IMPLEMENTASI ENKRIPSI DATA DENGAN ALGORITMA RIVEST-SHAMIR-ADLEMAN (RSA)

Adventus Wijaya Lumbantobing – NIM : 13505112

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if115112@students.if.itb.ac.id

Abstrak

Sebelum *public key* dalam kriptografi ditemukan, sistem *chiper* menggunakan perjanjian antar kedua belah pihak untuk melakukan enkripsi dan dekripsi pesan. Dalam kriptografi saat ini terdapat individu yang mempublikasikan sebuah kunci enkripsi (*public key*) yang bias digunakan siapa saja yang ingin mengirim pesan-pesan pribadi. Untuk mendekripsi pesan-pesan tersebut, individu tersebut mempunyai kunci yang berbeda dengan *public key* namun masih berelasi secara matematika, kunci ini disebut *private key*. Tingkat keamanan sistem tergantung pada kerahasiaan dan kesulitan publik untuk mengetahui *private key* tersebut. Dalam prakteknya tingkat keamanan sistem tergantung pada adanya kemungkinan cara lain untuk memperoleh kedua pasangan kuci tersebut, yaitu *public key* dan *private key*.

Penggunaan *public key* dalam kriptografi pertama kali menggunakan teorema yang sangat kompleks dan sangat sulit untuk dipecahkan. Metode yang lebih praktis diajukan oleh Ron Rivest, Adi Shamir, dan Leonard Adleman (RSA) dari *Massachusetts Institut of Technology (MIT)* pada tahun 1977.

Pertama-tama individu memilih dua bilangan prima dan hasil kali dari kedua bilangan tersebut. Hasil kali dari bilangan-bilangan prima tersebut digunakan dalam proses enkripsi dan juga dalam proses dekripsi yang memerlukan pengolahan dalam bilangan prima. Untuk memecahkan kode ini harus diketahui hasil faktorisasi dari hasil kali bilangan prima yang sebelumnya. Dan hal ini sangat sulit jika bilangan tersebut merupakan bilangan yang besar dan bahkan memerlukan waktu yang sangat lama dengan menggunakan *supercomputer* sekalipun.

1. Pendahuluan

Metode penggunaan *public key* seperti ini pertama kali ditemukan pada tahun 1973 oleh Clifford Cocks. RSA merupakan sistem enkripsi dan autentifukasi yang merupakan implementasi dari algoritma yang dikembangkan oleh Ron Rivest, Adi Shamir, dan Leonard Adleman dari Massachusetts Institut of Technology (MIT) pada tahun 1977. Nama RSA sendiri diambil dari inisial nama ketiga orang tersebut.

Algoritma tersebut dipatenkan oleh *Massachusetts Institute of Technology* pada tahun 1983 di Amerika Serikat sebagai U.S. Patent 4405829. Paten tersebut berlaku hingga 21 September 2000. Semenjak algoritma RSA dipublikasikan sebagai aplikasi paten, regulasi di sebagian besar negara-negara lain tidak memungkinkan penggunaan paten.[3]

Algoritma RSA menjadi enkripsi dan autentifikasi yang paling banyak digunakan dan juga telah menjadi bagian dari *web browsers* dari Microsoft dan Netscape. Algoritma ini juga diimplementasikan dalam aplikasi *Lotus Notes*, *Intuit's Quicken*, dan berbagai produk lainnya.

Sistem enkripsi ini data ini dimiliki dan dikembangkan oleh RSA Security. Perusahaan tersebut telah melisensikan teknologi algoritma tersebut dan juga menjual kakas-kakas yang dikembangkan dengan algoritma tersebut. Teknologi ini telah menjadi standar dalam sistem keamanan dalam web, jaringan internet, dan komputerisasi.

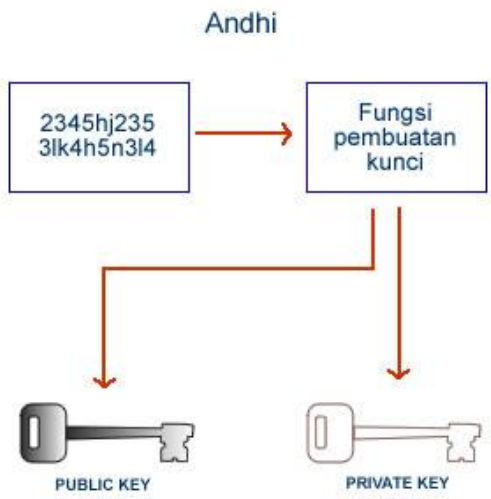
2. Operasional RSA

RSA merupakan metode dalam kriptografi yang menggunakan public key. RSA terdiri dari dua bagian utama yaitu :

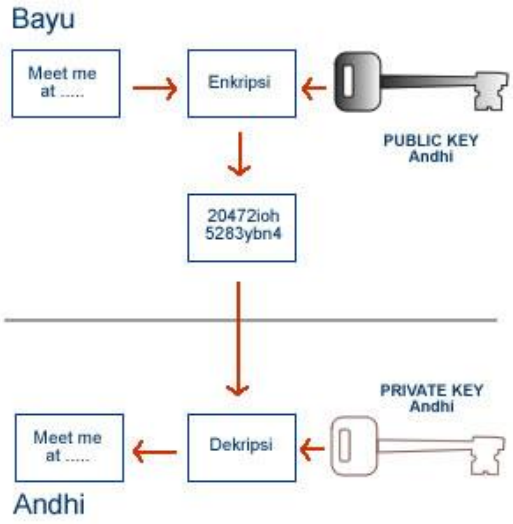
- *public key* (untuk enkripsi)
- *private key* (untuk dekripsi).

Bagian *public key* dapat dipublikasikan ke khlayak umum sehingga publik dapat melakukan enkripsi data sedangkan bagian *private key* tidak dipublikasikan dan tetap dijaga rahasianya untuk melakukan dekripsi data. Pesan yang dienkripsi dengan *public key* tertentu hanya dapat didekripsi dengan *private key* yang bersesuaian.

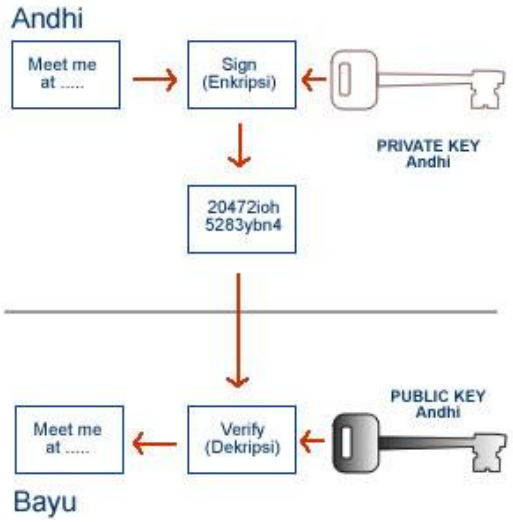
Secara garis besar, algoritma RSA melibatkan perkalian dua bilangan prima yang besar dan dengan tambahan operasi matematika lainnya menghasilkan kedua buah set kunci seperti di atas. Jika kunci sudah disebarkan bilangan prima yang pertama tidak lagi penting dan dapat diabaikan.[7]



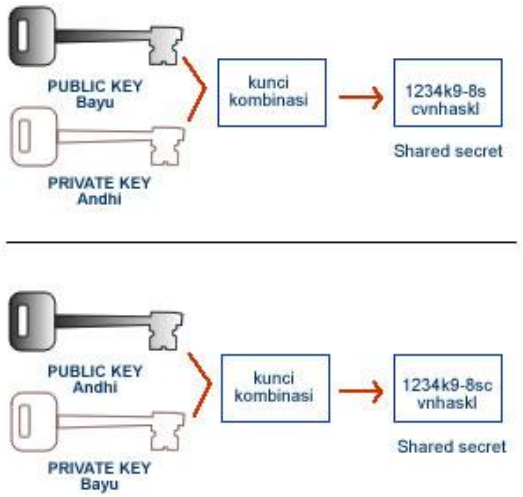
Pembuatan kunci dilakukan dengan pemilihan bilangan acak yang besar lalu dibuat *public key* (dipublikasikan untuk umum) dan *private key* (dirahasiakan).



Setelah publik mendapatkan *public key* yang telah dipublikasikan maka pihak tersebut dapat melakukan enkripsi pesan kepada tujuannya. Penerima pesan akan mendekripsi pesan tersebut dengan *private key* miliknya.



Dengan menggunakan *private key* untuk mengenkripsi *signature* maka siapa saja dapat melakukan dekripsi dengan *public key*. Dengan demikian publik dapat mengetahui siapa pengirim pesan yang ia terima. Validasi pesan tergantung pada sistem keamanan dalam *private key*.



Dengan menggabungkan *private key* dengan *public key* dari orang lain maka kedua pihak dapat membagi pesan rahasia yang hanya diketahui oleh kedua pihak. [5]

Proses enkripsi dan dekripsi dalam penyampaian pesan seperti di atas di gambarkan dalam tabel berikut :

Aksi	Pemilik Kunci	Kunci
Kirim pesan terenkripsi	Penerima	Public key
Kirim *signature* terenkripsi	Pengirim	Private key
Dekripsi pesan	Penerima	Private key
Dekripsi dan enkripsi *signature*	Pengirim	Public key

Berikut merupakan langkah-langkah pembuatan *public key* dan *private key* :

1. Pemilihan dua bilangan prima secara acak misalnya bilangan p dan q dan $p \neq q$.
2. Hitung $n = p\ q$. Besaran n tidak dirahasiakan.
3. Hitung $\varphi(n) = (p\text{-}1)\ (q\text{-}1)$.
4. Pilih bilangan bulat e (*integer*) antara satu dan $\varphi(n)$ $(1 < e < \varphi(n))$ yang juga merupakan relatif prima terhadap $\varphi(n)$ (e dan $\varphi(n)$ tidak memiliki factor lain selain 1). Bilangan e merupakan eksponen dari public key.
5. Hitung d hingga $d\ e \equiv 1 \pmod{\varphi(n)}$. d merupakan eksponen dari private key. Atau dengan cara lain $d\ e = 1 + k\ \varphi(n)$ untuk k bilangan bulat.

Bilangan prima dapat diuji probabilitasnya menggunakan teorema *Fermat's little :*

$$a^{(n-1)} \bmod n = 1$$

Jika n adalah bilangan prima dan dilakukan pengujian dengan beberapa nilai a, kemungkinan besar n adalah bilangan prima.

Terkadang hasil dari persamaan ini memang benar untuk nilai a meskipun n bukanla bilangan prima. Ada cara lain yaitu dengan *Carnichael numbers* yang benar untuk semua nilai a kecuali untuk nilai b yang merupakan faktor dari n.

Berdasarkan teorema *Fermat's Little* dan *Camichael numbers* dikembangkan pengetesan bilangan prima yaitu *Solovay-Strassen probabilistic primality test* :

- Pertama pemilihan nilai a dengan nilai antara 1 sampai n-1. Tes ini untuk melihat apakah b relatif prima terhadap n, jika tidak maka sudah jelas n bukan bilangan prima.
- Sebuah fungsi untuk b dan n dilambangkan $J(b,n)$ disebut simbol Jacobi. Persamaannya adalah sebagai berikut:

$$J(b,n) = b^{((n-1)/2)}$$

- Nilai $J(b,n)$ selalu bernilai 1 atau -1.
- Jika b adalah bilangan genap dan n bilangan ganjil, maka

$$J(b,n) = J(b/2,n)((-1)^{(n \times n-1)/8)})$$

Jika sebaliknya, maka persamaannya menjadi:

$$J(b,n) = J(n \bmod b/2,b)((-1)^{((b-1) \times (n-1)/4)})$$

Pemilihan bilangan e yang umum adalah :

$$e = 2^{16} + 1 = 65537.$$

Beberapa aplikasi menggunakan nilai *e* yang kecil misalnya 3, 5, 35 dengan tujuan agar proses enkripsi dapat berlangsung lebih cepat namun hal ini tentu mengandung risiko yang lebih besar.

Public key terdiri atas :

- modulus *n*
- bilangan eksponen publik *e* (sering juga disebut eksponen enkripsi).

Private key terdiri atas:

- modulus *n*
- bilangan eksponen *d* (sering juga disebut eksponen dekripsi), yang harus dijaga kerahasiaannya.

Agar lebih efisien bentuk dari *private key* dapat dituliskan:

- *p* dan *q*, bilangan prima.
- *d mod (p-1)* dan *d mod (q-1)* (ditulis dengan *dmp1* dan *dmq1*).
- *(1/q) mod p* (ditulis dengan *iqmp*).

Dalam bentuk seperti ini proses dekripsi dan signing lebih cepat namun kurang aman karena adanya *side channel attacks*. Jika $y = x^e \bmod n$ dan sistem akan mendekripsikan persamaan tersebut. Sistem akan mengomputasi $y^d \pmod p$ dan $y^d \pmod q$ yang menghasilkan nilai *z*. Maka nilai gcd(*z-x*,*n*) akan menghasilkan nilai *p* dan *q*. Oleh karena itu semua bagian dari *private key* harus dijaga kerahasiaannya.

Nilai *p* dan *q* sangat sensitif karena kedua bilangan tersebut merupakan faktorial dari *n*, dan mengakibatkan perhitungan dari *d* dapat menghasilkan *e*. Jika *p* dan *q* tidak disimpan dalam bentuk CRT dari *private key*, maka *p* dan *q* telah terhapus bersama nilai-nilai lain dari proses pembangkitan kunci.

Proses Enkripsi Pesan

Pertama-tama pengirim mempublikasikan *public key* (*q* dan *e*) kepada publik sedangkan *private key* tetap dijaga rahasianya.

Misalkan publik akan megirim pesan *p*, maka *p* diubah terlebih dahulu menjadi angka *q* ($q < n$) dengan menggunakan protokol yang telah disepakati sebelumnya (dikenal dengan *padding scheme*).

Dengan demikian, publik akan memiliki *q* dan mengetahui nilai *n* dan *e*. Lalu publik akan menghitung *chipertext c* yang bersesuaian dengan *q* dengan persamaan :

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

Perhitungan tersebut dapat diselesaikan dengan cepat menggunakan metode *exponentiation by squaring*. Kemudian publik mengirim *c* kepada pengirim.

Proses Dekripsi Pesan

Pengirim akan mengembalikan nilai *q* dari *c* dengan menggunakan *private key d* dengan prosedur sebagai berikut :

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

Langkah-langkah dekripsi tersebut sebagai berikut :

$$c \equiv (q^e)^d \equiv q^{ed} \pmod n$$

Karena

$$ed \equiv 1 \pmod{a - 1} \text{ dan } ed \equiv 1 \pmod{b - 1}$$

menurut teorema Fermat's little :

$$q^{ed} \equiv q \pmod a \text{ dan } q^{ed} \equiv q \pmod b.$$

Karena *a* dan *b* bilangan prima, maka dengan menerapkan teorema *Chinese Remainder* pada kedua persamaan ini diperoleh :

$$q^{ed} \equiv q \pmod{ab}$$

Dengan demikian,

$$c^d \equiv q \pmod n$$

atau

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

Jika nilai q diperoleh maka pengirim dapat megubahnya menjadi pesan yang sebenarnya yaitu *p*.

Contoh Soal

Misalkan nilai $p = 19$ dan nilai $q = 23$.

Maka nilai modulus n :

$$n = p \times q$$
$$= 19 \times 23$$
$$= 437$$

Kemudian menentukan nilai $\varphi(n)$:

$$\varphi(n) = \varphi(p) \times \varphi(q)$$
$$= (p - 1) \times (q - 1)$$
$$= (19 - 1) \times (23 - 1)$$
$$= 18 \times 22$$
$$= 396$$

Lalu pemilihan nilai e dan penentuan nilai d :

$$(e \times d) \bmod \varphi(n) = 1$$

$e = 13$
$d = 61$

$$(e \times d) \bmod 396 = (13 \times 61) \bmod 396$$
$$= 793 - (2 \times 396)$$
$$= 793 - 792$$
$$= 1$$

Maka public key adalah ($n = 437$, $e = 13$) maka fungsi pada enkripsi adalah :

$$c = q^e \bmod n$$
$$= q^{13} \pmod{437}$$

Sedangkan private key ($n = 437$, $d = 61$) maka fungsi dekripsinya adalah :

$$q = c^d \bmod n$$
$$= c^{61} \pmod{437}$$

Sebagai contoh jika seseorang ingin mengirim pesan yang sudah diubah ke dalam angka, misalnya dengan nilai $q = 77$.

Proses penghitungan enkripsi yang terjadi adalah sebagai berikut :

$$c = 77^{13} \pmod{437}$$
$$= 248$$

Untuk dekripsi $c = 248$, jika pesan akan diterjemahkan oleh penerima pesan maka penghitungan dekripsi untuk menentukan nilai q:

$$q = 248^{61} \pmod{437}$$
$$= 77$$

Implementasi RSA

RSA dapat diimplementasikan secara hardware dan software, dimana standar implementasi menggunakan PKCS#1. *Public key e* harus terlindungi secara baik sehingga pemilihan 3,17 dan 65537 sering digunakan. Rekomendasi PKCS#1 : 3 (011_2),17 (10001_2) atau 65537 (10000000000000001_2) karena ketiga bilangan tersebut memiliki bit 1 sebanyak 2 buah, sehingga diperlukan hanya 17 perkalian untuk operasi eksponensiasi, akibatnya operasi dekripsi menjadi lebih cepat.

Metoda penghitungan biner untuk $q^e \bmod n$, integer q,e dan n menggunakan algoritma modular exponential, yaitu eksponensial yang disusun dari operasi perkalian modular dan pangkat 2 modular. Berikut adalah algoritma tersebut :

1. a :=1;
2. b :=q;
3. for i=0 to h-2
   a. if ei=1 then a:=$a.b$ mod n
   b. b :=$b.b$ mod n
4. if eh-1=1 then a:=$a.b$ mod n
5. return a

Tiap bit dari *e* di*scan* dari bit 0 LSB hingga h-1 MSB, hal ini dilakukan dalam loop, jika $e_i = 1$ maka dilakukan perkalian modular $a.b$ mod n dan kemudian diteruskan dengan $b.b \bmod n$. Algoritma ini membuat algoritma model *Fermat F4* ($2^{16}+1$) atau 65537, melakukan 17 perkalian , yaitu 15 kali menghitung $b.b \bmod n$ dan 1 kali $a.b \bmod n$ dalam loop 3 dan kemudian 1 kali $a.b \bmod n$ pada langkah 4.[2]

Padding Schemes

Padding scheme adalah protocol yang telah disepakati bersama untukmengubah data atau pesan ke dalam bentuk numerik.

Padding Scheme harus dibuat dengan hati-hati untuk mencegah timbulnya masalah dalam pesan yang telah diubah dalam angka (nilai q).

RSA yang tidak menggunakan *padding scheme* akan mengalami beberapa permasalahan misalnya :

- Jika kita ambil contoh sederhana dari penampilan ASCII dari *p* dan menggabungkan bit-bit secara bersamaan akan menghasilkan *n*, kemudian pesan yang berisi ASCII tunggal karakter NUL (nilai numeris 0) akan menghasilkan $q = 0$, yang akan menghasilkan *ciphertext* 0 untuk setiap nilai *e* dan *n* yang digunakan.
  Sama halnya dengan karakter ASCII tunggal SOH (nilai numeris 1) akan selalu menghasilkan *chiphertext* 1.

- Untuk sistem yang menggunakan nilai *e* yang kecil (misalnya 3) seluruh karakter tunggal ASCII pada pesan akan disandikan menggunakan skema yang tidak aman. Hal ini disebabkan nilai terbesar *q* adalah nilai 255, dan 255^3 menghasilkan nilai yang lebih kecil dari modulus yang seharusnya.
  Maka dengan pola dasar dari *ciphertext* dalam kondosi tersebut proses dekripsi akan menjadi lebih sederhana tanpa perlu menggunakan modulus *n*.

Validasi Pesan

Misalkan A menggunakan *public key* milik B untuk mengirim pesan kepada B. Dalam pesan tersebut A dapat mengidentitaskan dirinya sebagai A namun permasalahannya B tidak dapat mengidentifikasi bahwa pesan tersebut berasal dari A. Hal ini dikarenakan publik memiliki *public key* B dan siapa saja bisa mengirimkan pesan yang telah dienkripsi pada B. Jadi, untuk menjaga orisinalitas pesan (identitas pengirim) RSA juga dapat diterapkan untuk menvalidasi pesan yang diterima.

Metode yang digunakan sama seperti halnya pada enkripsi dan dekripsi pesan. Untuk menyertakan identitas maka pengirim pesan menggunakan metode dekripsi. Sebaliknya untuk mengidentifikasi pesan yang dikirim penerima pesan menggunakan metode dekripsi untuk menerjemahkan pengirim pesan tersebut.

Berdasarkan ilustrasi di atas A membuat sebuah *hash value* dari pesan tersebut. Bilangan tersebut dipangkatkan dengan bilangan *d* dimodulus *n* (seperti halnya pada deskripsi pesan) dan melampirkannya sebagai identitas pengirim pada pesan tersebut.

$$s = h(k)^d \pmod{n}$$

Saat B menerima pesan yang telah memiliki identitas pengirim (*signature*), B memangkatkan *signature* tersebut dengan bilangan *e* dimodulus dengan *n* (seperti halnya pada enkripsi pesan) dan membandingkannya dengan nilai hasil dari *hash value* dengan *hash value* pada pesan tersebut.

$$h(k) = s^e \pmod{n}$$

Jika kedua cocok *hash value*, maka B dapat mengetahui bahwa pemilik dari pesan tersebut adalah A. Pesan tidak mengalai perubahan sepanjang pengiriman.

Padding scheme merupakan hal yang esensial untuk mengamankan pengesahan pesan seperti halnya pada enkripsi pesan, oleh karena itu kunci yang sama tidak digunakan pada proses enkripsi dan pengesahan.

3. Keamanan

Pengamanan dengan metode RSA didasarkan pada dua permasalahan matematika seacra umum yaitu :

- Penanganan masalah faktorisasi pada bilangan yang sangat besar.
- Permasalahan RSA. Yang dimaksud permasalahan RSA adalah penghitungan nilai *n* yang sesuai dari persamaan $q^e = c \bmod n$, dengan (*e*, *n*) adalah *public key* RSA dan *c ciphertext* dari RSA. Saat ini solusi yang paling baik untuk memecahkan algoritma RSA adalah dengan memfaktorkan modulus *n*. Dengan kemampuan faktorisasi bilangan prima maka ada kemungkinan untuk melakukan menghitung bilangan eksponen *d* dari public key (*e*, *n*), lalu mendekripsi *c* dengan menggunakan prosedur yang dipaparkan di atas. Untuk melakukan ini harus dilakukan

faktorisasi bilangan prima *n* menjadi *p* dan *q*, lalu menghitung (*p*-1)(*q*-1) yang mengakibatkan munculnya bilangan *d* dari *e*.

Penemu algoritma RSA menyarankan nilai *p* dan *q* panjangnya lebih dari 100 digit. Dengan demikian hasil kali n = *p* x *q* akan berukuran lebih dari 200 digit. Bayangkan betapa besar usaha yang diperlukan untuk memfaktorkan bilangan 200 digit menjadi faktor primanya. Menurut Rivest dan kawan-kawan usaha mencari bilangan 200 digit membutuhkan komputasi selama 4 milyar tahun (dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma yang tercepat saat ini dan komputer yang dipakai mempunyai kecepatan 1milidetik).[1]

Berdasarkan ilustrasi sebelumnya meskipun ada pihak tertentu yang mendapatkan *public key n* dan *e*, dan *ciphertext c*. Bagaimanapun juga, pihak tesebut tidak mampu untuk secara langsung memperoleh *d* yang dijaga kerahasiannya. Cara paling efektif yang untuk memperoleh *n* dari *c* ialah dengan melakukan faktorisasi *n* kedalam *p* dan *q*, dengan tujuan untuk menghitung (*p*-1)(*q*-1) yang dapat menghasilkan *d* dari *e*.

Masih belum ada bukti pula bahwa melakukan faktorisasi *n* adalah satu-satunya cara untuk memperoleh *n* dari *c*, tetapi sampai saat ini belum ditemukan adanya metode yang lebih mudah.[8]

Sampai saat ini belum ada metode yang efisien dan efektif untuk memfaktorkan bilangan-bilangan bulat yang besar dengan teknologi komputer saat ini, walapun tidak dapat dipastikan bahwa metode tersebut benar-benar tidak ada.

Pada tahun 2005, bilangan faktorisasi terbesar yang digunakan adalah sepanjang 663 bit, dengan menggunakan metode distribusi mutakhir. Kunci RSA pada umumnya sepanjang 1024-2048 bit. Walaupun demikia diyakini bahwa kunci 1024-bit akan dapat dipecahkan di masa depan dan sampai saat makalah ini dituliskan hal ini masih dikembangkan.

Jika *n* sepanjang 256-bit atau lebih pendek, *n* akan dapat difaktorisasi dalam beberapa jam pada *Personal Computer*, dengan menggunakan perangkat lunak yang tersedia. Jika *n* sepanjang 512-bit atau lebih pendek, *n* akan dapat difaktorisasi dalam hitungan ratusan jam seperti pada tahun 1999.

Secara teori, perangkat keras bernama TWIRL dan penjelasan dari Shamir dan Tromer pada tahun 2003 mengundang berbagai pertanyaan akan keamanan dari kunci 1024-bit. Oleh karena itu, disarankan bahwa *n* setidaknya sepanjang 2048-bit.

Pada tahun 1993, Peter Shor menerbitkan Algoritma Shor, menunjukkan bahwa sebuah komputer quantum secara prinsip dapat melakukan faktorisasi dalam waktu polinomial, mengurai RSA dan algoritma lainnya. Bagaimanapun juga, masih terdapat pedebatan dalam pembangunan komputer quantum secara prinsip.[5]

4. Berbagai Hal dalam Penggunaan RSA

Key Generation

Menemukan bilangan prima yang besar *p* dan *q* pada biasanya didapat dengan mencoba serangkaian bilangan acak dengan menggunakan probabilitas bilangan prima (*primaly test*) yang dengan cepat mengeliminasi hampir semua bilangan bukan prima.

Bilangan *p* dan *q* seharusnya tidak memiliki selisih yang kecil (saling berdekatan), agar faktorisasi *Fermat* pada *n* berhasil. Selain itu pula, jika (*p* – 1) atau (*q* – 1) memiliki faktorisasi bilangan prima yang kecil, *n* dapat difaktorkan dengan mudah dan nilai-nilai dari *p* atau *q* dapat diabaikan.

Seseorang seharusnya tidak melakukan metode pencarian bilangan prima yang dapat memberikan informasi penting tentang bilangan prima tersebut kepada pihak lain. Biasanya, *random number generator* (Pemilihan bilangan acak) yang baik akan memiliki nilai bilangan awal yang digunakan untuk pencarian. Dan bilangan yang diperoleh adalah acak dan tidak terduga.

Berikut ini contoh yang mungkin tidak memenuhi kriteria : sebuah bilangan mungkin dapat diperoleh dari proses acak berarti tidak menggunakan pola apapun, tetapi jika bilangan

itu mudah untuk ditebak atau diprediksi (bisa mendekati atau mirip dengan bilangan yang mudah ditebak), maka metode tersebut akan kurang baik untuk diterapkan karena kemungkina adanya celah dalam keamanannya.

Misalnya, tabel bilangan acak yang diterbitkan oleh Rand Corp pada tahun 1950-an mungkin memang benar-benar teracak, tetapi dikarenakan diterbitkan secara umum, hal ini akan mempermudah para pihak tertentu untuk mencari bilangan acak. Jika penyerang dapat menebak separuh dari digit *p* atau *q*, para penyerang dapat dengan cepat menghitung separuh yang lainnya (ditunjukkan oleh Donald Coppersmith pada tahun 1997).

Sangatlah penting bahwa kunci rahasia *d* bernilai cukup besar, Wiener menunjukkan pada tahun 1990 bahwa jika *p* diantara *q* dan 2*q* (yang sangat mirip) dan *d* lebih kecil daripada $n^{1/4}/3$, maka *d* akan dapat dihitung secara effisien dari *n* dan *e*. Kunci enkripsi *e* = 2 sebaiknya tidak digunakan.[5]

Faktorisasi

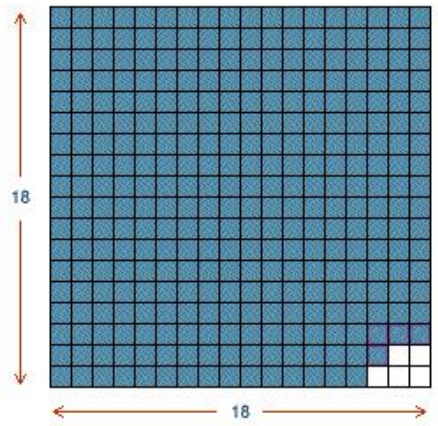
Yang dibahas dalam faktorisasi kali ini adalah faktorisasi Fermat. Tujuannya adalah untuk mengetahui faktorisasi nilai *n* ke dalam *p* dan *q*.

Pembahasan dalam bagian ini berdasarkan pada contoh dengan ilustrasi persegi berikut :

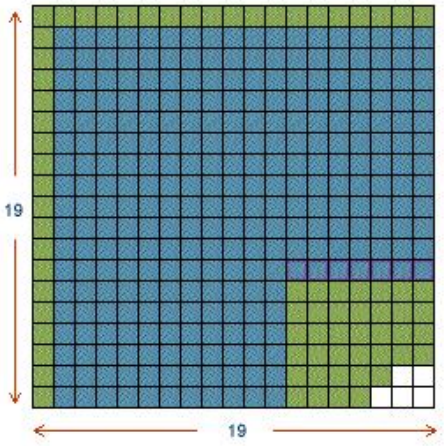
Misalkan kita ingin memfaktorkan 319 (faktor dari 319 adalah 11 dan 29) dengan faktorisasi Fermat.

Langkah-langkahnya adalah :

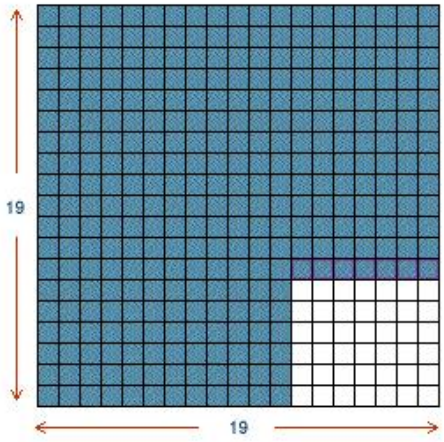
- Akar 319 adalah 17.86057..., dan 319 lebih kecil dari 18^2 yaitu 324.Kita bisa melihat selisih 18 kuadrat dengan 319, yaitu 5. Angka 5 sendiri bukanlah bilangan kuadrat, melainkan 4 lebih kecil dari biangan kuadrat 9 (lih. gambar).



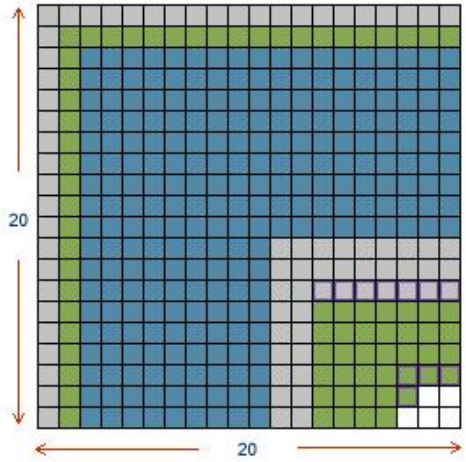
- Lalu perbesar persegi menjadi 19 x 19 yaitu dengan menambahkan 1baris dan 1 kolom di luar persegi (warna hijau). Semuanya berjumlah 37 kotak berwarna hijau. Lalu kita pindahkan kotak dengan jumlah yang sama (37) ke dalam membentuk persegi (warna hijau).



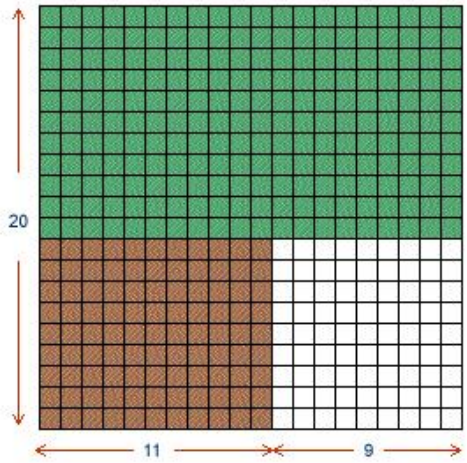
- Sekarang, sama seperti langkah pertama, selisih 19^2 dengan 319 adalah 42. 42 bukanlah blilangan kuadrat, melainkan 7 lebih kecil dari 49 yang merupakan bilangan kuadrat (lih. gambar).



– Sekarang perbesar lagi persegi menjadi 20 x 20, dengan menambahkan 1 baris dan 1 kolom di luar. Jumlah kotak yang ditambahkan (warna abu-abu) adalah 39. Lalu pindahkan kotak-kotak tersebut ke dalam. Setelah di pindahkan ternyata membentuk persegi yaitu 9 x 9 yaitu 81 kotak.



– Dengan membuang persegi bagian dalam (9 x 9) pada langkah sebelumnya kita mendapatkan diagram bentuk L. Bentuk L tersebut terbagi atas dua persegi panjang dengan ukuran 20 x 11 dan 9 x 11. Langkah selanjutnya yaitu memindahkan persegi panjang coklat ke sebelah persegi panjang hijau. Dengan menyatukan kedua persegi panjang, diperoleh persegi panjang dengan ukuran 29 x 11. Maka faktor dari 319 adalah panjang sisi -sisi persegi panjang tersebut yaitu 29 dan 11.



Kecepatan

RSA memiliki kecepatan yang lebih lambat dibandingkan dengan DES dan algoritma simetrik lainnya. Hal ini disebabkan pada prakteknya, jika seseorang menyandikan pesan rahasia menggunakan algoritma simetrik. Berarti proses yang berlangsung adalah penyandian kunci simetrik dengan menggunakan RSA, dan mengirimkan kunci simetrik yang dienkripsi menggunakan RSA dan juga mengirimkan pesan yang dienkripasi secara simetrik kepada si penerima pesan.

Prosedur ini menambah permasalahan dalam sistem keamanan. Dengan kata lain, sangatlah penting untuk menggunakan *random number generator* (pembangkit bilangan acak) yang efektif untuk kunci simetrik yang digunakan, karena adanya kemungkinan terjadi *bypass* terhadap RSA dengan menebak kunci simerik yang digunakan.

Distribusi kunci

Sebagaimana halnya *chipertext*, cara pendistribusian *public key* RSA merupakan hal penting dalam keamanan. Distribusi kunci harus aman dari *man-in-the-middle attack* (*penghadang-ditengah-jalan*).

Misalnya dalam ilustrasi A sebagai pemegang *private key*. Jika ada pihak ketiga C dengan suatu cara mampu memberikan *public key* kepada B

dan membuat B percaya bahwa kunci tersebut milik A. Anggap C dapat menghadang sepenuhnya transmisi antara A dan B. C mengirim B *public key* miliknya sendiri, dan B percaya bahwa *public key* tersebut milik A. C dapat menerjemahkan seluruh *ciphertext* yang dikirim oleh B dengan melakukan dekripsi dengan *private key* milik C sendiri dan menyimpan salinan dari pesan tersebut, lalu melakukan enkripsi menggunakan *public key* milik A, dan mengirimkan *ciphertext* yang baru kepada A.

Dengan kata lain antara pemegang *private key* sebenarnya dan publik terdapat pihak lain yang dapat memanipulasi transmisi pesan antar kedua pihak.

Time Attack

Kocher menjelaskan sebuah serangan baru yang cerdas pada RSA di tahun 1995: jika penyerang C, mengetahui perangkat keras yang dimiliki oleh A secara terperinci dan mampu untuk mengukur waktu yang dibutuhkan untuk melakukan dekripsi untuk beberapa *ciphertext*, maka C dapat menyimpulkan kunci dekripsi *d* secara cepat.

Penyerangan ini dapat juga diaplikasikan pada skema signature RSA. Salah satu cara untuk mencegah penyerangan ini yaitu dengan memastikan bahwa operasi dekripsi menggunakan waktu yang konstan untuk setiap *ciphertext* yang diproses. Cara yang lainnya, yaitu dengan menggunakan properti multipikatif dari RSA.

Cara lain penghitungan $c^d \bmod n$, A memilih nilai bilangan acak *r* terlebih dahulu dan menghitung $(r^e c)^d \bmod n$. Hasil dari penghitungan tersebut ialah $rm \bmod n$ yang kemudian hasil dari *r* dapat dihilangkan dengan perkalian dengan inversnya. Nilai baru dari *r* dipilih pada tiap *ciphertext*. Dengan teknik ini, dikenal sebagai *message blinding* (pembutaan pesan), waktu yang diperlukan untuk proses dekripsi tidak lagi berhubungan dengan nilai dari *ciphertext* sehingga penyerangan waktu akan gagal.[6]

Penyerangan *ciphertext* adaptive

Pada tahun 1998, Daniel Bleichenbacher menjelaskan penggunaan penyerangan *ciphertext* adaptive, terhadap pesan yang terenkripsi menggunakan RSA dan menggunakan PKCS #1 v1 *padding scheme*. Dikarenakan kecacatan pada skema PKCS #1, Bleichenbacher mampu untuk melakukan serangkaian serangan terhadap implementasi RSA pada protokol Secure Socket Layer, dan secara potensial dapat memperoleh kunci-kunci yang digunakan.

Oleh karena itu, para pengguna kriptografi menganjurkan untuk menggunakan *padding scheme* yang relatif terbukti aman seperti *Optimal Asymmetric Encryption Padding*, dan Laboratorium RSA telah merilis versi terbaru dari PKCS #1 yang tidak lemah terdapat serangan ini.[6]

5. Kesimpulan

RSA merupakan system pengamanan yang menggunakan metode public key. Komponen utama RSA terdiri atas *public key* yang sifatnya bebas dan digunakan dalam prose enkripsi dan *private key* yang sifatnya rahasia yang digunakan dalam proses dekripsi.

Saat ini algoritma RSA masih banyak digunakan dalam system pengamanan dalam dunia computer. Metode RSA masih bertahan dan masih teruji keamanannya karena algoritmanya yang sulit dipecahkan dengan teknologi komputer saat ini. Namun hal ini tidak menutup kemingkinan algoritma ini dapat di pecahkan di masa mendatang.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. 2006. *Diktat Kuliah IF2153 Matematika Diskrit.* Program studi Teknik Informatika, Institut Teknologi Bandung. Bandung.

[2] Nursanto, Djoko. 2003. *Meningkatkan Kecepatan Dekripsi RSA Menggunakan Integrasi Metode Montgomery Multification Chinese Remainder Theorem.* Program studi Teknik Informatika, Institut Teknologi Bandung. Bandung.

[3] Rahardjo, Budi. 2002. *Keamanan Sistem Informasi Berbasis Internet*. PT Insan Infonesia. Bandung .

[4] http://computing-dictionary.thefreedictionary.com/RSA. Tanggal akses 29 Desember 2006 pukul 21.30 WIB.

[5] http://en.wikipedia.org/wiki/Public-key_cryptography. Tanggal akses 23 Desember 2006 pukul 16.15 WIB.

[6] http://en.wikipedia.org/wiki/RSA. Tanggal akses 23 Desember 2006 pukul 16.00 WIB.

[7] http://searchsecurity.techtarget.com/sDefiniti on/0,,sid14_gci214273,00.html. Tanggal akses 23 Desember 2006 pukul 16.25 WIB.

[8] http://www.quadibloc.com/crypto/pk0502.htm. Tanggal akses 29 Desember 2006 pukul 21.20 WIB.