkripsi ke luar negeri; faktanya, enkripsi ditunjuk sebagai alat militer tambahan dan didaftarkan pada *Daftar Perlengkapan Militer Amerika Serikat*. Hingga pengembangan komputer personal, algoritma kunci asimetris (seperti teknik kunci publik), dan Internet, bukan terlalu menjadi masalah. Bagaimanapun, semakin Internet dan komputer menjadi tersedia dengan luas, semakin teknik enkripsi kualitas tinggi dikenal luas di seluruh dunia.

PEMBATASAN EKSPOR

PADA tahun 1990an, terdapat beberapa tantangan pada regulasi ekspor kriptografi. Setelah kode sumber program Pretty Good Privacy (PGP) milik Philip Zimmermann ditemukan dan digunakan Internet pada Juni 1991, terdapat komplain oleh RSA Security (yang kini dinamakan RSA Data Security, Inc.) menyebabkan investigasi kriminal yang panjang *US Costum Service* dan FBI, walaupun tidak ada tuntutan yang diberikan kepada Zimmermann. Daniel J. Bernstein, mahasiswa lulusan UC Berkeley, menuntut pemerintah Amerika mengenai beberapa aspek larangan berdasarkan undang-undang dasar negara Amerika. Pada tahun 1995 kasus Bernstein vs Amerika berakhir dengan

keputusan pada tahun 1999 mengenai kode sumber cetak untuk algoritma dan sistem kriptografi dilindungi sebagai kebebasan berbicara (atau *free speech*) oleh Konstitusi Amerika Serikat.

Pada tahun 1996, tiga puluh sembilan negara menandatangani Perjanjian Wassenaar, perjanjian pengawasan senjata yang mengatur ekspor senjata dan teknologi "kegunaan-ganda" seperti kriptografi. Perjanjian ini menetapkan bahwa penggunaan kriptografi dengan panjang-kunci pendek (*56-bit untuk enkripsi simetris, 512-bit untuk RSA*) tidak akan diatur ekspornya. Kriptografi yang diekspor dari Amerika menjadi kurang diatur diakibatkan perenggangan aturan pada tahun 2000;Tidak ada banyak larangan lagi pada besar kunci di perangkat lunak yang dijual bebas di Amerika. Sejak perenggangan dari larangan ekspor Amerika ini, dan karena komputer personal terhubung dengan Internet termasuk browser web Amerika seperti Firefox atau Internet Explorer, hampir setiap pengguna Internet di dunia memiliki akses pada kriptografi yang berkualitas via browser mereka (seperti, via Transport Layer Security. Program klien email Mozilla Thunderbird dan Microsoft Outlook dapat mentransmit dan menerima email melalui TLS, dan dapat mengirim dan menerima email terenkripsi dengan S/MIME. Banyak pengguna Internet tidak menyadari bahwa perangkat lunak aplikasi dasar mereka mengandung banyak kriptosistem yang sangat luas. Browser dan program email ini terdapat di mana-mana hingga pemerintah yang tugasnya mengatur penggunaan kriptografi secara umum pada dunia sipil tidak dapat menemukan cari yang praktis untuk mengatur distribusi atau penggunaan kriptografi pada kualitas ini, bahkan hukum yang dibuat saat ini oleh pemerintah hampir mustahil dapat dilaksanakan secara efektif.

KETERLIBATAN NSA

MASALAH lain yang sering diperdebatkan mengenai kriptografi di Amerika Serikat ialah pengaruh *National Security Agency* pada pengembangan dan aturan cipher. NSA terlibat dengan desain *Data Encryption Standard* selama pengembangannya di IBM dan dipertimbangkan oleh *National Bureau Standard* sebagai Biro untuk kriptografi. DES didesain untuk tahan terhadap kriptanalisis diferensial,Teknik kriptanalitik umum dan kuat diketahui oleh NSA dan IBM, yang menjadi diketahui publik hanya ketika hal ini ditemukan kembali pada akhir tahun 1980an.Menurut Steven Levy, IBM menemukan kriptanalisis differensial, namun NSA meminta teknik ini tetap terjaga rahasia. Teknik ini menjadi diketahui publik ketika Biham dan Shamir menemukan kembali dan mengumumkannya beberapa tahun kemudian. Keseluruhan keadaan ini mengilustrasikan kesulitan menentukan sumber daya dan pengetahuan apa saja yang penyerang mungkin miliki.

Contoh lain keterlibatan NSA ialah pada masalah chip Clipper pada tahun 1993, mikrochip enkripsi yang bertujuan menjadi bagian dari inisiatif pengawasan-kriptografi Capstone. Clipper dikritik secara luas oleh kriptografer untuk dua alsan. algoritma cipher (yang dinamakan Skipjack) yang dirahasiakan (dibongkar pada tahun 1998, jauh setelah Clipper pertama kali ditemukan). Cipher yang dirahasiakan menyebabkan keprihatinan bahwa NSA secara sengaja membuat cipher lemah untuk membantu usaha intelejennya. Keseluruhan inisiatif ini juga dikritik karena melanggar Prinsip Kerckhoff, karena skema yang termasuk pada *escrow key* diatur oleh pemerintah untuk digunakan menegakkan hukum, sebagai contoh metode penyadapan.

MANAJEMEN HAK DIGITAL

KRIPTOGRAFI merupakan pusat bagi manajemen hak digital (DRM), sebuah kelompok teknik teknologi mengatur penggunaan hak cipta, yang secara luas diimplementasikan dan disebar atas perintah beberapa hak cipta. Pada tahun 1998, Presiden Amerika Bill Clinton menanda-tangani (DMCA) Digital Millennium Copyright Act, yang mengkriminalisasikan segala produksi, penyebaran, dan penggunaan teknik dan teknologi kriptanalitik tertentu (telah ditemukan atau akan ditemukan); termasuk secara spesifik, segala ha lyang dapat digunakan untuk mengelabui skema teknologi DRM. Hal ini memiliki dampak yang nyata pada komunitas riset kriptografi sejak penjanjian ini dibuat di mana *selaga* riset kriptanalisis melanggar, atau dapat melanggar, DMCA. Undang-undang yang sama telah diberlakukan pada beberapa negara dan wilayah, termasuk implementasi Hak Cipta Direktif. Larangan yang sama juga tercantum pada perjanjian yang ditanda-tangani oleh anggota Organisasi Hak atas Kekayaan Intelektual Dunia.

Departemen Hukum dan Ham serta FBI Amerika belum menegakkan DMCA seperti yang ditakutkan oleh banyak pihak, namun hukumnya, bagaimanapun juga, tetap menjadi kontroversial. Niels Ferguson, peneliti kriptografi yang sangat dihormati, telah mengatakan bahwa dia tidak akan memberikan beberapa risetnya kepada desain kemananan Intelkarena ketakutannya dituntut di bawah hukum DMCA. Baik Alan Cox (pengembang kernel Linux kedua) dan Edward Felten (serta beberapa muridnya di Princeton) telah menghadapi banyak masalah terkait undang-undang ini. Dmitry Sklyarov ditangkap selama kunjungannya ke Amerika dari Rusia, dan dihukum selama lima bulan masa percobaan karena dugaan pelanggaran dengan DMCA atas pekerjaan yang telah Ia lakukan di Rusia, di mana hal itu legal di sana. Pada tahun 2007, kunci

kriptografi yang bertanggung jawab atas konten Blu-ray dan HD DVD tersebar luas di Internet. Atas kasus itu, MPAA mengirim banyak sekali peringatan penangkapan berdasarkan DMCA, sehingga serangan balasan dari Internet yang dipicu dari dampak yang dirasakan perihal kebebasan berbicara dan Penggunaan wajar.

PEMBUKAAN PAKSA KUNCI ENKRIPSI

DI Inggris, undang-undang Inggris memberikan izin kepada polisi untuk memaksa pelaku mengungkapkan berkas dekripsi atau memberikan password yang melindungi kunci enkripsi. Tidak memberikan kunci merupakan pelanggaran hukum, dan dikenakan hukuman dua hingga lima tahun penjara jika melibatkan keamanan nasional.[54] Penuntutan yang berhasil terlah terjadi di bawah undang-undang ini; pertama, di tahun 2009, menghasilkan 13 bulan penjara Hukum pemaksaan yang sejenis juga terdapat di Australia, Finlandia, Perancis, dan India memaksa terdakwa untuk memberikan kunci enkripsi atau password selama investigasi kriminal.

Di Amerika Serikat, kasus kriminal Amerika vs Fricosu menunjukkan jika surat perintah dapat memaksa seseorang untuk mengungkapkan kunci enkripsi atau password.***Elektronik Frontier Foundation*** **(EFF)** memperdebatkan masalah ini atas pelanggaran perlindungan hak asasi manusia berdasarkan Undang-Undang Dasar. Pada tahun 2002, pengadilan membuat undang-undang, bahwa terdakwa harus membawa perangkat keras yang tak terenkripsi ke persidangan.

Dalam banyak yuridikasi, status legal pemaksaan seperti ini masih belum jelas.

Bagian 2

PYTHON

Sejarah, Fitur

Paradigma	: Multiparadigma: berorientasi objek, imperatif, fungsional, prosedural, reflektif
Muncul Tahun	: 1991
Perancang	: Guido van Rossum
Pengembang	: Python Software Foundation
Rilis terbaru	: 3.4.2 /13 Oktober 2014; 2 tahun lalu
	2.7.8/13 Oktober 2014; 2 tahun lalu
Sistem pengetikan	: duck, dynamic, strong
Implementasi	: CPython, IronPython, Jython, Python for S60, PyPy
Dialek	: Cython, RPython, Stackless Python
Terpengaruh oleh	: ABC, ALGOL 68, C,C++, Dylan, Haskell,Icon, Java,Lisp,[Modula-3, Perl
Mempengaruhi	: Boo, Cobra, D, Falcon, Groovy, JavaScript, Ruby[8]
Sistem operasi	: Cross-platform
Lisensi	: Python Software Foundation License
Situs web	: python.org

PYTHON adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif.

Python mendukung multi paradigma pemrograman, utamanya; namun tidak dibatasi; pada pemrograman berorientasi objek, pemrograman imperatif, dan

pemrograman fungsional. Salah satu fitur yang tersedia pada python adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan manajemen memori otomatis. Seperti halnya pada bahasa pemrograman dinamis lainnya, python umumnya digunakan sebagai bahasa skrip meski pada praktiknya penggunaan bahasa ini lebih luas mencakup konteks pemanfaatan yang umumnya tidak dilakukan dengan menggunakan bahasa skrip. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi.

Saat ini kode python dapat dijalankan di berbagai platform sistem operasi, beberapa di antaranya adalah:

1) Linux/Unix
2) Windows
3) Mac OS X
4) Java Virtual Machine
5) OS/2
6) Amiga
7) Palm
8) Symbian (untuk produk-produk Nokia)

Python didistribusikan dengan beberapa lisensi yang berbeda dari beberapa versi. Namun pada prinsipnya Python dapat diperoleh dan dipergunakan secara **bebas**, bahkan untuk kepentingan komersial. Lisensi Python tidak

bertentangan baik menurut definisi ***Open Source*** maupun General Public License (GPL)

SEJARAH

PYTHON dikembangkan oleh **Guido van Rossum** pada tahun 1990 di CWI, Amsterdam sebagai kelanjutan dari bahasa pemrograman ABC. Versi terakhir yang dikeluarkan CWI adalah 1.2.

Tahun 1995, Guido pindah ke CNRI sambil terus melanjutkan pengembangan Python. Versi terakhir yang dikeluarkan adalah 1.6. Tahun 2000, Guido dan para pengembang inti Python pindah ke BeOpen.com yang merupakan sebuah perusahaan komersial dan membentuk BeOpen PythonLabs. Python 2.0 dikeluarkan oleh BeOpen. Setelah mengeluarkan Python 2.0, Guido dan beberapa anggota tim PythonLabs pindah ke DigitalCreations.

Saat ini pengembangan Python terus dilakukan oleh sekumpulan pemrogram yang dikoordinir Guido dan Python Software Foundation. Python Software Foundation adalah sebuah organisasi non-profit yang dibentuk sebagai pemegang hak cipta intelektual Python sejak versi 2.1 dan dengan demikian mencegah Python *dimiliki* oleh perusahaan komersial. Saat ini distribusi Python sudah mencapai versi 2.6.1 dan versi 3.0.

Nama Python dipilih oleh Guido sebagai nama bahasa ciptaannya karena kecintaan Guido pada acara televisi Monty Python's Flying Circus. Oleh karena itu seringkali ungkapan-ungkapan khas dari acara tersebut seringkali muncul dalam korespondensi antar pengguna Python.

FITUR

Beberapa fitur yang dimiliki Python adalah:

- memiliki kepustakaan yang luas; dalam distribusi Python telah disediakan modul-modul 'siap pakai' untuk berbagai keperluan.
- memiliki tata bahasa yang jernih dan mudah dipelajari.
- memiliki aturan *layout* kode sumber yang memudahkan pengecekan, pembacaan kembali dan penulisan ulang kode sumber.
- berorientasi objek.
- memiliki sistem pengelolaan memori otomatis (garbage collection, seperti java)
- modular, mudah dikembangkan dengan menciptakan modul-modul baru; modul-modul tersebut dapat dibangun dengan bahasa Python maupun C/C++.
- memiliki fasilitas pengumpulan sampah otomatis, seperti halnya pada bahasa pemrograman Java, python memiliki fasilitas pengaturan penggunaan ingatan komputer sehingga para pemrogram tidak perlu melakukan pengaturan ingatan komputer secara langsung.
- memiliki banyak faslitas pendukung sehingga mudah dalam pengoprasiannya.

Masukan / Keluaran

Contoh masukan :

```
nama = input("Masukkan nama Anda: ")
```

Contoh keluaran :

```
print ("Halo", nama, ":)"
```

Halo Dunia

Perintah ini biasanya digunakan untuk menguji keberhasilan pemasangan Python dalam komputer.

```
print ("Halo dunia!")
```

Keluaran yang seharusnya ditampilkan adalah seperti di bawah ini.

```
Halo dunia!
```

Kerangka Kerja (Framework)

Berikut ini beberapa perangkat kerja atau *Framework* yang menggunakan python:

- Django
- Cubicweb
- Pyramid
- Web.py
- Web2py

- Zope
- Flask
- Falcon
- Bottle
- Wezzy.web
- Giotto
- Grok

Bagian 3

LEMBAGA SANDI NEGARA

Lembaga Sandi Negara

Sthana Paroksharta Bhakti

KEGIATAN persandian dalam pemerintahan telah berlangsung sejak berdirinya Negara Kesatuan Republik Indonesia (NKRI), dimulai dari Jawatan Tehnik bagian B Kementerian Pertahanan pada masa perjuangan kemerdekaan baik di Jakarta maupun saat pemerintahan RI di Yogyakarta dan Pemerintahan Darurat RI di Bukittinggi, kemudian mendukung kegiatan diplomasi di Kementerian Luar Negeri dan Perwakilan RI di New Delhi, Den Haag, dan New York. Melalui perintah lisan Menteri Pertahanan tentang perlunya organisasi pelaksana fungsi persandian, maka dibentuk “Dinas Kode” Kementerian Pertahanan pada tanggal 4 April 1946, yang kemudian melembaga menjadi “Djawatan Sandi” dengan Surat Keputusan Menteri Pertahanan nomor 11/MP/1949 pada tanggal 2 September 1949. Dalam konteks lintasan sejarah inilah, tanggal 4 April ditetapkan sebagai Hari Persandian Republik Indonesia.

Melalui SK Presiden RIS nomor 65/1950, pada tanggal 14 Februari 1950, terjadi pemisahan struktur organisasi persandian dari Kementerian Pertahanan, yang

berada langsung di bawah Presiden. Pada 22 Februari 1972 menjadi "Lembaga Sandi Negara" dengan Keppres No. 7/1972. Sejalan dengan konsolidasi/penataan struktur kelembagaan Pemerintah, terjadi perubahan landasan hokum Lembaga Sandi Negara, berturut-turut pada 18 Juli 1994 dengan Keppres Nomor 54/1994, pada 7 Juli 1999 dengan Keppres Nomor 77/1999, dan terakhir dengan Keppres Nomor 103/2001.

Lembaga Sandi Negara sudah mengalami 6 (enam) masa kepemimpinan, dimulai dari Mayor Jenderal TNI Dr. Roebiono Kertopati dari tahun 1946-1984, diikuti kepemimpinan Laksamana Muda TNI Soebardo dari tahun 1986-1998, selanjutnya oleh Laksamana Muda TNI B.O. Hutagalung dari tahun 1998-2002, lalu Mayor Jenderal TNI Nachrowi Ramli, S.E. dari tahun 2002-2008, kemudian dari tahun 2009-2011 dibawah kepemimpinan Mayor Jenderal TNI Wirjono Budiharso, S.IP, dan sekarang dipimpin oleh Mayor Jenderal TNI Dr. Djoko Setiadi, M.Si.

Bagian 4

PROJEK KRIPTOGRAFI DAN PYTHON

PROJEK KRIPTOGRAFI DAN PYTHON

Spesifikasi dan Kebutuhan

Berjalan pada :[1] Windows 8 (64bit) python 3, Lenovo Ideapad 300S

[2] Backbox 4/Ubuntu 12.04 (32bit) python 2.7 Acer Aspire 5570 series (*tidak seluruh program)

[3] Windows 7 (32bit) python 3, Dell Inspiron N4050

Kebutuhan khusus : Pada dasarnya berjalan pada python 3

MEMBALIKKAN SANDI

Membalikkan sandi atau reverse cipher dapat dilakukan dengan cara mengubah urutan huruf pertama menjadi huruf terakhir dari kata atau kalimat, contohnya adalah “Hello world!” di enkripsi menjadi “!dlrow olleH”. Untuk melakukan dekripsi menjadi pesan asli cukup lakukan hal sebaliknya, yaitu mengubah urutan “!dlrow olleH”, dimana huruf pertama menjadi yang terakhir.

```
############################################################
# Membalikkan Sandi                                        #
# Editor  : Matius Celcius Sinaga                          #
# Author : http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

#pesan berisi kata/kalimat yang nantinya akan disusun secara terbalik
#contoh pesan yang akan diubah adalah "Hello World!" maka hasilnya "!dlroW olleH"
#nantinya akan menjadi Hello World
pesan = 'Indonesia Raya'

#mengenalkan ubah dan nantinya akan menyimpan string yang telah diubah
#ingat bahwa string yang digunakan adalah dua single qoute (' ')
#bukan double karakter
ubah = ''

#panjang pesan akan dikurangi 1 dari posisi awal untuk menentukan barisan
i = len(pesan) - 1

#pada while akan ditentukan apakah True maupun False
#jika nilai i yang dihasilkan True maka program dilanjukan pada blok selanjutnya
#jika nilai False maka akan langsung melompat ke print
while i >= 0:

    #melakukan penyimpanan nilai dan pengubahan string ke bentuk lain
    #dalam bentuk lain yang dimaksud disebut dengan enkripsi
    ubah = ubah + pesan[i]
    i = i - 1

#hasil yang telah diubah akan ditampilkan dan program dijalankan
```

```
print(ubah)
```

Output/Hasilnya adalah :

```
ayaR aisenodnI
```

Apabila anda mengubah isi dari variabel pesan menjadi “ayaR aisenodnI” (tanpa tanda kutip) akan menghasilkan teks asli dari proses membalikkan sandi yaitu Indonesia Raya.

Selamat anda sudah memulai mempelajari hal dasar mengenai kriptografi dan algoritmanya. Semangat untuk tetap melanjutkan. SEMANGAT !

SANDI CAESAR

Sandi Caesar atau Caesar Cipher adalah sandi yang digunakan oleh Julius Caesar 2.000 tahun yang lalu. Sandi ini sangat mudah untuk dipelajari namun mudah juga untuk ditebak.

```
#################################################################
# Sandi Caesar                                                  #
# Editor : Matius Celcius Sinaga                                #
# Author : # http://inventwithpython.com/hacking (BSD Licensed) #
#################################################################

#modul adalah program python yang memiliki fungsi tambahan
#yang  dimana dapat digunakan pada program lainnya
#modul pyperclip berada pada halaman lampiran buku ini
import pyperclip

pesan = 'Indonesia Raya'

#jumlah nilai langkah/sandi geser yang akan anda gunakan pada kunci
#silahkan ganti dengan angka yang anda inginkan
#kunci 9 berarti huruf yang pertama diubah menjadi huruf ke 9
#dalam urutan abjad
kunci = 9

#apabila akan melakukan enkripsi maka isi dengan enkripsi
#begitu juga dengan sebaliknya apabila ingin melakukan dekripsi
mode = 'encrypt'

#setiap huruf pada HURUF akan digunakan pada program
#pastikan tidak ada yang terlewatkan
HURUF ='ABCDEFGHIJKLMNOPWRSTUVWXYZ'

ubah = ''

#mengubah setiap huruf menjadi huruf kapital
pesan = pesan.upper()

# seluruh huruf akan diproses dan ditentukan nilai enkripsi/dekripsinya
for symbol in pesan:
    if symbol in HURUF:
```

```
        #enkripsi dan dekripsi pesan dilakukan disini
        #dengan menentukan jumlah symbol/huruf terlebih dahulu
        nomor = HURUF.find(symbol)
        if mode == 'encrypt':
            nomor = nomor + kunci
        elif mode == 'decrypt':
            nomor = nomor - kunci

        #jika pesan yang diproses lebih besar dari atau lebih kecil
        #atau sama dengan nol/tidak ada
        if nomor >= len(HURUF):
            nomor = nomor - len(HURUF)
        elif nomor < 0:
            nomor = nomor + len(HURUF)

        #jumlah symbol pada proses enkripsi/dekripsi ditambahkan
        #pada proses akhir pengubahan
        ubah = ubah + HURUF[nomor]
    else:
        #menambahkan simbol tanpa melakukan proses enkripsi/dekripsi
        ubah = ubah + symbol

#menampilkan hasil enkripsi/dekripsi pada tampilan
print(ubah)

#menyalin hasil enkripsi/dekripsi pada penyimpanan sementara
pyperclip.copy(ubah)
```

Output yang dihasilkan adalah :

```
JOEPOFTJB SBZB
```

MERETAS SANDI CAESAR DENGAN BRUTE-FORCE

Brute-force adalah suatu teknik kriptanalistik dengan menebak secara terus menerus terhadap kunci sandi, apabila kunci yang dihasilkan salah maka program akan mencoba kunci yang lain secara terus menerus hingga menemukan kunci yang tepat, kunci yang dihasilkan akan digunakan untuk melakukan dekripsi dan menampilkan pesan asli. Hal tersebutlah mengapa sandi Caesar (Caesar Cipher) tidak digunakan dewasa ini, selain jumlah kunci yang memungkinkan sandi untuk di tebak hanya memiliki kemungkinan 26 kunci dan membutuhkan hanya beberapa waktu (detik) untuk menemukan salah satunya. Untuk membuat data privasi anda aman, anda membutuhkan sandi yang memiliki kunci yang lebih banyak dan sulit untuk ditebak. Contohnya adalah dengan sandi transposisi.

Anda dapat melakukan hack/peretasan pada sandi Caesar dengan menggunakan teknik Kriptanalitik yang disebut dengan "brute-force". Pada program ini anda dapat memecahkan program sandi Caesar dengan sangat efektif, contoh sederhana mengapa lebih baik untuk tidak menggunakan sandi Caesar untuk penggunaan informasi rahasia.

Kerckhoffs's Principle dinamakan begitu setelah abad ke 19 seorang kriptografer bernama Auguste Kerckhoffs mengatakan bahwa sebuah sandi seharusnya tetap aman bahkan jika seseorang mengetahui bagaimana sandi bekerja dan memiliki teks sandi, yang dimaksud adalah orang lain bisa saja memiliki seluruh informasi kecuali kunci. Hal tersebut juga diulangi pada abad ke 20 oleh seorang matematikawan Claude Shannon sebagai Shannon's Maxim : "Musuh mengetahui sistem".

```
############################################################
# Meretas sandi Caesar dengan Brute-force                  #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################
```

```
pesan = 'JOEPOFTJB SBZB'
#kumpulan HURUF yang digunakan untuk menebak huruf kunci
HURUF = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

#coba seluruh kunci yang memungkinkan
for kunci in range(len(HURUF)):

    #ubah tetap memiliki string kosong seperti sebelumnya telah dijelaskan
    ubah = ''

#bagian akhir dari proram ini hampir sama dengan program asli Caesar Cipher
#kode enkripsi dan dekripsi berjalan dan melakukan tugasnya pada string pesan
    for symbol in pesan:
        if symbol in HURUF:
            #mencari jumlah simbol dalam huruf
            nomor = HURUF.find(symbol)
            nomor = nomor - kunci

            #menangani proses dimana jumlah lebih dari 26 atau kurang dari 0
            if nomor < 0:
                nomor = nomor + len(HURUF)
            #tambahkan jumlah simbol pada akhir pengubahan
            ubah = ubah + HURUF[nomor]
        else:
            #tambahkan symbol tanpa melakukan enkripsi/dekripsi
            ubah = ubah + symbol

  #tampilkan kunci yang telah dicoba selama melakukan dekripsi
    print('Key #%s: %s' % (kunci, ubah))
```

Output yang dihasilkan adalah :

```
Key #0: JOEPOFTJB SBZB
Key #1: INDONESIA RAYA
Key #2: HMCNMDRHZ QZXZ
Key #3: GLBMLCQGY PYWY
Key #4: FKALKBPFX OXVX
Key #5: EJZKJAOEW NWUW
Key #6: DIYJIZNDV MVTV
Key #7: CHXIHYMCU LUSU
Key #8: BGWHGXLBT KTRT
Key #9: AFVGFWKAS JSQS
Key #10: ZEUFEVJZR IRPR
Key #11: YDTEDUIYQ HQOQ
Key #12: XCSDCTHXP GPNP
Key #13: WBRCBSGWO FOMO
Key #14: VAQBARFVN ENLN
Key #15: UZPAZQEUM DMKM
Key #16: TYOZYPDTL CLJL
Key #17: SXNYXOCSK BKIK
Key #18: RWMXWNBRJ AJHJ
Key #19: QVLWVMAQI ZIGI
Key #20: PUKVULZPH YHFH
Key #21: OTJUTKYOG XGEG
Key #22: NSITSJXNF WFDF
Key #23: MRHSRIWME VECE
Key #24: LQGRQHVLD UDBD
Key #25: KPFQPGUKC TCAC
```

Sesuai dengan kunci yang digunakan pada sandi Caesar , maka urutan kunci yang menghasilkan pesan asli berada pada angka tersebut juga.

MELAKUKAN ENKRIPSI DENGAN SANDI TRANSPOSISI

Anda telah membuktikan bahwa sandi Caesar tidak cukup untuk melindungi data anda agar tidak mudah ditebak maupun dengan mudah diketahui oleh orang lain. Sandi Transposisi memiliki kemampuan melakukan perubahan karakter dengan karakter lainnya sehingga membuat pesan asli sulit untuk ditebak.

```
############################################################
# Enkripsi Transposisi                                     #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

import pyperclip

#fungsi (fungsi main) memiliki statemen def
#def tidak disebut sebagai fungsi bernama main()
#def berarti membuat, atau defining, sebuah fungsi bernama main()
#yang kemudian dapat anda panggil/digunakan kembali pada program
def main():
    pesanSaya = 'Indonesia Raya Tanah Airku:IRTA'
    kunciSaya = 8

    sanditeks = enkripsiPesan(kunciSaya, pesanSaya)

    #menampilkan hasil enkripsi dalam bentuk ciphertext pada layar
    #dengan a | (dikenal sebagai "pipe" karakter) setelah hal tersebut akan ada ruang
    #pada akhir pesan yang telah dienkripsi
    print(sanditeks + '|')

    #salin string yang telah dienkripsi dalam chipertext pada tampilan layar
    pyperclip.copy(sanditeks)

def enkripsiPesan(kunci, pesan):
   #setiap string dalam ciphertext menghasilkan sebual kolom
    sanditeks = [''] * kunci

    #perulangan dilakukan pada setiap kolom dalam ciphertext
```

```
    for col in range(kunci):
        pointer = col

        #tetap melakukan perulangan hingga pointer berada pada nilai pesan terakhir
        while pointer < len(pesan):
            #menempatkan karakter pada titik pesan yang terakhir
            #pada kolom di dalam daftar ciphertext
            sanditeks[col] += pesan[pointer]

            #pindah pada akhit pointer
            pointer += kunci

    #ubah daftar ciphertext hingga nilai string tunggal dan mengulang
    return ''.join(sanditeks)


#jika enkripsitransposisi.py sedang berjalan (didalam modul yang telah diimport)
#panggil fungsi main()
if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Iaakn nudRa:oahIny ReaATs iAiTr|
```

Contoh gambar sederhana untuk menjelaskan program diatas adalah sebagai berikut :

I	d	n	s
a	R	y	(spasi)
A	a	(spasi)	I
k	:	R	A
n	o	e	I
(spasi)	a	a	T
n	h	A	R
u	l	T	|

DEKRIPSI DENGAN SANDI TRANSPOSISI

Sesuai dengan gambar yang sebelumnya pada enkripsi transposisi maka anda akan mencoba meretas kembali dengan skema dan gambaran dari program dan gambar yang dihasilkan. Tetap perhatikan dan hati-hati.

Cara melakukan dekripsi :

1. Hitung jumlah dari kolom yang kamu akan ambil dengan panjang dari pesan dan dibagi dengan kunci. Ulangi hal tersebut untuk bagian lainnya.

2. Bayangkan jumlah dari kotak. Jumlah kolom yang sudah dihitung pada langkah 1. jumlah dari baris juga memiliki hal yang sama dengan kunci.

3. Hitung jumlah dari kotak yang menjadi bayang/tidak terpakai/spasi/ruang kosong dengan menghitung jumlah dari julah kotak yang terpakai dan tidak dan kurangi dengan panjang teks pesan yang dihasilkan.

4. Isi setiap karakter dari sandi teks dimulai dari baris paling atas dan berlanjut dari kiri ke kanan. Lewati setiap ruang kosong yang anda tentukan sebelumnya.

5. Tetapkan teks awal dengan membaca dari kolom yang paling kiri hingga mencapai yang paling akhir dan lanjutkan hingga mencapai kolom berikutnya.

```
############################################################
# Dekripsi dengan sandi Transposisi                        #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

#program math sudah tersedia disaat anda melakukan instalasi python
import math, pyperclip

def main():
    #ubah disini apabila ingin mengganti pesan dan kunci
```

```
    #pesanSaya
    #kunciSaya
    pesanSaya = 'Iaakn nudRa:oahIny ReaATs iAiTr|'
    kunciSaya = 8

    teksawal = dekripsiPesan(kunciSaya, pesanSaya)

    #tampilkan dengan pipe ("|") setelah hal tersebut terbentuk
    #berikan spasi pada akhir pesan yang telah di dekripsi
    print(teksawal + '|')
    pyperclip.copy(teksawal)

def dekripsiPesan(kunci, pesan):
    #fungsi dekripsi transposisi akan menyimulasikan "kolom dan "baris"
    #pada tiap-tiap teks awal yang dituliskan berdasarkan daftar pada string
    #pertama, anda harus menghitung beberapa nilai
    #jumlah "kolom' dalam transposisi
    #math.ceil adalah operator pembagian dimana akan menghasilkan nilai bulat
    #contohnya jika nilai yang dihasilkan 4,2 maka akan dibulatkan menjadi 4
    #jika nilai yang dihasilkan 4,9 maka akan dibulatkan menjadi 5
    jumlahKolom = math.ceil(len(pesan) / kunci)

    #jumlahBaris adalah kunci
    jumlahBaris = kunci

    #jumlah kotak yang terisi adalah jumlah kolom baris dan pesan
    jumlahKotakyangTerisi = (jumlahKolom * jumlahBaris) - len(pesan)

    #setiap string yang berada pada teks awal menghasilkan kolom
    teksawal = [''] * jumlahKolom

    #setelah karakter dienkripsi maka pesan selanjutnya
    kolom = 0
    baris = 0

    for symbol in pesan:
        teksawal[kolom] += symbol
        kolom += 1
      #setelah itu akan dilanjutkan pada kolom selanjutnya
```

```
        #jika ada kolom lebih atau adanya baris yang tidak sesuai, maka
        #kembali pada kolom pertama dan melanjutkannya
        if (kolom == jumlahKolom) or (kolom == jumlahKolom - 1 and baris >=
jumlahBaris - jumlahKotakyangTerisi):
            kolom = 0
            baris += 1

    return''.join(teksawal)

#jika transposisidekripsi.py sudah berjalan (setelah module di import)
#panggil fungsi if  main()
if __name__=='__main__':
    main()
```

Output yang dihasilkan adalah :

```
Indonesia Raya Tanah Airku:IRTA||
```

IRTA hanya sebatas singkatan dari Indonesia Raya Tanah Airku pengisi ruang kosong untuk memenuhi perhitungan “kotak-bayang/ruang kosong”.

MEMPROGRAM SEBUAH PROGRAM UNTUK MELAKUKAN TEST PADA PROGRAM ANDA

Anda dapat menggunakan keahlian anda lebih dari sekedar dalam menulis program. Anda dapat juga memprogram sebuah program untuk menguji program yang anda miliki apakah bekerja sebagaimana mestinya meskipun dengan nilai masukan yang berbeda-beda.

```
############################################################

# Test/ujicoba Transposisi                                 #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

#random dan sys sudah tersedia pada python saat anda melakukan instalasi
#enkripsitransposisi dan dekripsitransposisi adalah program sebelumnya
#tempatkan enkripsitransposisi dan dekripsitransposisi berada pada folder yang sama
import random, sys, enkripsitransposisi, dekripsitransposisi

def main():
    #mengatur settingan acak "seed" menjadi nilai statis
    #mengubah algoritma dan menentukan nilai 42
    #hal ini disebut juga sebagai PSEUDORANDOM NUMBERS
    #pada random.randint juga menghasilkan PSEUDORANDOM NUMBERS
    #secara teknis nilai yang dihasilkan random.randint() sesungguhnya tidaklah acak
    #dan hasil algoritmanya dapat anda prediksi
    random.seed(42)

    #batasi jumlah test hanya mencapai 20 test saja
    for i in range(20):

      #contohnya jika anda menentukan seed antara 1 hingga 10 akan menentukan nilai acak 7
        pesan = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' * random.randint(4, 40)

        #ubah pesan string menjadi daftar yang akan di acak
        pesan = list(pesan)
        random.shuffle(pesan)
```

```
        pesan = ''.join(pesan)
      #list/daftar diubah menjadi string

        print('Test #%s: "%s..."' % (i+1, pesan[:50]))

        #cek seluruh kemungkinan kunci untuk setiap pesan
        for kunci in range(1, len(pesan)):
            dienkripsi = enkripsitransposisi.enkripsiPesan(kunci, pesan)
            didekripsi = dekripsitransposisi.dekripsiPesan(kunci, dienkripsi)

            #jika dekripsi tidak cocok dengan pesan asli
            #akan menampilkan error dan keluar
            if pesan != didekripsi:
                print('Tidak cocok dengan kunci %s dan pesan %s.' % (kunci, pesan))
                print(didekripsi)
                sys.exit()

    print('Uji coba sandi Transposisi berhasil.')

        #jika testransposisi.py berjalan (dengan mengimport sebuah modul
        #panggil main() fungsi
if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Test #1: "KQDXSFQDBPMMRGXFKCGIQUGWFFLAJIJKFJGSYOSAWGYBGUNTQX..."
Test #2: "IDDXEEWUMWUJPJSZFJSGAOMFIOWWEYANRXISCJKXZRHMRNCFYW..."
Test #3: "DKAYRSAGSGCSIQWKGARQHAOZDLGKJISQVMDFGYXKCRMPCMQWJM..."
Test #4: "MZIBCOEXGRDTFXZKVNFQWQMWIROJAOKTWISTDWAHZRVIGXOLZA..."
Test #5: "TINIECNCBFKJBRDIUTNGDINHULYSVTGHBAWDQMZCNHZOTNYHSX..."
Test #6: "JZQIHCVNDWRDUFHZFXCIASYDSTGQATQOYLIHUFPKEXSOZXQGPP..."
Test #7: "BMKJUERFNGIDGWAPQMDZNHOQPLEOQDYCIIWRKPVEIPLAGZCJVN..."
Test #8: "IPASTGZSLPYCORCVEKWHOLOVUFPOMGQWZVJNYQIYVEOFLUWLMQ..."
Test #9: "AHRYJAPTACZQNNFOTONMIPYECOORDGEYESYFHROZDASFIPKSOP..."
Test #10: "FSXAAPLSQHSFUPQZGTIXXDLDMOIVMWFGHPBPJROOSEGPEVRXSX..."
Test #11: "IVBCXBIHLWPTDHGEGANBGXWQZMVXQPNJZQPKMRUMPLLXPAFITN..."
Test #12: "LLNSYMNRXZVYNPRTVNIBFRSUGIWUJREMPZVCMJATMLAMCEEHNW..."
Test #13: "IMWRUJJHRWAABHYIHGNPSJUOVKRRKBSJKDHOBDLOUJDGXIVDME..."
Test #14: "IZVXWHTIGKGHKJGGWMOBAKTWZWJPHGNEQPINYZIBERJPUNWJMX..."
Test #15: "BQGFNMGQCIBOTRHZZOBHZFJZVSRTVHIUJFOWRFBNWKRNHGOHEQ..."
Test #16: "LNKGKSYIPHMCDVKDLNDVFCIFGEWQGUJYJICUYIVXARMUCBNUWM..."
Test #17: "WGNRHKIQZMOPBQTCRYPSEPWHLRDXZMJOUTJCLECKEZZRRMQRNI..."
Test #18: "PPVTELDHJRZFPBNMJRLAZWRXRQVKHUUMRPNFKXJCUKFOXAGEHM..."
Test #19: "UXUIGAYKGLYUQTFBWQUTFNSOPEGMIWMQYEZAVCALGOHUXJZPTY..."
Test #20: "JSYTDGLVLBCVVSITPTQPHBCYIZHKFOFMBWOZNFKCADHDKPJSJA..."
Uji coba sandi Transposisi berhasil.
```

MENGENKRIPSI DAN MENDEKRIPSI FILE

Pada program sebelumnya anda hanya melakukan enkripsi dan dekripsi yang mungkin terasa sangat sederhana, dengan cara mengubahnya menjadi kata atau kalimat acak. Namun bagaimana bila anda mencobanya terhadap file yang sebenarnya ? pada hal ini anda akan mencoba pada file .txt (dot teks) yang digunakan untuk menyimpan kata atau teks, biasanya file seperti ini digunakan pada format penyimpanan notepad (pada Windows), TextMate atau TextEdit (pada OS X) atau gedit (pada Linux) dan beberapa program editor lainnya. Anda juga bisa menyimpan file dengan format .txt pada IDLE file editor bawaan Python30 lalu menyimpannya dalam extensi .txt yang biasanya anda menggunakan ekstensi .py .

```
#############################################################
# Transposisi File sandi                                    #
# Editor : Matius Celcius Sinaga                            #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
#############################################################

import time, os, sys, enkripsitransposisi, dekripsitransposisi

def main():
    #siapkan sebuah file teks berisi kata maupun kalimat
    #dengan nama file matiuscelciussinaga.txt (txt format teks)
    masukkanNamaFile = 'matiuscelciussinaga.txt'
    #jika nama file sudah ada dengan file yang akan anda gunakan
    #maka program akan melakukan perubahan pada nama file matiuscelciussinaga.txt
    namaFileKeluaran = 'matiuscelciussinaga.encrypted.txt'
    kunciSaya = 10
    #tentukan disini apakah dilakukan enkripsi atau dekripsi
    modeSaya = 'encrypt'


  #jika file yang dimaksudkan belum ada, maka program terlebih dahulu melakukan eliminasi
    if not os.path.exists(masukkanNamaFile):
```

```
        print('File %s yang dimaksud tidak ada. Silahkan keluar...' %
(masukkanNamaFile))
        sys.exit()

    #jika file yang dimaksud sudah ada, menghasilkan pilihan
    #untuk melanjutkan atau untuk berhenti
    if os.path.exists(namaFileKeluaran):
        print('Tulis ulang file %s. (C)ontinue atau (Q)uit?' % (namaFileKeluaran))
        response = input('> ')
        #startswith akan mengenalkan bahwa nilai masukkan yang dimulai dengan huruf
C
        if not response.lower().startswith('c'):
            #secara teknis user tidak memberi nilai masukkan C maupun Q
            #karena program akan keluar disaat sys.exit() dipanggil
            sys.exit()

    #membaca pesan dari file yang telah dimasukkan
    objekFile = open(masukkanNamaFile)
    isi = objekFile.read()
    objekFile.close()

    #fungsi dari title() akan mengubah huruf pertama
    #setiap kalimat menjadi huruf kapital
    print('%sing...' % (modeSaya.title()))

    #mengukur panjang enkripsi/dekripsi yang dilakukan
    #time.time() akan menghitung berapa lama waktu program berjalan
    #dan menyimpan variabel waktuAwal
    waktuAwal = time.time()
    if modeSaya == 'encrypt':
        ubah = enkripsitransposisi.enkripsiPesan(kunciSaya, isi)
    elif modeSaya == 'decrypt':
        ubah = dekripsitransposisi.dekripsiPesan(kunciSaya, isi)
    jumlahWaktu = round(time.time() - waktuAwal, 2)
    print('%sion time: %s seconds' % (modeSaya.title(), jumlahWaktu))

    #menghasilkan pesan yang diubah menjadi file yang baru untuk ditampilkan
    FileObjekKeluaran = open(namaFileKeluaran, 'w')
    FileObjekKeluaran.write(ubah)
```

```
    FileObjekKeluaran.close()

    print('Done %sing %s (%s characters).' % (modeSaya, masukkanNamaFile, len(isi)))
    print('%sed file is %s.' % (modeSaya.title(), namaFileKeluaran))


#jika transposisifilechiper.py telah berjalan (termasuk seluruh modul didalamnya)
#panggil fungsi main()
if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Encrypting...
Encryption time: 0.0 seconds
Done encrypting matiuscelciussinaga.txt (26 characters).
Encrypted file is matiuscelciussinaga.encrypted.txt.
```

MENDETEKSI BAHASA DALAM KEBIASAAN MEMPROGRAM

Dapatkah komputer mengerti bahasa manusia ? Sebenarnya tidak, dan bagaimanapun pada akhirnya komputer tidak akan menjadi manusia seutuhnya dan tak mampu berbahasa/berkomunikasi. Komputer tak benar-benar mengerti apa itu perhitungan, memainkan peran sebagai lawan dalam permainan komputer atau bahkan menjadi peralatan seperti pasukan robot militer. Namun instruksi-instruksi yang diberikan yang akan menentukan bagaimana hal tersebut menjadi kebiasaan dalam menanggapi sesuatu yang dimana hal ini sebenarnya sangat-sangatlah rumit dalam penjelasannya terutama bagaimana komputer mampu menyelesaikan masalah komputer lainnya, memenangkan permainan komputer atau memberikan anda peluang masa depan yang lebih baik juga bagi kemajuan manusia.

```
#################################################################
# Deteksi Bahasa                                                #
# Editor : Matius Celcius Sinaga                                #
# Author : # http://inventwithpython.com/hacking (BSD Licensed) #
#################################################################

#untuk menggunakan program ini gunakan python shell
#ketik "import deteksibahasa" (tanpa tanda kutip)
#program ini harus berada pada satu folder bersama "dictionary.txt"
#"dictionary.txt" file berisi seluruh bahasa silahkan anda ubah
#atau cari dictionary lainnya, Github atau penyedia source code lain
HurufBESAR = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
Huruf_Dan_Spasi = HurufBESAR + HurufBESAR.lower() + ' \t\n'

#melakukan pengenalan untuk memulai jalankanKamus
#pada bagian ini akan melakukan pengecekan dan membuka file dictionary
#apabila cekKata sudah benar maka program akan selesai
def memuatKamus():
    fileKamus = open('dictionary.txt')
    cekKata = {}
    for kata in fileKamus.read().split('\n'):
        cekKata[kata] = None
```

```
    fileKamus.close()
    return cekKata

#cek bahasa apakah sama dengan fungsi dengan memuat kamus
KATA_BAHASA = memuatKamus()

#melakukan pengenalan untuk melakukan perhitungan jumlah kata yang digunakan
#jika ditemukan kata yang cocok atau dimungkinkan akan dibandingkan dengan
#jumlah kata yang cocok
def hitungJumlahKata(pesan):
    pesan = pesan.upper()
    pesan = bukanKata(pesan)
    kemungkinanKata = pesan.split()

    if kemungkinanKata == []:
        #jika tidak ditemukan satupun huruf maka hasilkan nilai 0
        return 0.0
    cocok = 0
    for kata in kemungkinanKata:
        if kata in KATA_BAHASA:
            cocok += 1
    return float(cocok) / len(kemungkinanKata)

#melakukan pendefinisian yang tidak termasuk di dalam kata
def bukanKata(pesan):
    hurufSaja = []
    for symbol in pesan:
        if symbol in Huruf_Dan_Spasi:
            hurufSaja.append(symbol)
    return ''.join(hurufSaja)

#melakukan pendefinisian dari cekBahasa yang dimana terdapat pesan
#jumlah kata dan huruf yang diijinkan dalam program yang akan dimuat
#secara umum, 20% dari kata seharusnya ada pada file dictionary
#dan 85% dari seluruh karakter di dalam pesan berisi kata dan spasi
#bukan tanda baca atau angka
def cekBahasa(pesan, IjinkanJumlahKata=20, IjinkanJumlahHuruf=85):
    kataYangCocok = hitungJumlahKata(pesan) * 100 >= IjinkanJumlahKata
    nomorHuruf = len(bukanKata(pesan))
```

```
    IjinkanJumlahKalimat = float(nomorHuruf) / len(pesan) * 100
    hurufYangCocok = IjinkanJumlahKalimat >= IjinkanJumlahHuruf
    return kataYangCocok and hurufYangCocok
```

Program ini tidak menghasilkan tulisan atau cuplikan yang dapat ditampilkan sebagai output, nantinya program ini akan diimport oleh program lain.

Jika anda ingin mencoba melakukan cekBahasa silahkan import pada IDLE dan ikuti penjelasan pada komentar program di awal.

MELAKUKAN PERETASAN TERHADAP SANDI TRANSPOSISI

Untuk melakukan peretasan terhadap sandi transposisi, anda akan menggunakan cara brute-force. Dengan ribuan kata kunci, kunci yang benar akan menghasilkan nilai yang dapat terbaca dalam bahasa inggris. Anda telah mengembangkan program deteksi bahasa pada bagian terakhir program dimana hal tersebut dapat membantu anda jika menemukan kunci yang benar.

```
#####################################################################
# Meretas Sandi Transposisi                                         #
# Editor : Matius Celcius Sinaga                                    #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)     #
#####################################################################

#jalankan program terlebih dahulu tekan enter pada shell saat melakukan proses
import pyperclip, deteksibahasa, dekripsitransposisi

def main():
    #anda akan melakukan bruteforce pada pesan ini
    #silahkan dicopy-paste jika ingin melakukan eliminasi
    pesanSaya = """Cb b rssti aieih rooaopbrtnsceee er es no npfgcwu  plri ch nitaalr eiuengiteehb(e1  hilincegeoamn  fubehgtarndcstudmd  nM  eu eacBoltaeteeoinebcdkyremdteghn.aa2r81a condari fmps" tad  l t oisn sit u1rnd stara nvhn fsedbh ee,n  e necrg6  8nmisv l nc muiftegiitm tutmg cm shSs9fcie ebintcaets h aihda  cctrhe  ele  1O7  aaoem  waoaatdahretnhechaopnooeapece9etfncdbgsoeb uuteitgna.rteoh add e,D7c1Etnpneehtn beete" evecoal lsfmcrl iu1cifgo ai. sl1rchdnheev sh meBd ies e9t)nh,htcnoecplrrh ,ide hmtlme. pheaLem,toeinfgn t e9yce da' eN eMp a ffn Fc1o ge eohg dere.eec s nfap yox hla yon. lnrnsreaBoa t,e eitsw il ulpbdofgBRe bwlmprraio po  droB wtinue r Pieno nc ayieeto'lulcih sfnc  ownaSserbereiaSm-eaiah, nnrttgcC  maciiritvledastinideI  nn rms iehn tsigaBmuoetcetias rn"""
    #pada pesanSaya memiliki 3 tanda kutip diakhir dan diawal
    #hal tersebut digunakan untuk menandakan bahwa pesanSaya
    #baris-baris yang berbeda

    #proses menampilkan apakah berhasil atau gagal dalam melakukan peretasan
    pesanSaya = retasTransposisi(pesanSaya)

    #jika anda gagal
```

```
    if pesanYangTeretas == None:
        print('Gagal melakukan peretasan pada enkripsi.')
    else:
        print('Menyalin pesan yang teretas ke dalam layar:')
        print(pesanYangTeretas)
        #pyperclip harus berada satu folder dengan program
        pyperclip.copy(pesanYangTeretas)

def retasTransposisi(pesan):
    print('Sedang meretas...')

    #saat program python dalam kondisi berjalan
    #anda bisa melakukan pemberhentian setiap kali
    #anda menekan Ctrl-C pada Windows
    #atau Ctrl-D pada Mac dan Linux
    print('(Press Ctrl-C or Ctrl-D to quit at any time.)')

    #disini program melakukan bruteforce
    #dengan mengulangi setiap kunci yang memungkinkan
    for kunci in range(1, len(pesan)):
        print('Mencoba kunci #%s...' % (kunci))

        #decryptMessage berasal dari program dekripsitransposisi
        pesanTerdekripsi = dekripsitransposisi.dekripsiPesan(kunci, pesan)

        if deteksibahasa.cekBahasa(pesanTerdekripsi):
            #apabila kunci telah ditemukan
            print()
            print('Kemungkinan dapat di retas:')
            print('Kunci %s: %s' % (kunci, pesanTerdekripsi[:100]))
            print()
            print('Tekan D untuk berhenti dan Enter untuk melanjutkan peretasan:')
            respon = input('> ')

            if respon.strip().upper().startswith('D'):
                return pesanTerdekripsi

    return None
```

```
if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Melakukan Peretasan...
(Tekan Ctrl-C atau Ctrl-D setiap kali anda ingin berhenti)
Mencoba kunci #1...
Mencoba kunci #2...
Mencoba kunci #3...
Mencoba kunci #4...
Mencoba kunci #5...
Mencoba kunci #6...
Mencoba kunci #7...
Mencoba kunci #8...
Mencoba kunci #9...
Mencoba kunci #10...

enkripsi dapat diretas:
Kunci 10: Charles Babbage, FRS (26 December 1791 - 18 October 1871) was an
English mathematician, philosopher,

Tekan D untuk selesai, atau tekan Enter untuk melanjutkan:
```

ARITMATIKA MODULAR DENGAN MENGGUNAKAN PERKALIAN DAN SANDI AFFINE

Algoritma Euclid akan anda gunakan untuk mencari GCD (Greatest Common Divisor/Great Common Factor) dalam bahasa Indonesia adalah FPB (Faktor Persekutuan Terbesar).

Mencoba GCD/FPB dengan dua angka akan menjadi hal penting untuk melakukan perkalian dan juga dalam sandi Affine. Caranya cukup mudah hanya perlu mencari angka-angka dan tulis setiap faktor yang memungkinkan, lalu bandingkan daftar dan tentukan angka terbesar dari keduanya.

Seorang matematikawan yang hidup 2.000 tahun lalu bernama Euclid menjadi seorang algoritim dalam menemukan pembagian terbesar pada dua angka. Namun tak seorangpun yang mengerti atau dapat menjelaskan bahwa Euclid ada di dalam dokumen sejarah dan bagaimana rupanya secara pasti namun pada Universitas Oxford telah dibuat patung yang kiranya menyerupai beliau dan disebut sebagai “Statue of Some Guy with a Beard”.

Anda akan diberi contoh seperti hal mencari symbol F yang dienkrip dengan kunci ke-7, lalu menambahkan mod sehingga mendapatkan nilai (5 x 7)mod26 = 9 , (adalah angka untuk simbol J. Jadi F dapat dienkripsi sehingga menjadi J dan melakukan perkalian sandi dengan kunci 7.

Kali ini anda akan meramu dua teknik sandi menjadi satu yaitu perkalian sandi digubung dengan sandi Caesar menjadi sandi Affine. Salah satu kekurangan perkalian sandi adalah bahwa huruf A akan selalu memetakan huruf A. Hal ini karena huruf A adalah angka 0, dan jika 0 dikalikan dengan sesuatu akan menghasilkan angka 0 . Anda akan memperbaiki hal ini dengna menambahkan kunci kedua dengan sandi Caesar setelah melakukan perkalian sandi dan mod telah selesai.

Hal ini disebut dengan sandi Affine (affinity) karena memiliki dua kunci "kunci A" yang adalah integer (bilangan bulat) yang dengan huruf angka tersebut dikalikan. Setelah melakukan mod dengan angka 26. "Kunci B" adalah integer yang ditambahkan ke dalam angka. Hal ini dijumlahkan menjadi mod dengan 26. seperti halnya sandi Caesar.

Untuk menghitung kebalikan modular untuk mendapatkan kunci dekripsi, anda dapat menggunakan brute-force dengan menjelajahi dan mencoba memulai dari integer 1, 2, 3 seperti halnya diatas. Namun untuk kali ini anda akan menghemat waktu untuk kunci yang luasnya hingga 8,953,851.

```
############################################################
# Kriptomath                                               #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

def gcd(a, b):
    #menghasilkan GCD pada a dan b menggunakkan Algoritma Euclid
    while a != 0:
        a, b = b % a, a
    return b

def menentukanAturanKebalikan (a, m):
    #menghasilkan invers modular a % m,
    #dimana jumlah x sama seperti
    # a*x % m = 1

    if gcd(a, m) != 1:
        return None
    #tanpa pembagian mod inverse if a & m bukan bilangan prima

    #hitung menggunakan algoritma Euclidean
    u1, u2, u3 = 1, 0, a
    v1, v2, v3 = 0, 1, m
    while v3 != 0:
        q = u3 // v3 # adalah operator pembagian integer
```

```
        v1, v2, v3, u1, u2, u3 = (u1 - q * v1), (u2 - q * v2), (u3 - q * v3), v1, v2, v3
    return u1 % m

#jalankan shell lalu ikuti hal berikut
#import kriptomath
#kriptomath.gcd(1995, 27)
```

Karena perkalian sandi adalah hal yang sama dengan sandi Affine kecuali menggunakan kunci B dari 0. Anda tidak perlu membagi program untuk mengalikan program. Dan hal tersebut sangatlah tidak aman dalam hal sandi Affine dan sebenarnya tidak perlu selalu penggunaannya dengan cara tersebut.

SANDI AFFINE

Sandi Affine seperti yang dijelaskan sebelumnya adalah kelipatan atau perkalian dari sandi dimana sandi Caesar berada pada posisi atas. Sandi Affine membutuhkan dua kunci, satu untuk perkalian sandi dan kelipatannya dan lainnya untuk penambahan sandi Caesar.

Perihal program sandi Affine, anda akan menggunakan satu integer sebagai kunci. Dimana diperlukan penggunaan matematika sederhana untuk memisahkan kunci dari dua kunci, yang disebut dengan Kunci A dan Kunci B.

```
###################################################################
# Sandi Affine                                                    #
# Editor : Matius Celcius Sinaga                                  #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)   #
###################################################################

#modul sys diimport untuk keperluan fungsi exit()
#sudah tersedia pada saat menginstal Python
#modul pyperclip diimport untuk keperluan fungsi copy()
#modul kriptomath diimport untuk keperluan fungsi gcd() dan cariModMembalikkan
import sys, pyperclip, kriptomath, random
#memasukkan seluruh simbol yang mungkin akan digunakan nantinya
#termasuk diantaranya huruf besar dan kecil
#juga angka dan simbol, termasuk spasi berada di depan urutan
SYMBOLS = """ !"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[¥]^_`abcdefghijklmno
pqrstuvwxyz{|}~"""

#fungsi pada main() hampir sama dengan main() pada program transposisi
def main():
    pesanSaya = """"A computer would deserve to be called intelligent if it could
deceive a human into believing that it was human." -Alan Turing"""
    kunciSaya = 2023
    #pada modeSaya silahkan ubah apakah enkripsi maupun dekripsi
    modeSaya = 'enkripsi'
```

```
    #disini diatur agar memberikan nilai enkripsi pesanSaya
    if modeSaya == 'enkripsi':
        ubah = enkripsiPesan(kunciSaya, pesanSaya)
    #disini diatur apabila akan menghasilkan nilai dekripsi pesanSaya
    elif modeSaya == 'dekripsi':
        ubah = dekripsiPesan(kunciSaya, pesanSaya)
    print('Kunci: %s' % (kunciSaya))
    print('%si :' % (modeSaya.title()))
    print(ubah)
    #penggunaan pyperclip
    pyperclip.copy(ubah)
    print('Seluruh %s teks telah disalin.' % (modeSaya))

#kunci akan diubah menjadi kunciA dan kunciB
#contoh perhitungannya akan menjadi seperti
#kunciA 2023 // 95 atau 21
#kunciB 2023 % 95 atau 28
def cariBagianKunci(kunci):
    kunciA = kunci // len(SYMBOLS)
    kunciB = kunci % len(SYMBOLS)
    #hal ini akan memberikan hasil lebih cepat untuk kemudian digunakan berulang
    return (kunciA, kunciB)

#enkripsi dengan affine chiper melibatkan karakter dalam SYMBOLS menjadi berkalilipat
#dimana perkalian kunciA dan kunciB, kunciA = 1 dan kunciB = 0
#string akan melewati sys.exit(), fungsi ini akan menjadi pilihan fungsi parameter opsional
#yang akan ditampilkan pada layar sebelum program dijalankan
#hal ini juga berarti digunakan pada pesan error sebelum program berakhir
def cekKunci(kunciA, kunciB, mode):
    if kunciA == 1 and mode == 'enkripsi':
        sys.exit('Sandi Affine akan menjadi lebih lemah ketika A di tentukan menjadi 1. Silahkan pilih kunci yang berbeda.')
    if kunciB == 0 and mode == 'enkripsi':
        sys.exit('Sandi Affine akan menjadi lebih lemah ketika B di tentukan menjadi 0. Silahkan pilih kunci yang berbeda.')
    if kunciA < 0 or kunciB < 0 or kunciB > len(SYMBOLS) - 1:
```

```
        sys.exit('Kunci A harus lebih besar daripada 0 dan kunci B harus ada diantara 0 dan %s.' % (len(SYMBOLS) - 1))
    if kriptomath.gcd(kunciA, len(SYMBOLS)) != 1:
        sys.exit('Kunci A (%s) dan simbol ukuran yang ditentukan (%s) secara realtif bukanlah yang paling utama. Silahkan pilih kunci yang berbeda.' % (kunciA, len(SYMBOLS)))


def enkripsiPesan(kunci, pesan):
    kunciA, kunciB = cariBagianKunci(kunci)
    cekKunci(kunciA, kunciB, 'enkripsi')
    sanditeks = ''
    for symbol in pesan:
        if symbol in SYMBOLS:
            #untuk melakukan enkripsi anda harus menghitung jumlah kata yang akan dienkripsi
            #dengan melakukan perkalian pada symIndex dengan kunciA dan kunciB
            #lalu membagi nilai dengan ukuran simbol
            #jumlah yang anda hitung akan menjadi nilai SYMBOL karakter yang terenkripsi
            symIndex = SYMBOLS.find(symbol)
            sanditeks += SYMBOLS[(symIndex * kunciA + kunciB) % len(SYMBOLS)]
        else:
            #tambahkan apabila terdapat simbol yang tidak terenkripsi
            sanditeks += symbol
    return sanditeks


def dekripsiPesan(kunci, pesan):
    kunciA, kunciB = cariBagianKunci(kunci)
    cekKunci(kunciA, kunciB, 'dekripsi')
    teksawal = ''
    kunciUntukMembalikkanA = kriptomath.cariModMembalikkan(kunciA, len(SYMBOLS))

    for symbol in pesan:
        if symbol in SYMBOLS:
            #melakukan dekripsi pada SYMBOL dengan menggunakan kunci untuk membalikkan
```

```
            symIndex = SYMBOLS.find(symbol)
            teksawal += SYMBOLS[(symIndex - kunciB) * kunciUntukMembalikkanA % len(SYMBOLS)]
        else:
            #tambahkan apabila terdapat simbol yang tidak terdekripsi
            teksawal += symbol
    return teksawal


def menentukanKunciAcak():
    #disini akan dilakukan perulangan
    #jika perulangan  tidak pernah salah akan menjadi perulangan yang tiada akhir
    #jika program melakukan perulangan terus menerus, karena perulangan tidak menghasilkan nilai False
    #untuk berhenti maka tekan Ctrl-C atau Ctrl-D
    while True:
        kunciA = random.randint(2, len(SYMBOLS))
        kunciB = random.randint(2, len(SYMBOLS))
        if kriptomath.gcd(kunciA, len(SYMBOLS)) == 1:
            return kunciA * len(SYMBOLS) + kunciB


#apabila semua sudah berjalan termasuk modul yang diimport di dalamnya
#fungsi main akan mulai berjalan dari sini disini
if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Kunci: 2023
Enkripsi :
fX<*h>}(rTH<Rh()?<?T]TH=T<rh<tT<*_))T?<ISrT))I~TSr<Ii<Ir<*h()?<?T*TI=T<
_<4(>_S<ISrh<tT)IT=IS~<r4_r<Ir<R_]<4(>_SEf<0X)_S<k(HIS~
Seluruh enkripsi teks telah disalin.
```

7,125 adalah angka yang sama pada kunci yang paling memungkinkan dengan sandi pesan transposisi, dan anda sudah benar-benar belajar bagaimana program komputer untuk meretas angka dari kunci dengan brute-force. Yang berarti bahwasanya anda dapat memadukan sandi affine hingga mencapai titik kelemahan sandi yang memudahkan untuk di retas.

Sandi Affine tidak seaman sandi sebelumnya . Sandi Transposisi memiliki lebih banyak kunci, namun angka yang memungkinkan dibatasi oleh ukuran dari pesan. Untuk pesan dengan ukuran 20 karakter, sandi transposisi hanya dapat memiliki hampir 18 kunci (dari kunci 2 hingga 19). Sandi Affine juga dapat digunakan untuk mengenkripsi pesan singkat dengan lebih aman daripada sandi sebelumnya, sandi Caesar. Karena jumlah dari kunci yang memungkinkan di dasarkan pada jumlah set/pengaturan.

MERETAS SANDI AFFINE

Perlu anda ketahui bahwa sandi Affine dibatasi untuk beberapa ribuan kunci. Ini artinya bahwa brute-force belum tentu dapat memecah sandi ini.

```
# Meretas Sandi Affine                                                  #
# Editor : Matius Celcius Sinaga                                        #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)         #
#########################################################################

import pyperclip, sandiaffine, deteksibahasa, kriptomath

MODE_SENYAP = False

def main():
    pesanSaya = """U&'<3dJ^Gjx'-
3^MS'Sj0jxuj'G3'%j'<mMMjS'g{GjMMg9j{G'g"'gG'<3^MS'Sj<jguj'm'P^dm{'g{G3'%j
Mgjug{9'GPmG'gG'-m0'P^dm{LU'5&Mm{'_^xg{9"""

    pesanYangTeretas = retasAffine(pesanSaya)

    if pesanYangTeretas != None:
        #seluruh teks awal yang telah ditampilkan akan di salin ke dalam layar
        print('Sedang menyalin pesan yang sudah diretas:')
        print(pesanYangTeretas)
        pyperclip.copy(pesanYangTeretas)
    else:
        print('Gagal melakukan peretasan terhadap enkripsi.')


def retasAffine(pesan):
    print('Sedang mencoba meretas...')

    print('(Tekan Ctrl-C atau Ctrl-D untuk berhenti setiap kali.)')

    #mencoba brute-force dengan seluruh perulangan dan mencoba kunci yang
memungkinkan     for kunci in range(len(sandiaffine.SYMBOLS) ** 2):
        kunciA = sandiaffine.cariBagianKunci(kunci)[0]
        if kriptomath.gcd(kunciA, len(sandiaffine.SYMBOLS)) != 1:
```

```
            continue

        pesanTerdekripsi = sandiaffine.dekripsiPesan(kunci, pesan)
        if not MODE_SENYAP:
            print('Mencoba dengan kunci %s... (%s)' % (kunci, pesanTerdekripsi[:40]))

        if deteksibahasa.cekBahasa(pesanTerdekripsi):
            #apabila pengguna(user) melakukan pengecekan
            #terhadap kunci terdekripsi yang telah ditemukan
            print()
            print('Enkripsi dapat diretas:')
            print('Kunci: %s' % (kunci))
            print('Pesan Terdekripsi: ' + pesanTerdekripsi[:200])
            print()
            print('Tekan D untuk selesai, atau tekan Enter untuk melanjutkan peretasan:')
            respon = input('> ')

            if respon.strip().upper().startswith('D'):
                return pesanTerdekripsi
    return None


#jika program berjalan dengan sesuai juga dengan modul yang diimnport
#fungsi main dimulai dari sini
if __name__ == '__main__':
    main()
```

Outputnya yang dihasilkan adalah :

```
Sedang mencoba meretas...
(Tekan Ctrl-C atau Ctrl-D untuk berhenti setiap kali.)
Mencoba dengan kunci 95... (U&'<3dJ^Gjx'-3^MS'Sj0jxuj'G3'%j'<mMMjS'g)
Mencoba dengan kunci 2190... (?^=!-+.32#0=5-3*"="#1#04#=2-= #=!~**#"=')
Mencoba dengan kunci 2191... (` ^BNLOTSDQ^VNTKC^CDRDQUD^SN^AD^B@KKDC^H)
Mencoba dengan kunci 2192... ("A computer would deserve to be called i)

Enkripsi dapat diretas:
Kunci: 2192
Pesan Terdekripsi: "A computer would deserve to be called intelligent if it could deceive a human into believing that it was human." -Alan Turing

Tekan D untuk selesai, atau tekan Enter untuk melanjutkan peretasan:
```

Pada bagian ini tidak dijelaskan suatu teknik baru. Selama jumlah kemungkinan kunci yang memungkinkan lebih kurang sejuta, hal ini akan memberikan waktu lebih lama untuk melakukan brute-force terhadap setiap kunci yang memungkinkan dengan menggunakan cekBahasa() untuk melakukan pengecekan kunci yang benar.

SANDI SUBSTITUSI SEDERHANA

Transposisi dan sandi Affine memiliki ribuan kemungkinan kunci, namun sebuah komputer juga dapat melakukan brute-force untuk mengatasi semuanya dengan mudah. Anda nantinya akan membutuhkan sebuah sandi yang memiliki kemungkinan kunci yang lebih banyak, dimana sebuah komputer tak lagi mampu melakukan brute-force terhadap sandi tersebut.

Sandi Sibtitusi sederhana secara efektif tak dapat dikalahkan oleh brute-force. Meskipun komputer anda dapat menjalankan milyaran kunci setiap detik, hal ini akan tetap membutuhkan jutaan tahun untuk dapat mencoba semua kunci yang sudah ada dan dihasilkan.

Untuk mengimplementasikan bagaimana sandi Subtitusi bekerja, pilihlah sebuah sususan kata yang terdiri dari huruf acak untuk melakukan enkripsi terhadap huruf tersebut. Untuk satu huruf digunakan hanya untuk sekali saja. Kunci tersebut akan menghasilkan 26 huruf dari setiap bilangan acak yang ditukarkan. Sehingga menjadi terdapat 403,291,461,126,605,635,584,000,000 kemungkinan kunci yang akan digunakan.

```
############################################################
# Sub Sandi Sederhana                                      #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

import pyperclip, sys, random

HURUF = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def main():
    pesanSaya = 'If a man is offered a fact which goes against his instincts, he will scrutinize it closely, and unless the evidence is overwhelming, he will refuse to believe it. If, on the other hand, he is offered something which affords a reason for acting in
```

```
accordance to his instincts, he will accept it even on the slightest evidence. The origin of myths is explained in this way. -Bertrand Russell'
    kunciSaya = 'LFWOAYUISVKMNXPBDCRJTQEGHZ'
    #program ini akan melakukan enkripsi
    modeSaya = 'enkripsi'

    cekKebenaranKunci(kunciSaya)

    if modeSaya == 'enkripsi':
        ubah = pesanEnkripsi(kunciSaya, pesanSaya)
    elif modeSaya == 'dekripsi':
        ubah = pesanDekripsi(kunciSaya, pesanSaya)
    print('Menggunakan kunci %s' % (kunciSaya))
    print('Kunci %sed pesan:' % (modeSaya))
    print(ubah)
    pyperclip.copy(ubah)
    print()
    print('Pesan sedang disalin kedalam layar.')

#fungsi cekKebenaranKunci melakukan validasi/pemeriksaan kunci
#apakah kunci sudah digunakan, lalu mengurutkannya
#apabila sudah digunakan akan memberikan pesan kesalahan
def cekKebenaranKunci(kunci):
    daftarKunci = list(kunci)
    daftarHuruf = list(HURUF)
    #pada sort() seluruh hasil modifikasi ditempatkan dan tidak memberi nilai pengembalian
            #untuk mengatur daftarKunci dan daftarHuruf secara bersamaan
            #karena daftarKunci adalah karakter yang berasal dari HURUF
        #jika daftarkunci dan daftarHuruf adalah sama, anda dapat mengetahui bahwa daftarKunci
             #tidak memiliki salinan di dalamnya, karena HURUF tidak memiliki salinan di dalamnya
    daftarKunci.sort()
    daftarHuruf.sort()

#setiap nilai dari karakter di dalam simbol dicek
#tanpa melakukan duplikasi tanpa adanya kehilangan huruf yang digunakan
#pada daftarKunci nilai variabel pada daftar kunci dikembalikan
#di mulai dari huruf A - Z, simbol dan angka yang digunakan
```

```
    if daftarKunci != daftarHuruf:
        sys.exit('Ditemukan sebuah error pada kunci atau simbol.')

#salah satu cara untuk memasukkan fungsi dan memanggilnya dua kali tanpa harus membuat lagi
#pertama, hal ini tidak perlu di kode program lagi
#kedua, jika ada bug/kesalahan/celah dalam kode yang diduplikasi
#anda hanya perlu memperbaiki bug tersebut sekali pada dua tempat berbeda
#hal ini lebih baik daripada mengganti kode yang sudah diduplikat dengan satu fungsi yang memiliki kode
def pesanEnkripsi(kunci, pesan):
    return ubahPesan(kunci, pesan, 'enkripsi')

def pesanDekripsi(kunci, pesan):
    return ubahPesan(kunci, pesan, 'dekripsi')

#fungsi ubahPesan melakukan enkripsi/dekripsi tergantung mode yang ditetapkan
#proses enkripsinya sangat mudah, pada setiap kata dalam parameter pesan
#lalu menggantinya dengan parameter kunci dan begitu sebaliknya(untuk dekripsi)
def ubahPesan(kunci, pesan, mode):
    ubah = ''
    karakterA = HURUF
    karakterB = kunci
    if mode == 'dekripsi':

        #untuk dekripsi, anda dapat menggunakan kode enkripsi yang sama
        #anda hanya perlu melakukan persamaan dimana kunci dan HURUF digunakan
        karakterA, karakterB = karakterB, karakterA

    #lompati seluruh SYMBOL dalam pesan
    for symbol in pesan:
        if symbol.upper() in karakterA:

            #proses encrypt/decrypt the symbol
            #string isupper() akan mengembalikan nilai benar apabila :
            #string memiliki setidaknya satu huruf kapital
            #string tidak memiliki satu huruf kecil
            #string islower() akan mengembalikan nilai benar apabila :
            #string memiliki setidaknya satu huruf kecil
```

```
            #string tidak memiliki satu huruf kapital
            symIndex = karakterA.find(symbol.upper())
            if symbol.isupper():
                ubah += karakterB[symIndex].upper()
            else:
                ubah += karakterB[symIndex].lower()
        else:
            #symbol bukan bagian dari HURUF, hanya tambahan saja
            ubah += symbol
    return ubah


def menentukanKunciAcak():
    kunci = list(HURUF)
    random.shuffle(kunci)
    return ''.join(kunci)


if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Menggunakan kunci LFWOAYUISVKMNXPBDCRJTQEGHZ
Kunci enkripsied pesan:
Sy l nlx sr pyyacao l ylwj eiswi upar lulsxrj isr sxrjsxwjr, ia esmm rwctjsxsza sj wmpramh, lxo txmarr jia aqsoaxwa sr pqaceiamnsxu, ia esmm caytra jp famsaqa sj. Sy, px jia pjiac ilxo, ia sr pyyacao rpnajisxu eiswi lyypcor l calrpx ypc lwjsxu sx lwwpcolxwa jp isr sxrjsxwjr, ia esmm lwwabj sj aqax px jia rmsuijarj aqsoaxwa. Jia pcsusx py nhjir sr agbmlsxao sx jisr elh. -Facjclxo Ctrramm

Pesan sedang disalin kedalam layar.
```

Apabila huruf yang digunakan dalam pesan asli adalah huruf kecil, hal ini akan menjadi huruf kecil pada santi teks. Sandi Subtitusi sederhana tidak akan mengenkripsi program yang dapat mengenkripsi seluruhnya termasuk spasi dan tanda khusus.

MELAKUKAN PERETASAN PADA SANDI SUBSTITUSI SEDERHANA

Berapa banyak kata yang anda ketahui untuk menirukan pembentukan kalimat yang sama antar yang satu dan yang lainnya. Contohnya adalah “Puppy” hanya memiliki 3 jenis huruf berbeda saja, P, U dan Y hal ini hampir sama dengan Mommy dan Bobby bahkan mungkin Lilly hanya diikuti oleh 3 huruf berbeda saja. Banyak contoh kata yang berbeda yang pembentukannya hampir sama dan tak jarang anda temui bahwa dengan mengganti satu huruf dalam kata akan memiliki pengartian yang berbeda pula.

Dalam teks awal dan katasandi akan selalu menghasilkan pola kata yang sama pula, tidak masalah contohnya seperti apa pada kunci Subtitusi yang digunakan dalam melakukan enkripsi.

```
############################################################
# Membuat Pola Kata                                        #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

#modul pprint memiliki fungsi dalam pretty printing nilai
#dimana hal ini berguna untuk menampilkan kamus dan daftar nilai pada layar
#pprint sudah tersedia saat anda melakukan instalasi Python
#anda hanya perlu memanggil/import saja
import pprint

def membuatPolaKata(kata):
    #menghasilkan sebuah string dari pola yang diberikan oleh kata
    #contohnya adalah '0.1.2.3.4.1.2.3.5.6' dan selanjutnya
    kata = kata.upper()
    angkaSelanjutnya = 0
    nomorHuruf = {}
    polaKata = []

    for huruf in kata:
        if huruf not in nomorHuruf:
            nomorHuruf[huruf] = str(angkaSelanjutnya)
```

```
            angkaSelanjutnya += 1
        polaKata.append(nomorHuruf[huruf])
    return '.'.join(polaKata)

def main():
    seluruhPola = {}

    fo = open('dictionary.txt')
    daftarKata = fo.read().split('/n')
    fo.close()

    for kata in daftarKata:
        #tentukan pola untuk setiap string didalam daftarKata
        pola = membuatPolaKata(kata)

        if pola not in seluruhPola:
            seluruhPola[pola] = [kata]
        else:
            seluruhPola[pola].append(kata)

    #kode program ini akan menghasilkan kode program polaKata.py
    #dan program polaKata.py sangat besar dan memiliki statement yang luas
    fo = open('polaKata.py', 'w')
    fo.write('seluruhPola = ')
    fo.write(pprint.pformat(seluruhPola))
    fo.close()

if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah kumpulan kata yang dimana hal tersebut merupakan program baru polaKata.py

TEORI MELAKUKAN PERETASAN SANDI SUBSTITUSI SEDERHANA

Cara untuk menjalankan sandi Subtitusi sederhana cukuplah mudah, yaitu :

1. Cari kata pembentuk dari setiap katasandi dalam sanditeks
2. Cari daftar kata berbahasa inggris mewakili dari setiap katasandi yang memungkinkan pendekripsian
3. Buat satu pemetaan huruf sandi untuk setiap kata sandi menggunakan daftar kata sandi yang dicalonkan. (pemetaan huruf sandi hanyalah kamus yang benar).
4. Potong setiap pemetaan huruf sandi menjadi satu permotongan pemetaan huruf sandi
5. Pindahkan setiap huruf yang sudah selesai pada pemetaan intersect/pemotongan.

```
############################################################
# Meretas Sub Sandi Sederhana                              #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

#langkah-langkah eksekusi program
#1.Temukan pola/aturan kata pada setiap pesan kata dalam pesan teks
#2.Temukan daftar kandidat bahasa pada pesan kata yang akan di dekrip
#3.Buatlah pemetaan huruf pada setiap pesan kata dengan menggunakan daftar kandidat pesan kata
#4.Eliminasi setiap pemetaan huruf
#5.Pindahkan setiap huruf yang telah dieliminasi pada pemetaan
import os, re, copy, pprint, pyperclip, subsandisederhana, membuatpolakata

if not os.path.exists('polaKata.py'):
    membuatpolakata.main()
import polaKata

HURUF = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
bukanHurufatauPolaSpasi = re.compile('[^A-Z\s]')

def main():
    pesan = 'Sy l nlx sr pyyacao l ylwj eiswi upar lulsxrj isr sxrjsxwjr, ia esmm rwctjsxsza sj wmpramh, lxo txmarr jia aqsoaxwa sr pqaceiamnsxu, ia esmm caytra jp famsaqa sj. Sy, px jia pjiac ilxo, ia sr pyyacao rpnajisxu eiswi lyypcor l calrpx ypc lwjsxu sx
```

```
lwwpcolxwa jp isr sxrjsxwjr, ia esmm lwwabj sj aqax px jia rmsuijarj aqsoaxwa. Jia pcsusx py nhjir sr agbmlsxao sx jisr elh. -Facjclxo Ctrramm'

    #mencoba melakukan validasi pada ciphertext
    print('sedang meretas...')
    pemetaanHuruf = retasSubSederhana(pesan)

    #munculkan hasil yang sudah didapatkan
    print('Melakukan pemetaan:')
    pprint.pprint(pemetaanHuruf)
    print()
    print('Pesan sandi yang Asli:')
    print(pesan)
    print()
    print('Menyalin pesan yang sudah diretas ke dalam layar:')
    pesanYangTeretas = dekripsiDenganMenggunakanHurufSandiYangDipetakan(pesan, pemetaanHuruf)
    pyperclip.copy(pesanYangTeretas)
    print(pesanYangTeretas)

def menentukanPemetaanRuangKosongHurufSandi():
    #mengembalikan nilai dictionary dalam bentuk huruf-huruf asli yang dipetakan
    return {'A': [], 'B': [], 'C': [], 'D': [], 'E': [], 'F': [], 'G': [], 'H': [], 'I': [], 'J': [], 'K': [], 'L': [], 'M': [], 'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [], 'U': [], 'V': [], 'W': [], 'X': [], 'Y': [], 'Z': []}




def tambahkanHurufpadaPemetaan(pemetaanHuruf, katasandi, sampel):
    #pada letterMapping parameternya adalah pesan asli dimana nilai yang dikembalikan oleh fungsi ini dimulai dengan menyalinnya
    #pada cipherword parameternya adalalah nilai string pada ciphertext word
    #parameter candidate memungkinkan kata dalam bahasa inggris pada cipherword dapat di dekrip juga

    #fungsi ini menambahkan kata pada candidate sebagai dekripsi potensial pada cipherletters dalam pemetaan sandi huruf
    pemetaanHuruf = copy.deepcopy(pemetaanHuruf)
    for i in range(len(katasandi)):
```

```
        if sampel[i] not in pemetaanHuruf[katasandi[i]]:
            pemetaanHuruf[katasandi[i]].append(sampel[i])
    return pemetaanHuruf


def memotongPemetaan(petaA, petaB):
    #untuk menggabungkan dua peta, buatlah peta kosong, lalu tambahkan huruf yang memungkinkan untuk didekripsi jika ada maka lakukan pada kedua peta
    PemetaanYangDipotong = menentukanPemetaanRuangKosongHurufSandi()
    for huruf in HURUF:

        #daftar kosong berarti "setiap huruf memungkinkan". Pada hal ini hanya salin map lainnya pada masukan
        if petaA[huruf] == []:
            PemetaanYangDipotong[huruf] = copy.deepcopy(petaB[huruf])
        elif petaB[huruf] == []:
            PemetaanYangDipotong[huruf] = copy.deepcopy(petaA[huruf])
        else:
            #jika huruf dalam mapA[letter] ada di dalam mapB[letter] tambahkan huruf pada pemotongan pemetaan[huruf]
            for hurufYangDipetakan in petaA[huruf]:
                if hurufYangDipetakan in petaB[huruf]:
                    PemetaanYangDipotong[huruf].append(hurufYangDipetakan)

    return PemetaanYangDipotong


def mengubahHurufyangsudahselesaiDipetakan(pemetaanHuruf):
    #huruf asli dalam pemetaan dalam map hanya untuk per satu huruf saja yang didekrip dan dapat di ubah dalam bentuk huruf lainnya
    #sebagai contoh, jika 'A' dipetakan pada huruf yang potensial ['M','N'], dan 'B' dipetakan pada ['N'], lalu anda tahu bahwa 'B' harus dipetakan pada 'N',
    #lalu anda dapat mengubah 'N' dari daftar bahwa 'A' dapat dipetakan juga. Maka 'A' dipetakan pada ['M'].
    #sekarang peta 'A' hanya memiliki satu huruf, anda dapat mengubah nya ke 'M' dari daftar kata untuk setiap huruf. (hal inilah mengapa ada perulangan dan pemakaian kembali map
    pemetaanHuruf = copy.deepcopy(pemetaanHuruf)
    ulangiLagi = True
```

```
    while ulangiLagi:
        #asumsikan dahulu bahwa hal ini tidak melakukan perulangan lagi
        ulangiLagi = False

        #huruf yang sudah diubah hanya akan menjadi daftar huruf besar saja dan hanya memmungkinkan satu pemetaan dalam pemetaan huruf
        hurufYangDipecah = []
        for sandiHuruf in HURUF:
            if len(pemetaanHuruf[sandiHuruf]) == 1:
                hurufYangDipecah.append(pemetaanHuruf[sandiHuruf][0])

    #jika satu huruf telah selesai, lalu hal ini tidak berarti dapat menjadi huruf potensial untuk melakukan dekripsi pada chipertext yang berbeda
    #jadi anda harus mengubah hal ini dari daftar yang lainnya
    for sandiHuruf in HURUF:
            for s in hurufYangDipecah:
                if len(pemetaanHuruf[sandiHuruf]) != 1 and s in pemetaanHuruf[sandiHuruf]:
                    pemetaanHuruf[sandiHuruf].remove(s)
                    if len(pemetaanHuruf[sandiHuruf]) == 1:
                        #jika sudah ada huruf yang selesai diproses maka,
                        #diulangi untuk huruf selanjutnya dan seterusnya
                        loopAgain = True
    return pemetaanHuruf


def retasSubSederhana(pesan):
    PetaYangDipotong = menentukanPemetaanRuangKosongHurufSandi()
    daftarKataSandi = bukanHurufatauPolaSpasi.sub('', pesan.upper()).split()
    for kataSandi in daftarKataSandi:
        #membuat pemetaan cipherletter yang baru untuk setiap kata sandi teks
        petaBaru = menentukanPemetaanRuangKosongHurufSandi()

        #polaKata dan polaKata1 adalah dua hal berbeda
        polaKata1 = membuatrumuskata.membuatPolaKata(kataSandi)
        if polaKata1 not in polaKata.seluruhPola:
            continue #jika hal yang dicari tidak ada dalam kamus

         #tambahkan huruf untuk setiap pemetaan candidate
```

```
        for sampel in polaKata.seluruhPola[polaKata1]:
            petaBaru = tambahkanHurufpadaPemetaan(petaBaru, kataSandi, sampel)

        #peta yang dipotong adalah peta yang telah dipotong yang sebelumnya telah ada
        PetaYangDipotong = memotongPemetaan(PetaYangDipotong, petaBaru)

    #pindahkan setiap huruf yang sudah selesai ke dalam daftar lainnya
    return mengubahHurufyangsudahselesaiDipetakan(PetaYangDipotong)


def dekripsiDenganMenggunakanHurufSandiYangDipetakan(sanditeks, pemetaanHuruf):
    #mengembalikan nilai string pada ciphertext yang sudah didekrip dengan pemetaan huruf
    #setiap huruf hasil dekrip yang ambigu akan ditempatkan kembali dengan sebuah "_" underscore


    #buatlah dahulu sebuah kunci sub sederhana dari letterMapping dengan pemetaan
    kunci = ['x'] * len(HURUF)
    for sandiHuruf in HURUF:
        if len(pemetaanHuruf[sandiHuruf]) == 1:
            #jika hanya ada satu huruf saja, lalu tambahkan hal ini ke dalam kunci
            indeksKunci = HURUF.find(pemetaanHuruf[sandiHuruf][0])
            kunci[indeksKunci] = sandiHuruf
        else:
            sanditeks = sanditeks.replace(sandiHuruf.lower(), '_')
            sanditeks = sanditeks.replace(sandiHuruf.upper(), '_')
    kunci = ''.join(kunci)

    #dengan kunci yang telah anda buat, lakukan dekripsi pesan asli
    #pesanDekripsi harus sesuai dengan subsandisederhana
    return subsandisederhana.pesanDekripsi(kunci, sanditeks)


if __name__ == '__main__':
    main()
#program ini bekerja hanya jika spasi tidak di enkripsi
#memiliki 403,291,461,126,605,635,584,000,000 kemungkinan
```

```
#kemungkinan melakukan bruteforce hampir beberapa ratus tahun dan hampir tidak terpecahkan
#hal ini disebut dengan "polyalphabetic" oleh sandi Vigenère
#saat program sudah berjalan akan membutuhkan beberapa waktu untuk komputer berjalan normal
#komputer anda baik-baik saja dan begitupun dengan program ini
#hanya saja membutuhkan waktu untuk tidak sekedar mengkompilasi
#namun juga menghasilkan program baru bernama polaKata.py
#data polaKata.py berasal dari dictionary.txt
```

Kata sandi lebih banyak berada pada sanditeks, lebih banyak pemetaan huruf sandi yang anda miliki berarti dapat menjadikan intersect/pemotongan. Lebih banyak pemetaan huruf sandi yang anda intersectkan bersama, berarti adanya potensi dekripsi yang lebih banyak melakukan dekripsi huruf yang ada pada huruf sandi. Ini berarti lebih besar kemungkinan pesan sandi teks, lebih banyak kemungkinan anda mampu melakukan hack dan mendekripnya.

DAPATKAH ANDA MELAKUKAN ENKRIPSI TERHADAP RUANG KOSONG ?

Ya, pada teknik peretasan sebelumnya anda hanya melakukan peretasan terhadap kata atau huruf saja namun tidak dengan ruang kosong (spasi) dan simbol khusus. Anda dapat mengubah sandi substitusi sederhana dari bagian sebelumnya untuk mengenkripsi spasi, angka, dan karkater khusus sebagaimana sebuah kata, dan akan membuat pesan terenkripsi dengan kuat namun tetap bukan tidak mungkin untuk di retas. Bagaimanapun, semenjak spasi menjadi kata/huruf bagian dari sanditeks, anda dapat menulis sebuah program untuk meretasnya dan mengubahnya menjadi spasi, layaknya meretas sanditeks dengan normal. Jadi meskipun anda melakukan enkripsi terhadap karakter spasi namun tetap dapat diretas dan tidak terlalu menawarkan banyak perlindungan terhadap enkripsi tersebut.

Meretas program pada akhirnya memanglah sangat rumit. Pemetaan sanditeks sebagai alat utama untuk memodelkan huruf hingga memungkinkan huruf sanditeks dapat didekripsi . Dengan menambahkan huruf (berdasarkan sampel dari setiap katasandi) pada pemetaannya, lalu pemetaan dengan pemotongan/intersect untuk mengubah sandi yang sudah diubah berdasarkan daftar dari dekripsi pesan, Anda dapat mengubahnya menjadi lebih sederhana pada kunci-kunci yang mungkin. Sebagai perbandingannya anda mencoba sebanyak 403,291,461,126,605,635,584,000,000 kemungkinan kunci dimana anda bisa menggunakan beberapa kecanggihan kode python untuk menggambarkan bagaimana hal tersebut terjadi, namun tidak menghasilkan seluruh pesan asli kunci substitusi tersebut.

Kekuatan yang paling sederhana dari substitusi sederhana adalah luasnya jumlah kemungkinan kunci. Namun jika hanya dengan membongkar dan merusaknya saja tidaklah cukup dibandingkan dengan jumlah katasandi pada pada file dictionary untuk

menebak huruf sandi dengan cara yang sederhana lalu memilih hal manakah yang dapat mendekripsi huruf.

SANDI VIGENÈRE

Vigenère cipher adalah juga sandi yang disebut sebagai "polyalphabetic", dimana hal ini dimaksudkan bahwa sandi ini membutuhkan ratusan tahun untuk meretasnya. Sandi Vigenère memiliki sandi yang kuat bahkan lebih kuat dari sandi-sandi yang sebelumnya yang memiliki kemungkinan kunci yang terlalu banyak hingga anda dapat melakukan brute-force, bahkan hanya dengan menggunakan pendeteksian bahasa. Namun berbeda dengan sandi Vigenère yang tidak dapat dirusak/diretas hanya dengan pola kata yang bekerja pada sandi Subtitusi sederhana. Hal ini dijelaskan pertama sekali oleh salah seorang Kriptografer bernama Giovan Battista Bellaso lalu kemudian setelah sekian lama dikembangkan oleh Blaise de Vigenère. Sandi ini tidak dapat dirusak/diretas hingga akhirnya seorang ahli komputer melakukannya pada abad ke-19 yaitu Charles Babbage. Dan sandi ini disebut "le chiffreindéchiffrable" dalam bahasa Francis berarti "sandi yang tidak terbaca".

Sandi Vigenère hampir sama dengan sandi Caesar, perbedaannya adalah pada Sandi Vigenère menggunakan kunci perkalian. Karena menggunakan lebih dari satu set substitusi, maka hal ini disebut juga dengan sandi polyalphabetic substitution. Pada sandi Caesar terdapat kunci dari 0 hingga 25. Pada sandi Vigenère menggunakan sandi angka nomerik, kemungkinan anda akan menggunakan kunci huruf. Huruf A akan digunakan pada kunci 0. Huruf B akan menggunakan kunci 1, dan begitulah hingga mencapai Z pada kunci ke 25.

Sandi Vigenère tidak memiliki celah pada dictionari untuk pola kata menggunakan program sama halnya dengan meretas substitusi. Polyalphabetic cipher menyimpan pesan rahasia selama ratusan tahun. Bayangkan saja bagaimana peretasan terhadap sandi Vigenère dahulunya sangat susah hingga abad 19 yang dimana sandi ini akhirnya dapat diretas, anda akan belajar salah satu teknik baru yaitu "menganalisis frekuensi" yaitu teknik yang digunakan untuk meretas sandi Vigenère.

```
############################################################
# Vigenère Cipher ( Sandi Subtitusi Polyalphabetic )       #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

import pyperclip

HURUF = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

#pada fungsi main terdapat variabel pesan, kunci dan mode sebelum program dijalankan
#pada mode anda dapat menetapkan apakah program akan melakukan enkripsi atau dekripsi
def main():
    pesanSaya = """Alan Mathison Turing was a British mathematician, logician, cryptanalyst, and computer scientist. He was highly influential in the development of computer science, providing a formalisation of the concepts of "algorithm" and "computation" with the Turing machine. Turing is widely considered to be the father of computer science and artificial intelligence. During World War II, Turing worked for the Government Code and Cypher School (GCCS) at Bletchley Park, Britain's codebreaking centre. For a time he was head of Hut 8, the section responsible for German naval cryptanalysis. He devised a number of techniques for breaking German ciphers, including the method of the bombe, an electromechanical machine that could find settings for the Enigma machine. After the war he worked at the National Physical Laboratory, where he created one of the first designs for a stored-program computer, the ACE. In 1948 Turing joined Max Newman's Computing Laboratory at Manchester University, where he assisted in the development of the Manchester computers and became interested in mathematical biology. He wrote a paper on the chemical basis of morphogenesis, and predicted oscillating chemical reactions such as the Belousov-Zhabotinsky reaction, which were first observed in the 1960s. Turing's homosexuality resulted in a criminal prosecution in 1952, when homosexual acts were still illegal in the United Kingdom. He accepted treatment with female hormones (chemical castration) as an alternative to prison. Turing died in 1954, just over two weeks before his 42nd birthday, from cyanide poisoning. An inquest determined that his death was suicide; his mother and some others believed his death was accidental. On 10 September 2009, following an Internet campaign, British Prime Minister Gordon Brown made an official public apology on behalf of the British government for "the appalling way he was treated." As of May 2012 a private member's bill was before the House of Lords which would grant Turing a statutory pardon if enacted."""
```

```
    kunciSaya = 'MATIUS'
    #silahkan ubah untuk melakukan enkripsi atau dekripsi disini
    modeSaya = 'enkripsi'

    if modeSaya == 'enkripsi':
        ubah = enkripsiPesan(kunciSaya, pesanSaya)
    elif modeSaya == 'dekripsi':
        ubah = dekripsiPesan(kunciSaya, pesanSaya)

    print('%sed pesan : ' % (modeSaya.title()))
    print(ubah)
    pyperclip.copy(ubah)
    print()
    print('Pesan akan disalin kedalam papan klip.')

#pada bagian ini menjelaskan bagaimana enkripsi berjalan
def enkripsiPesan(kunci, pesan):
    return ubahPesan(kunci, pesan, 'enkripsi')

#pada bagian ini menjelaskan bagaimana dekripsi berjalan
def dekripsiPesan(kunci, pesan):
    return ubahPesan(kunci, pesan, 'dekripsi')

def ubahPesan(kunci, pesan, mode):
    ubah = []
#menyimpan pesan enkripsi dan dekripsi

    kunciIndex = 0
    kunci = kunci.upper()

    for symbol in pesan:
    #akan dilakukan pada seluruh karakter dalam pesan
        nomor = HURUF.find(symbol.upper())
        if nomor != -1: #-1 berarti symbol.upper() tidak ditemukan didalam HURUF
            if mode == 'enkripsi':
                nomor += HURUF.find(kunci[kunciIndex]) #tambahkan jika dienkripsi
            elif mode == 'dekripsi':
                nomor -= HURUF.find(kunci[kunciIndex]) #kurangi jika melakukan
dekripsi
```

```
            nomor %= len(HURUF)

            #tambahkan pada hasil symbol enkrip/dekrip yang sudah diubahkan
            if symbol.isupper():
                ubah.append(HURUF[nomor])
            elif symbol.islower():
                ubah.append(HURUF[nomor].lower())

            kunciIndex += 1
            #ubah kunci yang akan dipakai selanjutnya
            if kunciIndex == len(kunci):
                kunciIndex = 0

        else:
            #symbol tidak berada pada HURUF, maka tambahkan hal tersebut dan
ubahkan
            ubah.append(symbol)

    return ''.join(ubah)

#jika sandiVigenere.py sudah berjalan termasuk seluruh modulnya
#panggil fungsi main
if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Enkripsied pesan :
Mltv Gsfhbaif Fukqhy ial i Vjutbab emtamgsfivquf, xozqwamn, vzshfagifqet, tvx uamicnwd svqyffilb. Bw ial
pcytlr qhxxuxvnaml bv nzq dxdydapfmhl af vwghgtxz muuegky, hdooqxazg t nijyaeqmsfihv ix fhx kifoeibm gr
"aeoijutau" ufp "chujmfamqif" iimp nzq Tnzcfs mtkbaze. Mclazg ba qapeeg wgzsblyjqd mw vw fhx nultek wz
uamicnwd svqyfoe tvx sdtbncuuae qhlqleqawzcx. Lojunz Eijxd Pil AU, Tnzcfs whzewp fhz nzq Ghdyjzmxvn
Uadx ihv Oyipyj Ecawid (SCVA) ul Nlxbwzxer Xujw, Bkqnsun'l kivqbkmucunz kyffrx. Nij m tbuy zq wta
bwmd hn Bmf 8, tam mwotbwh jqsiwhkubem zgd Gxzgsz ntdud orrxnszaegmae. Hx lynusxl u fgmuml gr
txkbfuqnmm xar uzyswigo Awdmtv wabhxzm, azcecxazg mpy eqtawx gr tam vgybx, ih wxevblgyevpufuctt
gsohbvy ltam kimxd yqhv eembcfss ywl lte Xvcyya fiwzunx. Izlqr mpy omr am qgdkxl ul fhx Vuluogif
Htylqwsx Ltjijmthzs, otekm bw orxinwp ogm ix fhx ncjet wmmasnl nij m smwlwp-pkwajmm vwghgtxz, nzq
AVM. Cf 1948 Fukqhy vobvyv Yaq Vyoyag'a Wgypnbcfs Ltjijmthzs sf Mtvwzqsmml Mziomlkutr, ebwde am
ukeilbyv un mpy vqvxtihyegb ix fhx Uufohxanwd chujmfeka ufp bxkueq igbyjqsmmx az mtbbwyamqwsx
bbwfgsy. Am qjatx i jsbek wh lte vpyeuctt vseil wz earipiyqnxack, mnw xlwpivbyv asvqfdmtbva utefqwsx
```

rxiwluoga mmoh ta nzq Bxtimeoo-Hbsnomqhkwy kmuufihv, qzuca eyjq fbzml ablmlnqd bv nzq 1960s. Mclazg'l pieasxfosximg lweuebyv un t klayigif hdolmwmfihv cf 1952, ihxv bgyolmrmml tknk iekm mlule qfdqgtt cf fhx Chafew Scfsdhu. Bw mcvmjlqd mzysfmxvn outa nyemlx pijyogmm (utefqwsx ctanjmtbwh) se ag iflqrginahe mw jjushv. Nmdigo xaqd bv 1954, dmet hdyj fwh eywws umzgde aqm 42fp bbznzpar, nlgy crihape iwckanbva. Sz igyowet wmnwdmbvyv fhtb bae dxinz ial aoaoiwm; bae mhbbwd agl mgye hbbwds umfaqvxl bae dxinz ial iwuudxvnsx. Og 10 Ayhfefjyj 2009, roetiounz ih Aztxzhwf ctujsugg, Jlafilp Jjumx Ucfusmml Yarwwh Tdopv gspe tv ixrivqud buutcu mphtiyk og jyzmly wz lte Uzclusa oinqrguyff fhz "nzq aixudxigo qsk hx euk frxinwp." Al wz Emy 2012 t xlahamm gwybxz'm tule euk neywlw fhx Pimee hn Fgddl ebaoh pwodp gkihl Fukqhy m sminmfokg jsddhv cx qntknwp..

Pesan akan disalin kedalam papan klip.

ANALISIS FREKUENSI

Pada koin anda dapat melihat 2 bagian berbeda, saat anda membalikkan sebuah koin, anda hanya melihat kepala (atas) dan ekor (bawah). Frekuensi juga adalah banyaknya jumlah kemungkinan yang diberikan apabila 2 mata koin menghasilkan kepala atau ekor. Yaitu masing-masing 50% untuk setiap bagiannya.

Ada 26 huruf dalam alphabet bahasa inggris dan Indonesia, namun setiap huruf tidak memiliki bentuk yang sama dalam teks bahasa Inggris maupun Indonesia. Beberapa huruf digunakan lebih sering dibandingkan huruf lainnya. Sebagai contoh anda akan menggunakan bahasa Inggris pada program kali ini agar sesuai dengan dictionary yang saya miliki, jika anda memiliki dictionary atau kamus untuk tata bahasa Indonesia silahkan ubah dan tetap perhatikan letak perbedaannya. Baiklah coba anda lihat tulisan dalam buku-buku maupun teks berbahasa inggris akan menghasilkan E, T, A dan O dengan kemungkinan frekuensi lebih banyak. Namun huruf J, X, Q dan Z cukup jarang ditemukan dalam teks bahasa inggris. Anda akan menggunakan fakta ini untuk membantu dalam meretas pesan terenkripsi sandi Vigenère. Teknik ini disebut sebagai analisis frekuensi.

Menghitung huruf dan berapa frekuensi yang muncul diantara teksaawal dan tekssandi disebut juga analisi frekuensi.

Karena sandi Vigenère memiliki kunci perkalian sandi Caesar yang digunakan pada pesan adalah sama, anda dapat menggunakan analisis frekuensi untuk melakukan peretasan terhadap setiap sub kunci satu untuk satu kemungkinan dalam frekuensi huruf untuk mencoba dekripsi. Anda tidak akan mencoba dengan memerhatikan setiap teks berbahasa inggris, karena setiap kata dalam teks sandi akan dienkripsi dengan sub kunci yang dikalilipatkan. Dan anda tidak membutuhkan kumpulan kata dengan

lengkap, anda bisa menganalisa frekuensi huruf pada setiap sub kunci untuk mendekripsi teks.

```
############################################################
# Analisis Frekuensi                                       #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

#dasar frekuensi diambil dari Wikipedia http://en.wikipedia.org/wiki/Letter_frequency
frekuensiPenggunaanHuruf = {'E': 12.70, 'T': 9.06, 'A': 8.17, 'O': 7.51, 'I': 6.97, 'N': 6.75, 'S': 6.33, 'H': 6.09, 'R': 5.99, 'D': 4.25, 'L': 4.03, 'C': 2.78, 'U': 2.76, 'M': 2.41, 'W': 2.36, 'F': 2.23, 'G': 2.02, 'Y': 1.97, 'P': 1.93, 'B': 1.29, 'V': 0.98, 'K': 0.77, 'J': 0.15, 'X': 0.15, 'Q': 0.10, 'Z': 0.07}
ETAOIN = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'
HURUF = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

#pada fungsi ini akan mengambil parameter string dan menghasilkan dictionary/kamus
#dimana hal ini dapat menghitung seberapa sering setiap huruf muncul dalam string
def menghitungJumlahHuruf(pesan):
    #menghasilkan sebuah daftar kamus dengan kunci dari setiap huruf dan nilai
   #dimana hal ini dihitung berapa kali hal tersebut muncul dalam sebuah parameter pesan
    hitungHuruf = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0, 'G': 0, 'H': 0, 'I': 0, 'J': 0, 'K': 0, 'L': 0, 'M': 0, 'N': 0, 'O': 0, 'P': 0, 'Q': 0, 'R': 0, 'S': 0, 'T': 0, 'U': 0, 'V': 0, 'W': 0, 'X': 0, 'Y': 0, 'Z': 0}

    for huruf in pesan.upper():
        if huruf in HURUF:
            hitungHuruf[huruf] += 1

    return hitungHuruf


def menentukanNilaipadaDasar(x):
    return x[0]

#pada fungsi ini akan mengambil parameter string dan menghasilkan string dari 26 huruf
```

```
#diubah dari yang paling sering muncul menjadi parameter yang paling terakhir muncul
def menentukanJumlahPerintah(pesan):
    #mengembalikan sebuah nilai dengan huruf alpabet yang disusun dengan
    #mengatur frekuensi huruf yang muncul paling sering dalam parameter pesan
    #pertama, tentukan dictionary/kamus untuk setiap kata dan frekuensi yang dihitung
    hurufpadaFrekuensi = menghitungJumlahHuruf(pesan)
    #kedua, buatlah sebuah dictionary/kamus dimana setiap frekuensi yang dihitung
    #pada setiap huruf dengan frekuensinya
    frekuensipadaHuruf = {}
    for huruf in HURUF:
        if hurufpadaFrekuensi[huruf] not in frekuensipadaHuruf:
            frekuensipadaHuruf[hurufpadaFrekuensi[huruf]] = [huruf]
        else:
            frekuensipadaHuruf[hurufpadaFrekuensi[huruf]].append(huruf)

    #ketiga, masukkan setiap daftar dari kata dalam pengembalian "ETAOIN"
    #lalu ubah hal tersebut menjadi kalimat
    for frekuensi in frekuensipadaHuruf:
        frekuensipadaHuruf[frekuensi].sort(kunci=ETAOIN.find, reverse=True)
        frekuensipadaHuruf[frekuensi] = ''.join(frekuensipadaHuruf[frekuensi])

    #keempat, ubah setiap kamus freqToLetter menjadi list pertukaran
    #hubungkan (kunci, nilai), lalu urutkan hal tersebut
    frekuensiBerpasangan = list(frekuensipadaHuruf.items())
    frekuensiBerpasangan.sort(kunci=menentukanJumlahPerintah, reverse=True)

    #kelima, kata yang sudah diubahkan sesuai dengan frekuensi
    #ekstrak seluruh kata dengan kalimat penutup
    penugasanFrekuensi = []
    for frekuensiBerpasangan in frekuensiBerpasangan:
        penugasanFrekuensi.append(frekuensiBerpasangan[1])

    return ''.join(penugasanFrekuensi)

#pada fungsi ini akan mengambil parameter string dan menghasilkan integer/bilangan
bulat
#dari 0 hingga 12 dari string yang paling cocok nilainya
def JumlahfrekuensiYangCocok(pesan):
    #menampilkan jumlah kata yang sesuai dengan pesan
```

```
    #parameter memiliki frekuensi kata yang dapat dibandingkan dengan
    #kata berfrekuensi dalam bahasa inggris
    #jika cocok jumlah kata 6 terbanyak dan 6 terakhir yang memiliki frekuensi
    penugasanFrekuensi = menentukanPenugasanFrekuensi(pesan)

    jumlahyangCocok = 0
    #cari berapa jumlah yang sesuai dengan kata terbanyak yang ada disana
    for hurufYangUmum in ETAOIN[:6]:
        if hurufYangUmum in penugasanFrekuensi[:6]:
            jumlahyangCocok += 1
    #cari berapa jumlah yang sesuai dengan kata terbanyak yang ada disana
    for hurufYangtidakUmum in ETAOIN[-6:]:
        if hurufYangtidakUmum in freqOrder[-6:]:
            jumlahyangCocok += 1

    return jumlahyangCocok
```

Output dari program ini tidak ditampilkan namun akan digunakan/diimport oleh program lain untuk membantu dalam peretasan sandi Vigenère.

MELAKUKAN PERETASAN KAMUS/DICTIONARY VIGENÈRE

Ada beberapa hal dimana anda dapat mencoba meretas sandi Vigenère, hal pertama adalah melakukan brute-force menggunakan kamus/dictionary dimana akan mencoba semua kunci yang memungkinkan sebagai kunci Vigenère. Dan yang kedua dengan kunci acak yang telah anda miliki. Awalnya dicatat dan digunakan oleh seorang matematikawan Charles Babbage pada abad ke-19.

Hal yang perlu anda ingat adalah bahwa dalam kunci Vigenère jika kunci yang mudah anda ingat tersusun dari kata/huruf sederhana atau mudah untuk ditebak hal tersebut juga berarti memiliki kemungkinan celah dengan menggunakan teknik penyerangan dengan kamus.

Jika kunci Vigenère dalam kata bahasa inggris sangatlah mudah untuk diingat. Namun jangan pernah menggunakan kata bahasa inggris untuk kunci enkripsi. Hal ini akan membuat teks sandi memiliki celah untuk melakukan dictionary attack/penyerangan terhadap kamus.

Serangan dictionary attack dimaksud adalah teknik brute-force dimana seorang hacker mencoba untuk melakukan dekripsi pada sandi teks menggunakan kata dari file dictionary/kamus sebagai kunci. File dictionary/kamus memiliki 45.000 kata bahasa inggris, dan membutuhkan waktu 5 menit untuk komputer menjalankan seluruh proses dekripsi pesan menjadi paragrap yang panjang.

```
############################################################
# Meretas Sandi Kamus Vigenere                             #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

import deteksibahasa, sandivigenere, pyperclip
```

```
def main():
    sanditeks = """Tzx isnz eccjxkg nfq lol mys bbqq I lxcz."""
    pesanYangTeretas = meretasVigenere(sanditeks)

    if pesanYangTeretas != None:
        print('Menyalin pesan ke layar tampilan:')
        print(pesanYangTeretas)
        pyperclip.copy(pesanYangTeretas)
    else:
        print('Gagal meretas enkripsi')


def meretasVigenere(sanditeks):
    fo = open('dictionary.txt')
    kata_kata = fo.readlines()
    fo.close()

    for kata in kata_kata:
        kata = kata.strip()
        pesanTerdekripsi = sandivigenere.dekripsiPesan(kata, sanditeks)
        if deteksibahasa.cekBahasa(pesanTerdekripsi, IjinkanJumlahKata=40):
            #cek apabila sudah tedekripsi dan sudah ditemukan
            print()
            print('Enkripsi dapat dirusak dengan kemungkinan:')
            print('Kunci ' + str(kata) + ': ' + pesanTerdekripsi[:100])
            print()
            print('Tekan D untuk selesai, atau tekan continue untuk melanjutkan:')
            respon = input('> ')

            if respon.upper().startswith('D'):
                return pesanTerdekripsi

if __name__ == '__main__':
    main()
```

Output yang dihasilkan masih memiliki nilai kesalahan.

SERANGAN BABBAGE DAN ELIMINASI KASISKI

Seperti halnya anda ketahui bahwa Charles Babbage dapat meretas sandi Vigenère, namun beliau tidak pernah mempublikasi hasilnya. Penyelidikan selanjutnya terungkap bahwa metode yang digunakan beliau diungkapkan pada abad 20 oleh matimatikawan Friedrich Kasiski.

Eliminasi Kasiski/Pengujian Kasiski adalah sebuah proses untuk menetapkan berapa panjang kunci Vigenère yang digunakan untuk mengenkripsi tesksandi sebelumnya. Setelah menetapkannya, analisis frekuensi dapat digunaan untuk memecah setiap subkunci.

Bagaimana hal tersebut terjadi?

1. Menetapkan pengulangan jarak rangkaian, yaitu dimana setiap pengulangan pada beberapa kaliamat akan dipisah. Hal-hal yang dianggap penting baik dalam halnya huruf dan sandi teks dengan kunci maupun subkunci.

2. Mencari faktor dari jarak, yaitu dimana jarak antara satu dan dua angka yang berbeda dan memiliki hibungan dicatat dan ditentukan faktornya.

MERETAS SANDI VIGENÈRE

```
############################################################
# Meretas Sandi Kamus Vigenere                             #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

#itertools dan re sudah tersedia saat anda melakukan instalasi python
import itertools, re
import sandivigenere, pyperclip, analisisfrekuensi, deteksibahasa

HURUF = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

#jika True maka program tidak akan mencoba menampilkan pesan
MODE_SENYAP = False

FREKUENSI_HURUF_TERBESAR = 4
PANJANG_KUNCI_MAKSIMAL = 16
POLA_YANG_BUKAN_HURUF = re.compile('[^A-Z]')



def main():
    sanditeks = """Mltv Gsfhbaif Fukqhy ial i Vjutbab emtamgsfivquf, xozqwamn, vzshfagifqet, tvx uamicnwd svqyffilb. Bw ial pcytlr qhxxuxvnaml bv nzq dxdydapfmhl af vwghgtxz muuegky, hdooqxazg t nijyaeqmsfihv ix fhx kifoeibm gr "aeoijutau" ufp "chujmfamqif" iimp nzq Tnzcfs mtkbaze. Mclazg ba qapeeg wgzsblyjqd mw vw fhx nultek wz uamicnwd svqyfoe tvx sdtbncuuae qhlqleqawzcx. Lojunz Eijxd Pil AU, Tnzcfs whzewp fhz nzq Ghdyjzmxvn Uadx ihv Oyipyj Ecawid (SCVA) ul Nlxbwzxer Xujw, Bkqnsun'l kivqbkmucunz kyffrx. Nij m tbuy zq wta bwmd hn Bmf 8, tam mwotbwh jqsiwhkubem zgd Gxzgsz ntdud orrxnszaegmae. Hx lynusxl u fgmuml gr txkbfuqnmm xar uzyswigo Awdmtv wabhxzm, azcecxazg mpy eqtawx gr tam vgybx, ih wxevblgyevpufuctt gsohbvy ltam kimxd yqhv eembcfss ywl lte Xvcyya fiwzunx. Izlqr mpy omr am qgdkxl ul fhx Vuluogif Htylqwsx Ltjijmthzs, otekm bw orxinwp ogm ix fhx ncjet wmmasnl nij m smwlwp-pkwajmm vwghgtxz, nzq AVM. Cf 1948 Fukqhy vobvyv Yaq Vyoyag'a Wgypnbcfs Ltjijmthzs sf Mtvwzqsmml Mziomlkutr, ebwde am ukeilbyv un mpy vqvxtihyegb ix fhx Uufohxanwd chujmfeka ufp bxkueq igbyjqsmmx az mtbbwyamqwsx bbwfgsy. Am qjatx i jsbek wh lte vpyeuctt vseil wz earipiyqnxack, mnw xlwpivbyv asvqfdmtbva utefqwsx rxiwluoga mmoh ta nzq Bxtimeoo-Hbsnomqhkwy kmuufihv, qzuca eyjq fbzml ablmlnqd bv nzq 1960s. Mclazg'l pieasxfosximg lweuebyv
```

```
un t klayigif hdolmwmfihv cf 1952, ihxv bgyolmrmml tknk iekm mlule qfdqgtt cf fhx Chafew Scfsdhu. Bw mcvmjlqd mzysfmxvn outa nyemlx pijyogmm (utefqwsx ctanjmtbwh) se ag iflqrginahe mw jjushv. Nmdigo xaqd bv 1954, dmet hdyj fwh eywws umzgde aqm 42fp bbznzpar, nlgy crihape iwckanbva. Sz igyowet wmnwdmbvyv fhtb bae dxinz ial aoaoiwm; bae mhbbwd agl mgye hbbwds umfaqvxl bae dxinz ial iwuudxvnsx. Og 10 Ayhfefjyj 2009, roetiounz ih Aztxzhwf ctujsugg, Jlafilp Jjumx Ucfusmml Yarwwh Tdopv gspe tv ixrivqud buutcu mphtiyk og jyzmly wz lte Uzclusa oinqrguyff fhz "nzq aixudxigo qsk hx euk frxinwp." Al wz Emy 2012 t xlahamm gwybxz'm tule euk neywlw fhx Pimee hn Fgddl ebaoh pwodp gkihl Fukqhy m sminmfokg jsddhv cx qntknwp."""
    pesanYangTeretas = meretasVigenere(sanditeks)

    if pesanYangTeretas != None:
        print('menyalin pesan yang sudah teretas ke dalam layar:')
        print(pesanYangTeretas)
        pyperclip.copy(pesanYangTeretas)
    else:
        print('Gagal melakukan peretasan terhadap enkripsi.')


def cariRuangKosongberangkaiBerulang(pesan):
    #akan mencoba pesan dan menetapkan 3 hingga 5 huruf untuk dirangkai dan diulang
    #menghasilkan  kunci dari rangkaian dan nilai pada daftar kosong antara jumlah huruf yang diulang

    #gunakan jenis huruf yang umum untuk mengubah pesan dan non pesan dari pesan
    pesan = POLA_YANG_BUKAN_HURUF.sub('', pesan.upper())

    #mengkompilasi daftar panjang rangkaian huruf yang ditemukan dalam pesan
rangkaianRuangKosong = {}
     for panjangRangkaian in range(3, 6):
        for memulaiRangkaian in range(len(pesan) - panjangRangkaian):
            #mendeminasikan bagaimana rangkaian dan menyimpanya
            rangkaian = pesan[memulaiRangkaian:memulaiRangkaian + panjangRangkaian]

            #mencari rangkaian dalam pesan
            for i in range(memulaiRangkaian + panjangRangkaian, len(pesan) - panjangRangkaian):
```

```
            if pesan[i:i + panjangRangkaian] == rangkaian:
                #menemukan perulangan pada rangkaian
                if rangkaian not in rangkaianRuangKosong:
                    rangkaianRuangKosong[rangkaian] = [] # initialize blank list

                #menambahkan ruang kosong sebagai penghubung diantara perulangan
                #rangkaian dan rangkaian asli
                rangkaianRuangKosong[rangkaian].append(i - memulaiRangkaian)
    return rangkaianRuangKosong


def menentukanfaktorYangberguna(angka):
    #menghasilkan daftar factor yang berguna dan menggunakannya
    #tidak kurang dari MAX_KEY_LENGTH + 1. Sebagai contohnya adalah (144)
    # menghasilkan [2, 72, 3, 48, 4, 36, 6, 24, 8, 18, 9, 16, 12]

    #angka yang kurang dari 22 tidak memiliki faktor
    if angka < 2:
        return []

    #daftar factor yang telah ditemukan
    faktor = []

    #ketika menemukan factor, anda hanya harus mencek nilai bilangan bulat yang
mencapai
MAX_KEY_LENGTH.
    for i in range(2, PANJANG_KUNCI_MAKSIMAL + 1):
        if angka % i == 0:
            faktor.append(i)
            faktor.append(int(angka / i))
    if 1 in faktor:
        faktor.remove(1)
    return list(set(faktor))


def menetapkanSatudasarIndex(x):
    return x[1]
```

```
def menentukanFaktorYangPalingUmum(faktorRangkaian):
    #hitung berapa banyak perulangan yang ditetapkan sebagai factor yang ditemukan pada rangkaian faktor
    menghitungFaktor = {} # key is a factor, value is how often if occurs

    #rangkaian factor adalah bentuk-bentuk rangkaian yang memiliki daftar nilai ruang kosong
    #pada rangkaian  factor memiliki nilai seperti: {'GFD': [2, 3, 4, 6, 9, 12,
    # 18, 23, 36, 46, 69, 92, 138, 207],
    for rangkaian in faktorRangkaian:
        daftarFaktor = faktorRangkaian[rangkaian]
        for faktor in daftarFaktor:
            if faktor not in menghitungFaktor:
                menghitungFaktor[faktor] = 0
            menghitungFaktor[faktor] += 1

    #setelah itu masukkan factor dan hitung lalu buat daftarnya
    menghitungdenganFaktor = []
    for faktor in menghitungFaktor:
        #tidak termasuk factor yang lebih besar dari  MAX_KEY_LENGTH
        if faktor <= PANJANG_KUNCI_MAKSIMAL:
            #faktor yang dihitung adalah daftar dari factor dan hitung faktor
            #faktor yang dihitung memiliki nilai seperti [(3, 497), (2, 487), ...]
            menghitungdenganFaktor.append( (faktor, menghitungFaktor[faktor]) )

    #urutkan daftar dengan factor yang dihitung
    menghitungdenganFaktor.sort(key=menetapkanSatudasarIndex, reverse=True)

    return menghitungdenganFaktor

def pengujianKasiski(sanditeks):
    #menetapkan rangkaian dari 3 hingga 5 huruf yang didapatkan dari perulangan yang berkali-kali di dalam pesan teks,
    #perulangan jarak rangkaian memiliki nilai seperti
    # {'EXG': [192], 'NAF': [339, 972, 633], ... }
    melakukanUrutanRuangkosong = cariRuangKosongberangkaiBerulang(sanditeks)

    #menentukan penetapan factor yang paling umum sebagai dekripsi dari factor rangkaian
```

```
    faktorRangkaian = {}
    for rangkaian in melakukanUrutanRuangkosong:
        faktorRangkaian[rangkaian] = []
        for jarak in melakukanUrutanRuangkosong[rangkaian]:
            faktorRangkaian[rangkaian].extend(menentukanfaktorYangberguna(jarak))

    #menentukan penetapan factor yang paling umum sebagai dekripsi dari factor yang dihitung
    menghitungdenganFaktor = menentukanFaktorYangPalingUmum(faktorRangkaian)

    #ekstrak factor yang dihitung dari factor yang dihitung dan masukkan seluruhnya dalam panjang kunci
     #yang menyerupai atau sama dengan begitu akan lebih mudah untuk digunakan nantinya
    seluruhKunciyangMemungkinkan = []
    for duaBilangan in menghitungdenganFaktor:
        seluruhKunciyangMemungkinkan.append(duaBilangan[0])

    return seluruhKunciyangMemungkinkan


def menentukanNSubKunciHuruf(n, panjangKunci, pesan):
    #menghasilkan setiap nilai ke-N pada setiap panjang kunci yang ditentukan oleh huruf dalam teks
    #contoh menetapkan nilai ke –N sub kunci huruf (1, 3, 'ABCABCABC') menghasilkan 'AAA'
    #       menetapkan nilai ke –N sub kunci huruf (2, 3, 'ABCABCABC') menghasilkan 'BBB'
    #       menetapkan nilai ke –N sub kunci huruf (3, 3, 'ABCABCABC') menghasilkan 'CCC'
    #       menetapkan nilai ke –N sub kunci huruf (1, 5, 'ABCDEFGHI') menghasilkan 'AF'

    #menggunakan ekspressi yang umum untuk menentukan jenis huruf dan bukan huruf pada pesan
    pesan = POLA_YANG_BUKAN_HURUF.sub('', pesan)

    i = n - 1
    huruf = []
```

```
    while i < len(pesan):
        huruf.append(pesan[i])
        i += panjangKunci
    return ''.join(huruf)


def mencobaMeretasDenganPanjangKunci(sanditeks, kunciYangpalingmemungkinkan):
    #menetapkan huruf yang paling sering ada disetiap huruf dalam kunci
    sanditeksKapital = sanditeks.upper()
    #seluruh nilai frekuensi yang ada pada panjang huruf yang menyerupai dalam panjang kunci yang paling cocok jumlahnya pada daftar
    #dalam dafatr terdapat daftar frekuensi nilai
    seluruhNilaiFrekuensi = []
    for nilaiKeN in range(1, kunciYangpalingmemungkinkan + 1):
        hurufKeN = menentukanNSubKunciHuruf(nilaiKeN, kunciYangpalingmemungkinkan, sanditeksKapital)

        #nilai frekuensi adalah daftar dari penyatuan dua hal seperti huruf, bahasa, frekuensi dan nilai yang cocok
        # daftar yang telah diurutkan dengan kesamaan nilai contohnya dari yang paling banyak kesamaan
        #silahkan cek dari nilai frekuensi kecocokan bahasa pada program analisis frekuensi
        frekuensiNilai = []
        for kunciYangMemungkinkan in HURUF:
            teksTerdekripsi = sandivigenere.dekripsiPesan(kunciYangMemungkinkan, hurufKeN)
            kunciDanFrekuensiYangcocok = (kunciYangMemungkinkan, analisisfrekuensi.JumlahfrekuensiYangCocok(teksTerdekripsi))
            frekuensiNilai.append(kunciDanFrekuensiYangcocok)
        #urutkan dengan kecocokan nilai
        frekuensiNilai.sort(key=menetapkanSatudasarIndex, reverse=True)


seluruhNilaiFrekuensi.append(frekuensiNilai[:FREKUENSI_HURUF_TERBESAR])

    if not MODE_SENYAP:
        for i in range(len(seluruhNilaiFrekuensi)):
            #gunakan i+1 untuk huruf yang pertama disebut dengan  huruf ke 0
```

```
            print('Kemungkinan huruf yang cocok  %s dengan kunci: ' % (i + 1), end='')
            for frekuensiNilai in seluruhNilaiFrekuensi[i]:
                print('%s ' % frekuensiNilai[0], end='')
            #menampilkan baris baru
            print()

    #coba dengan setiap kombinasi dari yang paling cocok dengan huruf pada setiap posisi pada kunci
    for terindeks in itertools.product(range(FREKUENSI_HURUF_TERBESAR), repeat=kunciYangpalingmemungkinkan):
        #membuat kunci yang memungkinkan dari huruf dalam nilai  frekuensi keseluruhan
        kunciYangMemungkinkan = ''
        for i in range(kunciYangpalingmemungkinkan):
            kunciYangMemungkinkan += seluruhNilaiFrekuensi[i][terindeks[i]][0]

        if not MODE_SENYAP:
            print('Mencoba dengan kunci: %s' % (kunciYangMemungkinkan))

        pesanTerdekripsi = sandivigenere.dekripsiPesan(kunciYangMemungkinkan, sanditeksKapital)

        if deteksibahasa.cekBahasa(pesanTerdekripsi):
            #menentukan teks awal yang dapat diretas menjadi bentuk asli
            bentukAsli = []
            for i in range(len(sanditeks)):
                if sanditeks[i].isupper():
                    bentukAsli.append(pesanTerdekripsi[i].upper())
                else:
                    bentukAsli.append(pesanTerdekripsi[i].lower())
            pesanTerdekripsi = ''.join(bentukAsli)

            #cek dengan pengguna apabila kunci telah ditemukan
            print('Possible encryption hack with key %s:' % (kunciYangMemungkinkan))
            #hanya menampilkan 200 karakter
            print(pesanTerdekripsi[:200])
            print()
            print('Untuk berhenti tekan D, untuk melanjutkan tekan Enter:')
            respon = input('> ')
```

```
            if respon.strip().upper().startswith('D'):
                return pesanTerdekripsi

    #tidak menemukan bahasa dengan dekripsi yang ditentukan dan menghasilkan nilai kembali 0 (none)
    return None


def meretasVigenere(sanditeks):
    #lakukan pengujian Kasiski untuk menggambarkan bagaimana panjang sandi teks enkripsi dari kunci
    seluruhKunciyangMemungkinkan = pengujianKasiski(sanditeks)
    if not MODE_SENYAP:
        panjangKunciString = ''
        for panjangKunci in seluruhKunciyangMemungkinkan:
            panjangKunciString += '%s ' % (panjangKunci)
        print('Pengujian Kasiski menghasilkan apa yang disebut dengan panjang kunci yang menyerupai tersebut : ' + panjangKunciString + '¥n')

    for panjangKunci in seluruhKunciyangMemungkinkan:
        if not MODE_SENYAP:
            print('Mencoba meretas dengan panjang kunci %s (%s kemungkinan kunci)...' % (panjangKunci, FREKUENSI_HURUF_TERBESAR ** panjangKunci))
        pesanYangTeretas = mencobaMeretasDenganPanjangKunci(sanditeks, panjangKunci)
        if pesanYangTeretas != None:
            break

    #jika tidak ditemukan panjang kunci dengan pengujian Kasiski maka lakukan brute-force pada panjang kunci
    if pesanYangTeretas == None:
        if not MODE_SENYAP:
            print('Tidak dapat meretas pesan dengan panjang kunci. Panjang kunci brute-force...')
        for panjangKunci in range(1, PANJANG_KUNCI_MAKSIMAL + 1):
            #tidak memerlukan pengecekan kembali pada pengujian Kasiski
            if panjangKunci not in seluruhKunciyangMemungkinkan:
                if not MODE_SENYAP:
```

```
                print('Mencoba meretas dengan panjang kunci %s (%s kunci yang memungkinkan)...' % (panjangKunci, FREKUENSI_HURUF_TERBESAR ** panjangKunci))
                pesanYangTeretas = mencobaMeretasDenganPanjangKunci(sanditeks, panjangKunci)
                if pesanYangTeretas != None:
                    break
    return pesanYangTeretas


#jika program sudah berjalan dengan semestinya dan seluruh program yang diimport berjalan dengan lancar
if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Kasiski Examination results say the most likely key lengths are: 3 2 6 4 12 8 9 16 5 11 10 15 7 14 13

Attempting hack with key length 3 (64 possible keys)...
Possible letters for letter 1 of the key: A L M E
Possible letters for letter 2 of the key: S N O C
Possible letters for letter 3 of the key: V I Z B
Attempting with key: ASV
Attempting with key: ASI
Attempting with key: ASZ
Attempting with key: ASB
Attempting with key: ANV
Attempting with key: ANI
Attempting with key: ANZ
Attempting with key: ANB
Attempting with key: AOV
Attempting with key: AOI
Attempting with key: AOZ
Attempting with key: AOB
Attempting with key: ACV
Attempting with key: ACI
Attempting with key: ACZ
Attempting with key: ACB
Attempting with key: LSV
Attempting with key: LSI
Attempting with key: LSZ
Attempting with key: LSB
Attempting with key: LNV
Attempting with key: LNI
Attempting with key: LNZ
Attempting with key: LNB
Attempting with key: LOV
Attempting with key: LOI
Attempting with key: LOZ
Attempting with key: LOB
Attempting with key: LCV
Attempting with key: LCI
Attempting with key: LCZ
```

```
Attempting with key: LCB
Attempting with key: MSV
Attempting with key: MSI
Attempting with key: MSZ
Attempting with key: MSB
Attempting with key: MNV
Attempting with key: MNI
Attempting with key: MNZ
Attempting with key: MNB
Attempting with key: MOV
Attempting with key: MOI
Attempting with key: MOZ
Attempting with key: MOB
Attempting with key: MCV
Attempting with key: MCI
Attempting with key: MCZ
Attempting with key: MCB
Attempting with key: ESV
Attempting with key: ESI
Attempting with key: ESZ
Attempting with key: ESB
Attempting with key: ENV
Attempting with key: ENI
Attempting with key: ENZ
Attempting with key: ENB
Attempting with key: EOV
Attempting with key: EOI
Attempting with key: EOZ
Attempting with key: EOB
Attempting with key: ECV
Attempting with key: ECI
Attempting with key: ECZ
Attempting with key: ECB
Attempting hack with key length 2 (16 possible keys)...
Possible letters for letter 1 of the key: O A E Z
Possible letters for letter 2 of the key: M S I D
Attempting with key: OM
Attempting with key: OS
Attempting with key: OI
Attempting with key: OD
Attempting with key: AM
Attempting with key: AS
Attempting with key: AI
Attempting with key: AD
Attempting with key: EM
Attempting with key: ES
Attempting with key: EI
Attempting with key: ED
Attempting with key: ZM
Attempting with key: ZS
Attempting with key: ZI
Attempting with key: ZD
Attempting hack with key length 6 (4096 possible keys)...
Possible letters for letter 1 of the key: A E O P
Possible letters for letter 2 of the key: S D G H
Possible letters for letter 3 of the key: I V X B
Possible letters for letter 4 of the key: M Z Q A
Possible letters for letter 5 of the key: O B Z A
Possible letters for letter 6 of the key: V I K Z
Attempting with key: ASIMOV
Possible encryption hack with key ASIMOV:
Alan Mathison Turing was a British mathematician, logician, cryptanalyst, and computer scientist. He was highly influential in the development of
computer science, providing a formalisation of the con

Enter D for done, or just press Enter to continue hacking:
```

SANDI ONE-TIME PAD (OTP)

Ada satu sandi yang hampir tidak mungkin untuk di pecah/dirusak, tidak peduli seberapa tangguh komputermu, berapa lama waktu yang kamu habiskan untuk memecahnya, atau seberapa pintarnya kamu menjadi seorang peretas. Anda tidak harus menulis program baru untuk menggunakannya di lain waktu. Program Vigenere dapat diterapkan pada sandi ini tanpa harus membuat perubahan yang berarti. Namun sandi ini tidak nyaman apabila penggunaannya bersifat monoton/tetap hal ini sering digunakan pada pesan yang sangat amat rahasia.

Pada sandi Vigenère anda temukan perihal :

1. Panjang kunci memiliki panjang yang sama dengan pesan yang terinkripsi.

2. Kunci disusun dengan susunan yang sangat acak.

3. Kunci yang digunakan hanya sekali saja, dan tidak akan pernah digunakan kembali untuk pesan apapun.

Dengan mengikuti aturan ini, anda mengenkripsi pesan yang tidak memiliki celah untuk setiap serangan yang digunakan kriptanalis. Bahkan dengan kata harafiahnya jika komputer anda memiliki perhitungan yang tidak ada habisnya sandi tetap tidak akan dapat dirusak.

Kunci pada OTP(One-Time Pad) disebut Pad karena menampilkan pesan pada pada kertas. Bagian atas dari kertas akan diambil setelah itu akan anda gunakan memperlihatkan kunci selanjutnya untuk digunakan.

Kenapa OTP tidak dapat dirusak ?

Mengapa OTP tidak terpecahkan, coba fikirkan tentang bagaimana halnya sandi Vigenère memiliki celah dan digunakan untuk merusak. Program Sandi Vigenère untuk dapat di retas bekerja dengan menggunakan analisis frekuensi. Namun jika kunci

memiliki panjang yang sama dengan pesan, maka setiap kemungkinan sandi teks huruf memilik kemungkinan yang sama menghasilkan teks awal huruf.

Contohnya jika anda ingin mengenkripsi teks awal pada OTP 'matiuscelciussinaga'(berjumlah 19 huruf) akan menghasilkan sebanyak 7.66467265E+26 (26 ^ 19) kemungkinan kunci, bahkan jika anda melakukan brute-force hal tersebut tidak akan bekerja karena terlalu banyak perhitungan yang dilakukan komputer. Apakah anda masih yakin bahwa sebuah komputer anda memecahnya ?

Pada OTP besarnya kemungkinan bahwa seorang kriptanalis tidak mengetahui manakah kunci yang asli. Namun pada dasarnya ada kemungkinan dari teks awal pesan yang memiliki panjang 19 huruf dapat dijadikan sebagai pesan awal yang asli. Tapi tidak berarti bahwa jikapun anda mampu melakukan dekripsi terhadap pesan awal berarti anda sudah mendapatkan kunci asli enkripsi.

TWO-TIME PAD (TTD)

Jika anda menggunakan OTP secara bersamaan untuk mengenkripsi dua pesan berbeda, maka anda telah mengetahui kelemahan enkripsi. OTP dan TTD pada dasarnya adalah hal yang sama.

Jika hanya karena sebuah kunci mendekripsi sebuah OTP teks sandi sehingga dapat dibaca bukan berarti hal tersebut adalah kunci yang benar. Bagaimanapun jika anda menggunakan kunci yang sama pada pesan yang berbeda maka seorang akan mengetahui jika sebuah kunci mendekripsi pesan awal agar dapat dibaca tetapi untuk mendekripsi dengan kunci yang sama pada pesan kedua pada pesan sampah yang acak tidaklah mengharuskan kunci asli. Pada dasarnya seperti satu kunci yang akan mendekripsi dua pesan agar dapat dibaca.

Jika seorang peretas hanya memiliki satu kunci dari dua pesan dan maka hal ini tetap dalam kondisi enkripsi, ketahuilah bahwasanya semua pesan yang terenkripsi akan ditangkap oleh peretas lainnya dan kemungkinan juga pemerintah dan tak usah takut akan hal tersebut.

Sederhananya, OTP sama halnya dengan sandi Vigenère dengan kunci yang sama panjangnya dengan pesan dimana hanya digunakan sekali. Selama kedua hal ini anda ikuti, secara harafiah OTP tidak mungkin untuk dipecahkan . Bagaimanapun, hal ini merepotkan dalam penggunaan OTP dimana hal ini tidak umum digunakan kecuali hal yang sangat rahasia. Biasannya semakin besar daftar OTP yang diubah dan dibagikan pada orang yang dimaksud, dengan waktu yang khusus. Dalam hal ini, jika anda menerima pesan berasal manapun khususnya pada 31 Oktober, anda dapat langsung melihat daftar OTP yang dimaksud. Namun jangan sampai jatuh pada orang yang salah.

MENCARI ANGKA PRIMER

Pada awalnya seluruh sandi yang anda gunakan diatas membutuhkan waktu untuk dapat diretas bahkan hampir tidak mungkin namun setelah beberapa waktu bahkan ratusan tahun berlalu dan penelitian berlanjut, berkembang menjadikan hal tersebut menjadi mungkin, pesan tersebut kini dapat di pecah, didekripsi. Namun penelitian dan perkembangan sandi tetap berjalan untuk mencapai apa yang disebut dengan keamanan maksimal, pengujian demi pengujian yang menghasilkan inovasi-inovasi dari ide kreatif yang terkait.

Sebuah angka primer memiliki lebih dari 1 faktor dan setidaknya dua faktor. Angka 1 dan 21 adalah faktor dari 21. Angka 1 dan 12 adalah faktor dari 12. Hal ini dikarenakan 1 dari setiap angka berarti selalu menghasilkan nilai yang sama. Namun jika tidak ada faktor lain pada angka tersebut, maka angka tersebut adalah primer.

Ini adalah deretan angka primer (angka 1 bukanlah angka primer) :

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281 …dan selanjutnya

Googol adalah angka dari 1 dengan ratusan nol dibelakangnya : 10,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000, 000,000,000,000,000,000,000,000,000,000,000,000 (sedikit mengenai sejarah : nama search engine perusahaan Google berasal dari kesalahan abjad "googol", namun akhirnya mereka memutuskan bahwa ejaannya lebih baik dan membiarkannya)

Satu milyar milyar milyar googols memiliki lebih dari 27 angka nol daripada googol

:10,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000

,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000

Namun dalam angka lebih sederhana. Tipe angka primer yang digunakan pada

program RSA memiliki ratusan angka :

112,829,754,900,439,506,175,719,191,782,841,802,172,556,768,253,593,054,977,186,2355,8

4,979,780,304,652,423,405,148,425,447,063,090,165,759,070,742,102,132,335,103,295,947,0

00,718,386,333,756,395,799,633,478,227,612,244,071,875,721,006,813,307,628,061,280,861,

610,153,485,352,017,238,548,269,452,852,733,818,231,045,171,038,838,387,845,888,589,41

1,762,622,041,204,120,706,150,518,465,720,862,068,595,814,264,819

Saya tidak yakin bahwa anda akan membaca angka tersebut sampai habis namun percaya saja tidak ada typo pada deretan angka tersebut.

GABUNGAN ANGKA

Bilangan bulat bukanlah angka primer yang disebut gabungan angka, hanya karena tergabung pada dua faktor yang saling bersebelahan antara 1 angka dan lainnya. Angka gabungan dapat dikatakan begitu apabila angka tersebut gabungan dari angka primer yang dikalikan bersama, seperti halnya 1.386 adalah gabungan dari angka primer dari 2 x 3 x 3 x 7 x 11.

Beberapa faktor mengenai angka primer :

1. Angka primer adalah bilangan bulat yang lebih besar dari 1 dan hanya memiliki 1 dan dirinya sebagai faktor.

2. Dua adalah angka primer dan juga angka ganjil.

3. Pada angka primer tidak memiliki batas maka hal tersebut menjadikan bahwasanya tidak ada angka primer terbesar.

4. Perkalian dua angka primer akan menghasilkan dua angka primer yang hanya memiliki dua pasang faktor yang sama, 1 dan dirinya sendiri, dan dua angka primer yang dikalikan. Sebagai contoh, 3 dan 7 adalah angka primer, faktornya adalah 21 yaitu 1, 21, 3 dan 7.

Sieve dari Eratosthenes (melafalkannya “era, taws, thuh, knees”) adalah sebuah algoritma untuk menghitung angka primer.

Dalam fungsi adalahPrimer() dalam program angkaprimersieve.py melakukan cek jika kondisi angka dapat dibagi dengan rentang akar kuadrat dari 2. Tetapi bagaimana bila angka yang dihasilkan adalah 1,070,595,206,942,983 ? untuk mengecek apakah angka tersebut adalah primer atau tidak akan membutuhkan waktu yang lumayan lama. Dan

jikapun angka dalam jumlah ratusan maka akan membutuhkan waktu hingga triluyan tahun untuk mengetahui bahwa angka tersebut adalah primer atau bukan.

```
##########################################################
# Angka Primer Sieve                                     #
# Editor : Matius Celcius Sinaga                         #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
##########################################################

import math

def adalahPrimer(angka):
    #menghasilkan True apabila angka adalah angka primer
    #contohnya adalah bahwa angka dibawah 2 adalah bukan primer
    #menghasilkan True apabila angka yang dimaksud angka primer, vice versa
    if angka < 2:
        return False

    #jika sebuah angka dapat dibagi dua dengan setiap angka hingga
    #perpangkatan dua akar dari angka tersebut
    for i in range(2, int(math.sqrt(nomor)) + 1):
        if angka % i == 0:
            return False
    return True


def SievePrimer(ukuranSieve):
    #menghasilkan sebuah daftar angka primer yang dapat dihitung
    #menggunakan algoritma Sieve dari Eratosthenes
    sieve = [True] * ukuranSieve
    #angka nol dan satu (dibawah dua) bukanlah angka primer
    sieve[0] = False
    sieve[1] = False

    #proses menghasilkan sieve
    for i in range(2, int(math.sqrt(ukuranSieve)) + 1):
        pointer = i * 2
        while pointer < ukuranSieve:
            sieve[pointer] = False
```

```
            pointer += i
    #mengkompilasi daftar angka primer
    primes = []
    for i in range(ukuranSieve):
        if sieve[i] == True:
            primes.append(i)

    return primes
```

RABIN MILLER

```
############################################################
# Rabin Miller                                             #
# Editor : Matius Celcius Sinaga                           #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
############################################################

import random

def rabinMiller(angka):
    #mengembalikan nilai Benar jika num adalah angka primer

    s = angka - 1
    t = 0
    while s % 2 == 0:
        #s tetap setengah hingga menghasilkan nilai genap
        #gunakan t untuk menghitung berapa banyak jumlah setengah dari s
        s = s // 2
        t += 1

    for mencoba in range(5):
    #mencoba untuk memalsukan 5 kali angka primer
        a = random.randrange(2, angka - 1)
        v = pow(a, s, angka)
        if v != 1:
        #test ini tidak akan menerima jika v adalah 1
            i = 0
            while v != (angka - 1):
                if i == t - 1:
                    return False
                else:
                    i = i + 1
                    v = (v ** 2) % angka
    return True

def adalahPrimer(angka):
    #menghasilkan True jika num adalah angka primer
    #fungsi ini lberjalan lebih cepat
    #angka primer melakukan cek sebelum memanggil rabinMiller()
```

```
    if (angka < 2) :
        return False

    #0, 1, dan angka negatif bukanlah angka primer
    #1/3 kali lebih cepat anda mampu menentukan jika num bukanlah angka primer
    #dengan membagi dengan yang pertama beberapa lusin angka primer
    #hal ini lebih cepat dari rabinMiller(), namun berbeda dengan rabinMiller hal ini
tidak menjamin
    #menunjukkan bahwa angka tersebut adalah primer
    primerDasar = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167,
173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269,
271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379,
383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487,
491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607,
613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727,
733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853,
857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977,
983, 991, 997]

    if angka in primerDasar:
        return True

    #melihat apakah setiap angka kecil primer dapat dibagi dengan angka
    for prime in primerDasar:
        if (angka % prime == 0):
            return False

    #jika seluruh kemungkinan gagal, panggil rabinMiller() untuk menentukan jika num
adalah angka primer
    return rabinMiller(angka)

def generateLargePrime (ukuranKunci=1024):
    #menghasilkan angka primer yang acak dari ukuran kunci
    while True:
        angka = random.randrange(2**(ukuranKunci-1), 2**(ukuranKunci))
        if isPrime(angka):
            return angka
```

Angka primer memiliki sesuatu yang khusus dalam matematika. Seperti yang anda ketahui angka primer dapat digunakan sebagai inti dari sandi, biasanya digunakan pada teknik enkripsi software profesional. Jika saya mencoba untuk mendefinisikan secara sederhana yaitu sebuah angka dimana terdiri dari satu dan faktor dari dirinya sendiri. Namun untuk membedakan setiap angka adalah primer atau bukan dan juga kumpulannya akan sangat membutuhkan waktu yang tidak sedikit meski sudah dengan kode yang sangat cerdas.

Saat ini, kriptografi yang kuat dan sulit untuk di retas banyak digunakan sebagai perlindungan bisnis dan pengusaha online dibanyak jutaan toko online yang menjalankan bisnisnya setiap waktu. Kriptografi menjadi suatu dasar yang kuat dimana diperlukannya juga pengembangan sesuai perkembangan global yang secara terus-menerus tiada hentinya. Pada tahun 1990 hal seperti berbagi pengetahuan seperti yang dilakukan oleh buku ini adalah hal yang illegal dan dapat membahayakan diri anda sendiri, maka beruntunglah jika saat ini semua telah berubah dimana anda dapat menikmati berbagai ilmu secara bebas namun tetap bertanggung jawab pada apa dan bagaimana anda melakukan sesuatu hal di dunia maya.

Untuk detail sejarah lengkap Kriptografi dapat saya referensikan pada buku Steven Levy's. Crypto : How the Code Rebels Beat the Government, Saving Privacy in the Digital Age.

Rasa takut dan keraguan adalah ilusi mengapa anda tak lagi berkembang diciptakan dari kelompok orang-orang yang ahli akan bidangnya membuat ancaman dan keamanan anda terganggu untuk merahasiakan apapun dan bagaimana anda meraih sesuatu namun bayang-bayang tentang ketakutan semakin terungkap hingga ragu tak lagi membekas berjalan berderap beriring-iring melihat semua dan mengetahuinya.

Pernahkah anda berfikir tentang bagaimana jika anda berbagi suatu pesan(data) yang telah terenkripsi pada orang yang anda tidak pernah anda temui atau kenal sebelumnya ?

Katakanlah seseorang yang berada pada sudut lain bumi ini ingin melakukan kontak(komunikasi) dengan anda. Tetapi anda tidak ingin orang lain tahu

karena seiring perkembangan zaman jaringan data yang digunakanpun dicurigai adanya akses dari pemerintah atau pihak yang ingin mengambil keuntungan. Mungkin salah satu caranya berbagi kunci rahasia terhadap file yang terenkripsi yang sebelumnya disepakati dahulu. Hal itupun tidak boleh dilakukan dalam pesan yang melibatkan jaringan data yang dicurigai, karena apa ? Hal itu sama saja memberikan akses terhadap orang lain dengan membaca kunci tersebut sebelumnya. Ini akan menjadi masalah baru apabila orang yang anda tuju berada pada jarak yang tidak memungkinkan untuk bertemu.

Masalah seperti diatas dapat diatasi dengan menggunakan Kunci Publik Kriptografi dimana sandi ini memiliki dua kunci, satu digunakan sebagai enkripsi dan satu lagi digunakan sebagai dekripsi. Kunci sandi yang dapat digunakan untuk enkripsi dan dekripsi disebut Asymmetric, Lalu dimana sandi digunakan untuk mengirim kunci yang sama untuk melakukan dekripsi dan enkripsi disebut sandi Symmetric.

Yang perlu anda ketahui adalah sebuah pesan yang telah terenkripsi dengan satu kunci hanya dapat didekripsi dengan kunci yang lain. Jadi apabila seseorang mendapatkan kunci enkripsi belum tentu mereka dapat membaca pesan yang telah terenkripsi karena kunci enkripsi hanya dapat mengenkripsi namun tidak dapat mendekrip pesan yang telah terenkripsi.

Jadi jika anda memiliki dua kunci ini, anda memanggil satu kunci publik dan satu kunci private. Kunci publik adalah yang telah disebarkan/dipublikasikan. Namun kunci private

tetaplah menyimpan rahasia. Jika Andin ingin mengirim sebuah pesan kepada Bob karena Andin sebelumnya telah memiliki/mengetahui kunci publik Baim dimana hal itu dimanfaatkan sebagai bentuk enkripsi dalam komunikasi rahasia antara Andin dan Bob. Orang lain boleh saja mengetahui kunci publik Bob namun ketika Bob menerima pesan ia akan menggunakan kunci private untuk mendekripnya, untuk membalas Andin, Bob harus mengetahui kunci publiknya dan mengenkripsi balasan tersebut maka Andin mengetahui kunci private Bob.

Ingatlah ketika mengirim pesan terekripsi dengan sandi kunci publik :

1. Kunci publik digunakan untuk mengenkripsi.

2.Kunci private digunakan untuk mendekrip.

Untuk kembali pada contoh dari komunikasi dengan seseorang dalam dunia maya tidak masalah apabila seseorang mendapatkan kunci publik milikmu bahkan jika itu orang yang tidak bertanggung jawab, mereka tidak akan bisa membaca pesan yang telah terenkripsi dengan kunci publik. Hanya kunci private yang dapat mendekripsi pesan tersebut dan tetaplah menjaganya agar dalam kerahasiaan anda pribadi.

Secara khusus sandi kunci publik yang telah diterapkan dan disebut sandi RSA dikembangkan pada tahun 1977 dan dinamakan sesuai dengan pengembangnya : Ron Rivest, Adi Shamir and Leonard Adleman.

RESIKO PUBLIKASI RSA

Anda tidak akan menulis program hacking program sandi RSA dari buku ini, agar tidak ada kesalahpahaman tentang program sandi RSA nantinya, program RSA sudah benar-benar aman. Untuk mendapatkan hak dalam melakukan Kriptografi sangatlah sulit dan membutuhkan banyak pengalaman untuk benar-benar paham tentang sandi dan programnya dengan cara yang benar-benar aman.

Program RSA yang akan anda bahas hanya sebatas buku pelajaran semata saja, ketika hal tersebut diimplementasikan menggunakan algoritma yang benar dengan angka primer yang luas, akan ada beberapa bagian kesalahan pada hal ini yang akan memberikan celah pesan terinkripsi untuk bisa di retas. Perbedaan antara pseudorandom dan angka yang benar-benar acak sehingga mengubah fungsi adalah salah satu kesalahan. Dan masih banyak contoh lainnya di luar sana

Jadi jika anda tidak mampu meretas teks sandi yang telah dibuat dengan program sandi RSA bukan berarti orang lain tidak bisa. Seorang kriptografer Bruce Scheier pernah berkata, seseorang yang paling amatir tanpa pengetahuan adalah kriptografer yang terbaik, dapat membuat sebuah algoritma yang tidak seorangpun dapat meretas. Bahkan dirinya sendiri. Hal ini bahkan tidaklah susah. Apa susahnya membuat sebuah algoritma yang tak seorangpun mampu merusaknya, bahkan setelah beberapa tahun menganalisis. Dimana para kriptografer akan mulai menganalisa dan menganalisa kemungkinan yang ada dan itu akan membuang waktu yang tidak sedikit. :)

HAL DALAM MEMBUKTIKAN SESUATU

Terkadang anda abai mengenai kunci sandi publik. Bayangkan apabila anda menerima pesan begini :

> *"Hello saya ini Andini mantan kamu, boleh share publik keynya. Aku tiba-tiba kangen sama kamu :') ."*

Anda sudah mengetahui sebelumnya bahwa kunci publik membuat anda mampu untuk mengirimkan pesan dengan tingkat rahasia yang tinggi dan membuat orang selain Andini tidak bisa mengetahui apa balasan kamu. Tapi bagaimana anda benar-benar yakin bahwa itu adalah Andini jangan-jangan itu Butet pacar kamu yang lagi menguji rasa setia seorang Bob (anggap saja nama kamu sekarang Bob) dan jika salah dalam menganalisa hancur sudah apa yang sudah kamu jalin dengan Butet si boru batak. Maka anda harus benar-benar yakin bahwa Andini adalah orang dimaksud dengan melakukan pengujian dan melihat bagaimana dan benarkah Andini dapat melakukan hal tersebut.

Pada dasarnya ini bukanlah menjadi masalah sandi Symmetric, karena ketika anda mengubah kunci dengan orang tersebut anda dapat mengetahuinya. Bagaiamanapun jika anda tidak ingin seseorang dapat mengetahui kunci publik anda sendiri dan sedang berbagi pesan rahasia maka tetaplah jaga kunci publik kriptografi tersebut.

Ada hal yang disebut dengan PKI (Public Key Infrastructure) yang membuktikan bahwa kebenaran dan kesamaan kunci publik dengan beberapa tingkat keamanan.

Man In The Middle Attack

Yang paling berbahaya dalam peretasan pesan terenkripsi adalah sesorang pada bagian tengah jaringan atau Man In The Middle Attack. Katakanlah seseorang sangat ingin berkomunikasi dengan anda dan mengirimi anda beberapa pesan, namun seseorang mencegatnya dalam jaringan anda. MITMA dapat mengubah kunci publik yang disematkan dalam email dengan kunci publik mereka, lalu mengirimnya kepada anda. Anda akan berfikir bahwa kunci tersebut berasal dari orang yang anda maksudkan.

Sekarang ketika anda mengenkripsi balasan pada penerima, pesan akan tersaring dan ditangkap oleh MITMA lalu mebacanya dengan kunci publik anda yang sudah terlebih dahulu diketahui pihak MITMA lalu mengirimkannya pada penerima yang anda maksud. Dan hal itu akan terus berlanjut.

Anda dan penerima akan berfikir bahwasanya pesan anda masih dalam kerahasiaan. Namun sebenarnya seseorang telah mengetahui apa yang sedang anda dan penerima rahasiakan, membacanya dan menggunakannya. Anda dan penerima akan mengenkripsi pesan anda dengan kunci publik yang telah diubah. Sekali lagi, masalah ini disebabkan oleh fakta kunci publik tidak hanya memberikan keamanan namun juga pengenalan.

MENGUBAH KUNCI PUBLIK DAN PRIVATE

Sebuah kunci pada skema RSA dibuat oleh dua angka. Cara membuat kunci tersebut :

1. Buat dua angka acak, angka primer yang sangat luas. Angka ini akan dinamakan p dan q. Kalikan angka ini untuk mendapatkan apa yang anda sebut sebagai n.

2. Buatlah angka acak yang disebut dengan e, dimana angka primer relatif dengan $(p - 1) \times (q - 1)$.

3. Hitunglah kebalikan modular dari e. Dan disebut sebagai d.

Kunci publik berasal dari dua angka yaitu n dan e. Kunci private akan menjadi dua angka disebut n dan d. Anda akan melengkapi bagaimana enkripsi dan dejripsi dengan angka ini ketika program sandi RSA dijelaskan.

```
#############################################################
# Membuat kunci RSA publik/private                          #
# Editor : Matius Celcius Sinaga                            #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)  #
#############################################################

import random, sys, os, rabinMiller, kriptomath


def main():
    #membuat kunci publik dan privat dengan 1024 bit kunci
    print('Sedang membuat kunci...')
    buatKunciFile('matius_celcius_sinaga', 1024)
    print('Kunci file sudah dibuat.')
```

```
def hasilkanKunci(ukuranKunci):
    #membuat sebuah kunci publik dan private dengan kunci dengan ketentuan panjang kunci
    #hal ini akan berlangsung sedikit lebih lama dibandingkan program lainnya

    #langkah pertama, buatlah dua angka primer p dan q. Hitung n = p * q
    print('Mengubah p primer...')
    p = rabinMiller.menghasilkanAngkaPrimeryangLuas(ukuranKunci)
    print('Mengubah q primer...')
    q = rabinMiller.menghasilkanAngkaPrimeryangLuas(ukuranKunci)
    n = p * q

    #langkah kedua, buatlah angka a dimana e menjadi lebih relatif menjadi primer (p-1)*(q-1)
    print('Mengubah e menajdi angka primer yang lebih relatif menjadi (p-1)*(q-1)...')
    while True:
        #tetap mencoba dengan angka acak hingga e mencapai nilai yang benar
        e = random.randrange(2 ** (ukuranKunci - 1), 2 ** (ukuranKunci))
        if kriptomath.gcd(e, (p - 1) * (q - 1)) == 1:
            break

    #langkah ketiga, menghitung d, kebalikan mod dari e
    print('Menghitung d menjadi kebalikan mod dari e...')
    d = kriptomath.menentukanAturanKebalikan(e, (p - 1) * (q - 1))
    kunciPublik = (n, e)
    kunciPrivat = (n, d)

    print('Kunci Public :', kunciPublik)
    print('Kunci Private :', kunciPrivat)

    return (kunciPublik, kunciPrivat)

def buatKunciFile(nama, ukuranKunci):
    #membuat dua file 'x_pubkey.txt' dan 'x_privkey.txt' dimana x menjadi nilai nama
    #dengan n, e and d, e bilangan bulat yang ditulis dengan batasan koma

    #mengecek untuk mencegah penulisan kunci yang terlalu banyak dari kunci file
    if os.path.exists('%s_pubkey.txt' % (nama)) or os.path.exists('%s_privkey.txt' % (nama)):
```

```
        sys.exit('PERHATIAN : bahwa file %s_pubkey.txt atau %s_privkey.txt sudah ada!
Gunakan nama berbeda atau hapus dan ulangi kembali program ini' % (nama, nama))

    kunciPublik, kunciPrivat = hasilkanKunci(ukuranKunci)

    print()
    print('Kunci publik adalah %s dan a %s banyak angka.' % (len(str(kunciPublik[0])),
len(str(kunciPublik[1]))))
    print('Menulis kunci publik pada file %s_pubkey.txt...' % (nama))
    fo = open('%s_pubkey.txt' % (nama), 'w')
    fo.write('%s,%s,%s' % (ukuranKunci, kunciPublik[0], kunciPublik[1]))
    fo.close()

    print()
    print('Kunci private adalah %s dan a %s banyak angka.' % (len(str(kunciPublik[0])),
len(str(kunciPublik[1]))))
    print('Menulis kunci publik pada file %s_privkey.txt...' % (nama))
    fo = open('%s_privkey.txt' % (nama), 'w')
    fo.write('%s,%s,%s' % (ukuranKunci, kunciPrivat[0], kunciPrivat[1]))
    fo.close()

#jika keseluruhan program sudah berjalan dengan benar juga program yang diimport
sudah berjalan dengan semestinya
if __name__ == '__main__':
    main()
```

Output yang dihasilkan :

```
Sedang membuat kunci...
Mengubah p primer...
Mengubah q primer...
Mengubah e menajdi angka primer yang lebih relatif menjadi (p-1)*(q-1)...
Menghitung d menjadi kebalikan mod dari e...
Kunci                                   Public                                  :
(25406994949120356546856595741912553359603430637267624959543674532210254018
7436299744322120668381912637855946707603383437795842944150239429235037679004
4670033834890884732737141791712981288425374295076287420377479808760420128193
8664318623273832140247758232232432830994257653938371435117117403769602012567
```

```
4247195624154527945398610662445031319825666093063099445861916768421457206416
2155188134182539244154160024346228091432168063039239752792548720334991004838
8473288846899034944914263926819038138933002734727985629473062687590759765222
3317542842276124747131085499830940109512988462483095635550081990575391227653
9737800003,
1696544267543113468020172440459455790338087287195167973754756902676837061578
3547207906185010320762220903713211901823715407847470734049854501656774905937
4903778828311435765557300621596483396672744390514244147562293745944291021152
1293024016486372218904937399781784474415201324540969850662102127528354576784
27637)
Kunci Private :
(2540699494912035654685659574191255335960343063726762495954367453221025401807
4362997443221206683819126378559467076033834377958429441502394292350376790044
6700338348908847327371417917129812884253742950762874203774798087604201281938
6643186232738321402477582322324328309942576539383714351171174037696020125674
2471956241545279453986106624450313198256660930630994458619167684214572064162
1551881341825392441541600243462280914321680630392397527925487203349910048388
4732888468990349449142639268190381389330027347279856294730626875907597652223
3317542842276124747131085499830940109512988462483095635550081990575391227653
9737800003,
1604413766046392076714912367413318346914505202022007956792800826057542458896
1092244608263333480359719361115065128462812224275784261710766049372013510001
0234248166152015640975671050305692821752671640257323951322195699647053631667
3329125810006364988074926022283421554081897034369343088083215248214577481803
7361409104878361196316638116542403644229011920277374566298870713547175324098
1440489903506088043491625101609324608997365605537734746018638750346136993904
0717063896227175075016589693819757073877506409607942983081128385913711787883
8750387678407735951453839532331964150786318588866506534313482951041273122668
773817573)

Kunci publik adalah 617 dan a 309 banyak angka.
Menulis kunci publik pada file matius_celcius_sinaga_pubkey.txt...

Kunci private adalah 617 dan a 309 banyak angka.
Menulis kunci publik pada file matius_celcius_sinaga_privkey.txt...
Kunci file sudah dibuat.
```

KRIPTOSISTEM HYBRID

Dalam kehidupan sebenarnya, kerumitan matematika membuat RSA dan enkripsi kunci publik sangat lambat untuk dihitung. Hal ini khususnya bagi server yang membutuhkan pembuatan ratusan bahkan ribuan koneksi terenkripsi dalam satu detik. Sebaliknya, sandi RSA sering digunakan untuk mengenkripsi kunci pada sandi kunci symmetrik. Kunci terenkripsi dikirim pada orang lain, dan kunci tersebut digunakan untuk mengambil pesan terenkripsi menggunakan sandi simetrik. Dengan menggunakan sandi kunci simetrik dan sebuah kunci asimmetrik akan membuat komunikasi benar-benar aman dan disebut sebagai kriptosistem hybrid.

```
##########################################################################
# Sandi RSA                                                              #
# Editor : Matius Celcius Sinaga                                         #
# Author : # http://inventwithpython.com/hacking (BSD Licensed)          #
##########################################################################

import sys

#ukuran blok harus lebih kecil daripada atau sama dengan ukuran kunci
#ukuran blok dalam bytes, ukuran kunci dalam bits
#ada 8 bits dalam 1 byte
UKURAN_BLOK_STANDAR = 128
#maksudnya terdapat 128 bytes
UKURAN_BYTE = 256
#dalam satu byte memiliki 256 nilai yang berbeda

def main():
    #jalankan sebuah test yang mengenkripsi sebuah pesan dalam sebuah file atau mendekrip sebuah pesan dari file
    namafile = 'encrypted_file.txt'
    #file untuk menulis atau untuk membaca
    mode = 'encrypt'
    #tentukan apakah enkripsi atau dekripsi

    if mode == 'encrypt':
```

```
        pesan = """Journalists belong in the gutter because that is where the ruling classes throw their guilty secrets." -Gerald Priestland "The Founding Fathers gave the free press the protection it must have to bare the secrets of government and inform the people." -Hugo Black'''
        namaKunciFilePublik = 'matius_celcius_sinaga_pubkey.txt'
        print('Mengenkripsi dan menulis dalam %s...' % (namafile))
        teksTerenkripsi = enkripsidantulisDalamFile(namafile, namaKunciFilePublik, pesan)

        print('Teks yang telah dienkripsi :')
        print(teksTerenkripsi)

    elif mode == 'decrypt':
        namaKunciFilePrivate = 'matius_celcius_sinaga_privkey.txt'
        print('Membaca dari %s dan mendekripsikan...' % (namafile))
        pesanTerdekripsi = membacadariFiledanDekripsi(namafile, namaKunciFilePrivate)

        print('Teks yang telah dekripsi:')
        print(pesanTerdekripsi)

def menetapkanBlokblokDariTeks (pesan, ukuranBlok=UKURAN_BLOK_STANDAR):
    #mengubah sebuah pesan menjadi sebuah daftar blok bilangan bulat
    #setiap bilangan bulat menghasilkan 128 (ukuran ukuranBlok) karakter

    pesanDalamByte = pesan.encode('ascii')
    #mengubah bilangan bulat menjadi bytes

    blockInts = []
    for mulaiBlok in range(0, len(pesanDalamByte), ukuranBlok):
        #menghitung blok bilangan bulat pada blok pada teks
        blockInt = 0
        for i in range(mulaiBlok, min(mulaiBlok + ukuranBlok, len(pesanDalamByte))):
            blockInt += pesanDalamByte[i] * (UKURAN_BYTE ** (i % ukuranBlok))
        blockInts.append(blockInt)
    return blockInts

def menetapkanTeksDariBlokblok (blockInts, panjangPesan, ukuranBlok=UKURAN_BLOK_STANDAR):
```

```
    #mengubah daftar blok bilangan bulat pada pesan asli
    #pesan asli panjang yang dibutuhkan secara khusus
    #untuk mengubah bilangan bulat blok terakhir
    pesan = []
    for blockInt in blockInts:
        pesanBlok = []
        for i in range (ukuranBlok - 1, -1, -1):
            if len(pesan) + i < panjangPesan:
                #mengubah pesan string menjadi 128
                #sesuai dengan ukuranBlok
                #karakter pada blok bilangan bulat
                angkaASCII = blockInt // (UKURAN_BYTE ** i)
                blockInt = blockInt % (UKURAN_BYTE ** i)
                pesanBlok.insert(0, chr(angkaASCII))
        pesan.extend(pesanBlok)
    return ''.join(pesan)

def enkripsiPesan(pesan, kunci, ukuranBlok=UKURAN_BLOK_STANDAR):
    #mengubah pesan menjadi daftar blok bilangan bulat
    #lalu enkripsi setiap blok bilangan bulat
    #melalui kunci PUBLIK untuk enkripsi
    blokTerenkripsi = []
    n, e = kunci

    for blok in menetapkanBlokblokDariTeks(pesan, ukuranBlok):
        # ciphertext = plaintext ^ e mod n
        blokTerenkripsi.append(pow(blok, e, n))
    return blokTerenkripsi

def dekripsiPesan(blokTerenkripsi, panjangPesan, kunci,
ukuranBlok=UKURAN_BLOK_STANDAR):
    #dekrip daftar blok enkripsi yang ada menjadi pesan asli
    #panjang pesan asli yang dibutuhkan harus sesuai dengan blok akhirnya
    #pastikan agar kunci privatnya untuk melakukan dekripsi
    blokTerdekripsi = []
    n, d = kunci
    for blok in blokTerenkripsi:
        #teks awal = sandi teks ^ d mod n
        blokTerdekripsi.append(pow(blok, d, n))
```

```
    return menetapkanTeksDariBlokblok(blokTerdekripsi, panjangPesan, ukuranBlok)

def membacaKunciFile(namaFileKunci):
    #berikan nama file untuk setiap file yang memiliki kunci publik atau private
    #menghasilkan kunci sebagai berikut (n,e) atau (n,d) nilai tuple
    fo = open(namaFileKunci)
    content = fo.read()
    fo.close()
    ukurankunci, n, EorD = content.split(',')
    return (int(ukurankunci), int(n), int(EorD))

def enkripsidantulisDalamFile(namaFilePesan, namaFileKunci, pesan, ukuranBlok=UKURAN_BLOK_STANDAR):
    #dengan menggunakan kunci dari file kunci, enkripsi seluruh pesan dan simpan menjadi file
    #menghasilkan pesan yang sudah dienkripsi
    ukurankunci, n, e = membacaKunciFile(namaFileKunci)

    #cek bahwa ukuran kunci lebih besar daripada ukuran blok
    if ukurankunci < ukuranBlok * 8:
    #* 8 ubah bytes menjadi bits
        sys.exit('ERROR: Ukuran blok adalah %s bits dan ukuran kunci adalah %s bits. RSA cipher membutuhkan ukuran blok yang sama atau lebih besar daripada ukuran kunci. Begitu juga dengan kenaikan ukuran blok atau menggunakan kunci yang berbeda.' % (ukuranBlok * 8, ukurankunci))

    #mengenkripsi pesan
    blokTerenkripsi = enkripsiPesan(pesan, (n, e), ukuranBlok)

    #menguah nilai lebih besar dari satu jenis nilai lainnya
    for i in range(len(blokTerenkripsi)):
        blokTerenkripsi[i] = str (blokTerenkripsi[i])
    kontenYangTerenkripsi = ','.join(blokTerenkripsi)

    #menuliskan pesan yang sudah terenkripsi pada hasil keluaran file
    kontenYangTerenkripsi = '%s,%s,%s' % (len(pesan), ukuranBlok, kontenYangTerenkripsi)
    fo = open(namaFilePesan, 'w')
    fo.write(kontenYangTerenkripsi)
```

```
    fo.close()
    #juga menghasilkan string yang terenkripsi
    return kontenYangTerenkripsi

def membacadariFiledanDekripsi(namaFilePesan, namaFileKunci):
    #dengan menggunakan kunci pada file kunci, membaca sebuah pesan terenkripsi dari sebuah file lalu mendekripnya
    #menghasilkan pesan terdekripsi dalam bentuk string
    ukurankunci, n, d = membacaKunciFile(namaFileKunci)

    #membaca panjang pesan dan mengenkripsi pesan tersebut dari file
    fo = open(namaFilePesan)
    content = fo.read()
    panjangPesan, ukuranBlok, pesanTerenkripsi = content.split('_')
    panjangPesan = int(panjangPesan)
    ukuranBlok = int(ukuranBlok)

    #cek bahwa ukuran kunci lebih besar dari ukuran blok
    if ukurankunci < ukuranBlok * 8:
    # * 8 diubah dari bytes menjadi bits
        sys.exit('ERROR : Ukuran blok adalah %s bits dan ukuran kunci adalah %s bits. RSA cipher membutuhkan ukuran blok yang sama atau lebih besar daripada ukuran kunci. Apakah anda telah benar-benar memeriksa kunci file dan melakukan enkripsi pada file ?' % (ukuranBlok * 8, ukurankunci))

    #mengubah pesan terenkripsi hingga menjadi lebih besar dari nilai integer
    blokTerenkripsi = []
    for blok in pesanTerenkripsi.split(','):
        blokTerenkripsi.append(int(blok))

    #mendekrip nilai int yang lebih besar
    return dekripsiPesan(blokTerenkripsi, panjangPesan, (n,d), ukuranBlok)

#jika rsaCipher.py sudah berjalan termasuk didalamnya seluruh modul maka panggil fungsi main()
if __name__ == '__main__':
    main()
```

Output yang dihasilkan adalah :

```
Mengenkripsi dan menulis dalam encrypted_file.txt...
Teks yang telah dienkripsi :
262,128,48042021586627980031292744547218542226230265815384306347986391590175716527520397684692522142577300542918785182414469538851750375506276536082841728355478128689374210316862606567451727708923721953215331539780303035368682306381526153541869314750150397842782944849976705071489950892863583808727402057771362071594173629509443182231433829892755559652324768677125783241114055451222511009196635016122312927780387733382396888485528345217038071017310356237368566666881993467151288520139007945651761273814038019789711364489377216497034062409419738482355795895912739789428703338617284674376352188877617933314402740901555193105 8,186769312469860474457197356791547637525380810067567898331071106881271192708504732284251579455184715742826428043047483426219627730961165714048880866712916656804988438381265608198463615810651138364481270784776068439187227524992136922276768881801088036035018620556842454209386977812100609254192440836782228683224508493477224001975407675823521976354298972414611469551157910289006370279875673208778467117958587160016869712850399738820520059711673879092139623204595384237367191905355119748278100196036912852260695336655598544976000600994885837882479111524901079290746444478279730182764081028888837098857371455297542784262678,264578367585126922033914422947493696747124030057911551893510763013199377635000414113178460834852264788883815374356229140622893843749590711141456769185530282934042409853298954222580694534261457177937014375022755537569911686644922730792918544332373165133704844736095608031423262648179740876089982165820053074093923632022984157697962780732385037446415202010063024113256569027912186066925299148949226486426928756195500222740066834895220930860029203650452918919947452260461378120576046440588911180079503073846675627134062567023930640399258572317848297078124059137385432329802091300285650693441993768280612384813308854963 3
```

Untuk melakukan dekripsi terhadap sandi RSA anda hanya mengubah mode menjadi decrypt/dekrip/dekripsi dan menjalankan program seperti di awal.

Ini adalah contoh bagaimana program sandi RSA diimplementasikan, hal ini hanya akan mengenkrip dan dekripsi file teksawal. Sebuah teksawal (bukan maksud untuk membuat ambigu antara antara “teksawal” dengan cara kriptografi) adalah file yang hanya memiliki karakter teks (seperti halnya apa yang dapat anda ketikkan di keyboard). Sebagai contoh, .py yang anda ketikkan pada program python adalah file plaintext. File plainteks adalah tipe yang dibuat menggunakan teks editor seperti Notepad pada

Windows, TextMate pada OSX, atau Gedit pada Ubuntu. Secara khusus, file plainteks adalah file yang hanya mengandung nilai ASCII.

File seperti gambar, video, program exe, file atau bahasa prosessor disebut file binary. (file kata prosessor adalah file binary karena teks memiliki font, warna, dan ukuran informasi yang disatukan dengan teks).

Sandi RSA bukan hanya menunjukkan bagaimana enkripsi bekerja namun juga bagaimana pengelanan digital untuk mengetahui apakah orang yang anda maksud/tuju adalah orang yang benar. Ingatlah bahwasanya sandi RSA memiliki kunci publik dan privat dan setiap string akan di enkripsi dengan satu kunci yang dihasilkan oleh teks sandi dan hanya dapat di dekripsi dengan kunci sandu yang lain. Jika anda mengenkripsi dengan kunci publik maka hanya pemilik kunci privatlah yang dapat mendekripsi sandi awal tersebut.

Dalam Kriptografi terdapat blok yang dapat mengubah panjang dari sebuah bits di dalam sandi RSA ada blok yang dihasilkan dengan sebuah integer atau bilangan bulat yang ditentukan menjadi ukuran blok hingga mencapai 128 byte atau 1024 bits, ingat dalam perhitungannya 8 bits berarti 1 byte.

PROGRAM PYPERCLIP

```
# Pyperclip v1.4
# A cross-platform clipboard module for Python. (only handles plain text for now)
# By Al Sweigart al@coffeeghost.net
# Usage:
#   import pyperclip
#   pyperclip.copy('The text to be copied to the clipboard.')
#   spam = pyperclip.paste()
# On Mac, this module makes use of the pbcopy and pbpaste commands, which should
come with the os.
# On Linux, this module makes use of the xclip command, which should come with the
os. Otherwise run "sudo apt-get install xclip"
# Copyright (c) 2010, Albert Sweigart
# All rights reserved.
#
# BSD-style license:
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#     * Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#     * Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#     * Neither the name of the pyperclip nor the
#       names of its contributors may be used to endorse or promote products
#       derived from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY Albert Sweigart "AS IS" AND ANY
# EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED
# WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
# DISCLAIMED. IN NO EVENT SHALL Albert Sweigart BE LIABLE FOR ANY
# DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
# (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES;
```

```
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND
# ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS
# SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Change Log:
# 1.2 Use the platform module to help determine OS.
# 1.3 Changed ctypes.windll.user32.OpenClipboard(None) to
ctypes.windll.user32.OpenClipboard(0), after some people ran into some TypeError

import platform, os

def winGetClipboard():
    ctypes.windll.user32.OpenClipboard(0)
    pcontents = ctypes.windll.user32.GetClipboardData(1) # 1 is CF_TEXT
    data = ctypes.c_char_p(pcontents).value
    #ctypes.windll.kernel32.GlobalUnlock(pcontents)
    ctypes.windll.user32.CloseClipboard()
    return data

def winSetClipboard(text):
    text = str(text)
    GMEM_DDESHARE = 0x2000
    ctypes.windll.user32.OpenClipboard(0)
    ctypes.windll.user32.EmptyClipboard()
    try:
        # works on Python 2 (bytes() only takes one argument)
        hCd = ctypes.windll.kernel32.GlobalAlloc(GMEM_DDESHARE,
len(bytes(text))+1)
    except TypeError:
        # works on Python 3 (bytes() requires an encoding)
        hCd = ctypes.windll.kernel32.GlobalAlloc(GMEM_DDESHARE, len(bytes(text,
'ascii'))+1)
    pchData = ctypes.windll.kernel32.GlobalLock(hCd)
    try:
        # works on Python 2 (bytes() only takes one argument)
```

```
        ctypes.cdll.msvcrt.strcpy(ctypes.c_char_p(pchData), bytes(text))
    except TypeError:
        # works on Python 3 (bytes() requires an encoding)
        ctypes.cdll.msvcrt.strcpy(ctypes.c_char_p(pchData), bytes(text, 'ascii'))
    ctypes.windll.kernel32.GlobalUnlock(hCd)
    ctypes.windll.user32.SetClipboardData(1, hCd)
    ctypes.windll.user32.CloseClipboard()

def macSetClipboard(text):
    text = str(text)
    outf = os.popen('pbcopy', 'w')
    outf.write(text)
    outf.close()

def macGetClipboard():
    outf = os.popen('pbpaste', 'r')
    content = outf.read()
    outf.close()
    return content

def gtkGetClipboard():
    return gtk.Clipboard().wait_for_text()

def gtkSetClipboard(text):
    global cb
    text = str(text)
    cb = gtk.Clipboard()
    cb.set_text(text)
    cb.store()

def qtGetClipboard():
    return str(cb.text())

def qtSetClipboard(text):
    text = str(text)
    cb.setText(text)

def xclipSetClipboard(text):
    text = str(text)
```

```
    outf = os.popen('xclip -selection c', 'w')
    outf.write(text)
    outf.close()

def xclipGetClipboard():
    outf = os.popen('xclip -selection c -o', 'r')
    content = outf.read()
    outf.close()
    return content

def xselSetClipboard(text):
    text = str(text)
    outf = os.popen('xsel -i', 'w')
    outf.write(text)
    outf.close()

def xselGetClipboard():
    outf = os.popen('xsel -o', 'r')
    content = outf.read()
    outf.close()
    return content


if os.name == 'nt' or platform.system() == 'Windows':
    import ctypes
    getcb = winGetClipboard
    setcb = winSetClipboard
elif os.name == 'mac' or platform.system() == 'Darwin':
    getcb = macGetClipboard
    setcb = macSetClipboard
elif os.name == 'posix' or platform.system() == 'Linux':
    xclipExists = os.system('which xclip') == 0
    if xclipExists:
        getcb = xclipGetClipboard
        setcb = xclipSetClipboard
    else:
        xselExists = os.system('which xsel') == 0
        if xselExists:
            getcb = xselGetClipboard
```

```
            setcb = xselSetClipboard
        try:
            import gtk
            getcb = gtkGetClipboard
            setcb = gtkSetClipboard
        except Exception:
            try:
                import PyQt4.QtCore
                import PyQt4.QtGui
                app = PyQt4.QApplication([])
                cb = PyQt4.QtGui.QApplication.clipboard()
                getcb = qtGetClipboard
                setcb = qtSetClipboard
            except:
                raise Exception('Pyperclip requires the gtk or PyQt4 module installed, or
the xclip command.')
copy = setcb
paste = getcb
```

'Stay Hungry, Stay Foolish' atau dalam bahasa Indonesia berarti 'Tetap Lapar, Tetap Bodoh', banyak orang berfikir bahwa hal ini pertama kalinya diungkapkan oleh Steve Jobs namun tahukah anda bahwa sebenarnya kalimat tersebut pertama sekali dicetak oleh majalah Steward Brand dan diedit oleh Kevin Kelly, sebuah pesan perpisahan di belakang sampul majalah tersebut yang dimana dimaksudkan bahwa kita harus pahami ketidakpentingan kita sendiri, gunakan kesadaran dengan rendah hati dan haus, akan seolah-olah laparnya pengetahuan untuk merangkul masa depan. Anda adalah kelompok orang-orang yang benar-benar kelaparan hingga sangat benar-benar kelaparan untuk memiliki buku ini namun kelaparan yang anda mililki tak bisa menjadi alasan ilmu yang ada hanya untuk anda sendiri, laparlah akan ilmu pengetahuan selain dari buku ini namun tetap berbagi hingga menjadi habis dan bodoh, sampai akhir hayat menjemput lalu Allah berkata, sudah selesai.

Amin.

DAFTAR PUSTAKA

"6.4 UNITED STATES CRYPTOGRAPHY EXPORT/IMPORT LAWS". RSA Laboratories.

"6.5.1 WHAT ARE THE CRYPTOGRAPHIC POLICIES OF SOME COUNTRIES?". RSA Laboratories.

a b van Rossum, Guido (1993). "An Introduction to Python for UNIX/C Programmers". Proceedings of the NLUUG najaarsconferentie (Dutch UNIX users group). even though the design of C is far from ideal, its influence on Python is considerable.

a b c Diffie, Whitfield; Hellman, Martin (November 1976). "New Directions in Cryptography" (pdf). IEEE Transactions on Information Theory. IT-22: 644-654

a b "Classes". The Python Tutorial. Python Software Foundation. Diakses tanggal 20 February 2012. It is a mixture of the class mechanisms found in C++ and Modula-3

a b Ibrahim A. Al-Kadi (April 1992), "The origins of cryptology: The Arab contributions", Cryptologia 16 (2): 97-126

a b Schneier, Bruce (1996). Applied Cryptography (2nd ed.). Wiley. ISBN 0-471-11709-9.

a b Singh, Simon (2000). The Code Book. New York: Anchor Books. pp. 14-20. ISBN 9780385495325.

a b Wayner, Peter (24 December 1997). "British Document Outlines Early Encryption Discovery". New York Times.

a b c d David Kahn, The Codebreakers, 1967, ISBN 0-684-83130-9.

a b c Levy, Steven (2001). Crypto: How the Code Rebels Beat the Government-Saving Privacy in the Digital Age. Penguin Books. p. 56. ISBN 0-14-024432-8. OCLC 244148644 48066852 48846639.

a b c d e f g Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A. Handbook of Applied Cryptography. ISBN 0-8493-8523-7.

Babai, László (1985). "Trading group theory for randomness". Proceedings of the Seventeenth Annual Symposium on the Theory of Computing (Association for Computing Machinery).

Bellare, Mihir; Rogaway, Phillip (21 September 2005). "Introduction". Introduction to Modern Cryptography. p. 10.

"Bernstein v USDOJ". Electronic Privacy Information Center. United States Court of Appeals for the Ninth Circuit. 6 May 1999.

Biham, E.; Shamir, A. (1991). "Differential cryptanalysis of DES-like cryptosystems"(PDF). Journal of Cryptology (Springer-Verlag) 4 (1): 3-72.

Bini, Ola (2007). Practical JRuby on Rails Web 2.0 Projects: bringing Ruby on Rails to the Java platform. Berkeley: APress. p. 3. ISBN 978-1-59059-881-8.

Blakley, G. (June 1979). "Safeguarding cryptographic keys". Proceedings of AFIPS 197948: 313-317.Templat:Cite Journal

Blaze, Matt; Diffie, Whitefield; Rivest, Ronald L.; Schneier, Bruce; Shimomura, Tsutomu; Thompson, Eric; Wiener, Michael (January 1996). "Minimal key lengths for symmetric ciphers to provide adequate commercial security". Fortify. Diakses tanggal 26 March2015.

Brands, S. (1994). "Untraceable Off-line Cash in Wallets with Observers". Advances in Cryptology-Proceedings of CRYPTO (Springer-Verlag).[pranala nonaktif]

"Case Closed on Zimmermann PGP Investigation". IEEE Computer Society's Technical Committee on Security and Privacy. 14 February 1996.

Cocks, Clifford (20 November 1973). "A Note on 'Non-Secret Encryption'" (PDF). CESG Research Report.

Coppersmith, D. (May 1994). "The Data Encryption Standard (DES) and its strength against attacks" (PDF). IBM Journal of Research and Development 38 (3): 243. doi:10.1147/rd.383.0243. Diakses tanggal 26 March 2015.

"Cryptology (definition)". Merriam-Webster's Collegiate Dictionary (11th ed.). Merriam-Webster.

Diffie, Whitfield; Hellman, Martin (8 June 1976). "Multi-user cryptographic techniques". AFIPS Proceedings 45: 109-112.

"DUAL-USE LIST - CATEGORY 5 - PART 2 - "INFORMATION SECURITY"" (DOC). Wassenaar Arrangement.

Goldwasser, S.; Micali, S.; Rackoff, C. (1989). "The Knowledge Complexity of Interactive Proof Systems". SIAM Journal on Computing 18 (1): 186-208.

Ferguson, Niels (15 August 2001). "Censorship in action: why I don't publish my HDCP results". Diarsipkan Desember 1, 2001 di Wayback Machine

"FIPS PUB 197: The official Advanced Encryption Standard" (PDF). Computer Security Resource Center. National Institute of Standards and Technology.

Golen, Pawel (19 July 2002). "SSH". WindowSecurity.

Hakim, Joy (1995). A History of Us: War, Peace and all that Jazz. New York: Oxford University Press. ISBN 0-19-509514-6.

Ingold, John (January 4, 2012). "Password case reframes Fifth Amendment rights in context of digital world". The Denver Post.

James Gannon, Stealing Secrets, Telling Lies: How Spies and Codebreakers Helped Shape the Twentieth Century, Washington, D.C., Brassey's, 2001, ISBN 1-57488-367-4.

Junod, Pascal (2001). "On the Complexity of Matsui's Attack" (PDF). Selected Areas in Cryptography.

Kahn, David (Fall 1979). "Cryptology Goes Public". Foreign Affairs 58 (1): 153.

Kuchling, A. M. "Functional Programming HOWTO". Python v2.7.2 documentation. Python Software Foundation.

Kuchling, Andrew M. (22 December 2006). "Interview with Guido van Rossum (July 1998)". amk.ca.

Liddell and Scott's Greek-English Lexicon. Oxford University Press. (1984)

Leyden, John (2011-07-13). "US court test for rights not to hand over crypto keys". Theregister.co.uk.

Leyden, John (13 July 2011). "US court test for rights not to hand over crypto keys". The Register.

"NCUA letter to credit unions" (PDF). National Credit Union Administration. July 2004.

"Notices". Federal Register 72 (212). 2 November 2007.Diarsipkan 28 Februari 2008 di Wayback Machine

"NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition". Tech Beat. National Institute of Standards and Technology. October 2, 2012.

Oded Goldreich, Foundations of Cryptography, Volume 1: Basic Tools, Cambridge University Press, 2001, ISBN 0-521-79172-3

"ORDER GRANTING APPLICATION UNDER THE ALL WRITS ACT REQUIRING DEFENDANT FRICOSU TO ASSIST IN THE EXECUTION OF PREVIOUSLY ISSUED SEARCH WARRANTS" (PDF). United States District Court for the District of Colorado.

"Overview per country". Crypto Law Survey. February 2013.

Previously released as an MIT "Technical Memo" in April 1977, and published in Martin Gardner's Scientific American Mathematical recreations column

"RFC 2440 - Open PGP Message Format". Internet Engineering Task Force. November 1998.

Rivest, Ronald L.; Shamir, A.; Adleman, L. (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM (Association for Computing Machinery) 21 (2): 120-126.Diarsipkan November 16, 2001 di Wayback Machine

Rivest, Ronald L. (1990). "Cryptology". Di J. Van Leeuwen. Handbook of Theoretical Computer Science 1. Elsevier.

Rosenoer, Jonathan (1995). "CRYPTOGRAPHY & SPEECH". CyberLaw. Diarsipkan Desember 1, 2005 di Wayback Machine

Schemenauer, Neil; Peters, Tim; Hetland, Magnus Lie (18 May 2001). "PEP 255 - Simple Generators". Python Enhancement Proposals. Python Software Foundation.

Schneier, Bruce (15 June 2000). "The Data Encryption Standard (DES)". Crypto-Gram.

Schrödel, Tobias (October 2008). "Breaking Short Vigenère Ciphers". Cryptologia 32 (4): 334-337. doi:10.1080/01611190802336097.

Shannon, Claude; Weaver, Warren (1963). The Mathematical Theory of Communication. University of Illinois Press. ISBN 0-252-72548-4.

Simionato, Michele. "The Python 2.3 Method Resolution Order". Python Software Foundation. The C3 method itself has nothing to do with Python, since it was invented by people working on Dylan and it is described in a paper intended for lispers

Singh, Simon (1999). The Code Book. Doubleday. pp. 279-292.

Song, Dawn; Wagner, David A.; Tian, Xuqing (2001). "Timing Analysis of Keystrokes and Timing Attacks on SSH" (PDF). Tenth USENIX Security Symposium.

Smith, Kevin D.; Jewett, Jim J.; Montanaro, Skip; Baxter, Anthony (2 September 2004). "PEP 318 - Decorators for Functions and Methods". Python Enhancement Proposals. Python Software Foundation. Diakses tanggal 24 February 2012.

Sweigart, Al. Hacking Secret Ciphers with Python. 2013. United State. ISBN 978-1482614374. 1st Editio

"The Digital Millennium Copyright Act of 1998" (PDF). United States Copyright Office.

"UK Data Encryption Disclosure Law Takes Effect". Pcworld.com. 2007-10-01.

"UK Data Encryption Disclosure Law Takes Effect". PC World. 1 October 2007.

V. V. IÂ˜shchenko (2002). "Cryptography: an introduction". AMS Bookstore. p.6. ISBN 0-8218-2986-6

Williams, Christopher (11 August 2009). "Two convicted for refusal to decrypt data". The Register.

Williams, Christopher (24 November 2009). "UK jails schizophrenic for refusal to decrypt files". The Register.

http://google.com/watercolour-splatter-in-rainbow-colours_1048-6320

http://staff.neu.edu.tr/~fahri/cryptography.html

http://wikipedia.com/Kriptografi

http://wikipedia.com/python

PENULIS bernama lengkap ***Matius Celcius Sinaga*** akrab disapa sebagai *Naga/yuz/Gan/Stefenson.* Lahir di kota Tarutung tepatnya pada tanggal 27 September 1995 adalah juga seorang mahasiswa yang baru saja menyelesaikan pendidikan D-3 (Diploma Tiga)nya di Universitas Sumatera Utara jurusan Metrologi dan Instrumentasi, sangat menyukai hal yang berhubungan dengan komputer, khususnya dalam bidang pemograman maupun jaringan, memiliki pengalaman dan pendidikan di bidang komputer dan seringnya belajar dengan cara otodidak. Buku **Kriptografi Python** masih banyak memiliki kekurangan baik secara tatabahasa dan beberapa detail lainnya, namun tetaplah buku ini ditulis dengan kesungguhan hati untuk memberikan referensi tambahan juga menyadarkan teman-teman para pelajar dan akademis bahwa kita harus mendobrak keterbatasan dan tembok-tembok penghalang untuk tetap berkembang di setiap waktu. Topik kriptografi dipilih bukan tanpa alasan, sebagai seorang yang juga gemar membaca artikel internet yang menjuru pada pemograman pada umunya, penulis melihat bahwa masih kurang banyaknya referensi mengenai Kriptografi dalam buku berbahasa Indonesia, buku yang menjelaskan secara gamblang dengan benar-benar jelas menggambarkannya (dengan memberi source code) penulis menemukan dalam beberapa buku kriptografi dijelaskan lebih kepada teori kriptografi itu sendiri.