Metode Enkripsi RSA

Swastyayana Wisesa 13506005[1)]
1) Jurusan Teknik Informatika ITB, Bandung, email: if16005@students.if.itb.ac.id

Abstract *– Makalah ini membahas tentang metode enkripsi RSA, kegunaannya dan tantangan-tantangan baru yang muncul pada tahun-tahun belakangan ini.*

Kata Kunci*: Metode RSA,Permasalahan RSA, Skema Padding, Teori Bilangan Prima.*

1. PENDAHULUAN

Metode RSA digagas oleh Ron Rivest, Adi Shamir, dan Leonard Adleman dari MIT pada tahun 1977. Walaupun sudah berumur lebih dari 30 tahun, metode ini masih banyak dipakai untuk merahasiakan data. Metode perahasiaan lain juga bisa dipakai bersamaan dengan RSA, membuatnya lebih aman, dan memang sekarang sistem RSA dianjurkan untuk dikombinasikan dengan metode lain (seperti *Padding*) agar keamanannya terjamin. Meskipun mulai banyak metode untuk memecahkan sistem enkripsi ini, waktu dan kompleksitasnya masih lebih besar daripada dekripsi secara legal. Karena metode RSA sangat bergantung pada sulitnya memfaktorkan hasil perkalian dua bilangan prima yang sangat besar, dan memang belum ada algoritma yang ringkas untuk hal ini. Metode RSA diperkirakan masih bisa digunakan untuk belasan tahun lagi, bahkan lebih.

2. ISI

2.1. Sejarah

Algoritma RSA dijabarkan pada tahun 1977 oleh tiga orang: Ron Rivest, Adi Shamir dan Len Adleman dari Massachusetts Institute of Technology. Huruf **RSA** itu sendiri berasal dari inisial nama mereka (**R**ivest—**S**hamir—**A**dleman).

Clifford Cocks, seorang matematikawan Inggris yang bekerja untuk GCHQ (sebuah agen intelejen Inggris), menjabarkan tentang sistem yang setara pada dokumen internal di tahun 1973. Penemuan Clifford Cocks tidak terungkap hingga tahun 1997 karena alasan *top-secret classification*.

Algoritma RSA 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. Saat Algoritma RSA dipublikasikan sebagai aplikasi paten, regulasi di sebagian besar negara-negara lain tidak mengakui penggunaan paten dari luar negeri. Seandainya hasil temuan Clifford Cocks diumumkan, Amerika Serikat juga tidak dapat memberikan hak paten untuk Algoritma RSA.

2.2.1. Pembangkitan Kunci

Semisal Alice berkeinginan untuk mengizinkan Bob untuk mengirimkan kepadanya sebuah pesan pribadi (*private message*) melalui media transmisi yang tidak aman (*insecure*). Alice melakukan langkah-langkah berikut untuk membuat pasangan kunci *public key* dan *private key*:

1. Pilih dua bilangan prima $p \neq q$ secara acak dan terpisah untuk tiap-tiap p dan q. Hitung $N = p\,q$.
2. Hitung $\varphi = (p\text{-}1)(q\text{-}1)$.
3. Pilih bilangan bulat (*integer*) antara satu dan φ ($1 < e < \varphi$) yang juga relatif prima terhadap φ.
4. Hitung d hingga $d\,e \equiv 1 \pmod{\varphi}$.

Catatan :

- bilangan prima dapat diuji probabilitasnya menggunakan *Fermat's little theorem*- a^(n-1) mod n = 1 jika n adalah bilangan prima, diuji dengan beberapa nilai a menghasilkan kemungkinan yang tinggi bahwa n ialah bilangan prima. *Carmichael numbers* (angka-angka Carmichael) dapat melalui pengujian dari seluruh a, tetapi hal ini sangatlah langka.
- langkah 3 dan 4 dapat dihasilkan dengan algoritma *extended Euclidean*
- langkah 4 dapat dihasilkan dengan menemukan bilangan x sehingga $d = (x\,\varphi + 1)/e$ menghasilkan bilangan bulat, kemudian menggunakan nilai dari $d \pmod{\varphi}$;
- menurut *PKCS#1 v2.0* langkah 2 dapat menggunakan $\varphi = kpk\,(p\text{-}1, q\text{-}1)$, dengan kpk = kelipatan persekutuan terkecil.

Pada *public key* diberikan:

- *N*, modulus yang digunakan.
- *e*, eksponen publik (sering juga disebut eksponen enkripsi).

Pada *private key* diberikan:

- *N*, modulus yang digunakan, digunakan pula pada *public key*.
- *d*, eksponen pribadi (sering juga disebut eksponen dekripsi), yang harus dijaga kerahasiaannya.

Demi keringkasan, bentuk lain dari *private key* dapat disimpan (termasuk parameter CRT):

- *p* dan *q*, bilangan prima dari pembangkitan kunci.
- *d mod (p-1)* dan *d mod (q-1)* (dikenal sebagai *dmp1* dan *dmq1*).
- *(1/q) mod p* (dikenal sebagai *iqmp*).

Bentuk ini membuat proses dekripsi lebih cepat dan *signing* dapat menggunakan Chinese Remainder Theorem (CRT). Dalam bentuk ini, seluruh bagian dari *private key* harus dijaga kerahasiaannya.

Alice mengirimkan *public key* kepada Bob, dan tetap merahasiakan *private key* yang digunakan. *p* dan *q* sangat sensitif karena mereka merupakan faktorial dari *N*, dan membuat perhitungan dari *d* menghasilkan *e*. Jika *p* dan *q* tidak disimpan dalam bentuk CRT dari *private key*, maka *p* dan *q* seharusnya telah terhapus bersama nilai-nilai lain dari proses pembangkitan kunci.

2.2.2 Proses enkripsi pesan

Misalkan Bob ingin mengirim pesan *m* ke Alice. Bob mengubah *m* menjadi angka $n < N$, menggunakan protokol yang sebelumnya telah disepakati dan dikenal sebagai *padding scheme*.

Maka Bob memiliki *n* dan mengetahui *N* dan *e*, yang telah diumumkan oleh Alice. Bob kemudian menghitung *ciphertext c* yang terkait pada *n*:

$$c = n^e \mod N$$

Perhitungan tersebut dapat diselesaikan dengan cepat menggunakan metode *exponentiation by squaring*. Bob kemudian mengirimkan *c* kepada Alice.

2.2.3 Proses dekripsi pesan

Alice menerima *c* dari Bob, dan mengetahui *private key* yang digunakan oleh Alice sendiri. Alice kemudian memulihkan *n* dari *c* dengan langkah-langkah berikut:

$$n = c^d \mod N$$

Perhitungan diatas akan menghasilkan *n*, dengan begitu Alice dapat mengembalikan pesan semula *m*. Prosedur dekripsi bekerja karena

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

Kemudian, karena $ed \equiv 1 \pmod{(p-1)}$ dan $ed \equiv 1 \pmod{(q-1)}$, hasil dari *Fermat's little theorem*.

$$n^{ed} \equiv n \pmod p$$

dan

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

Dikarenakan *p* dan *q* merupakan bilangan prima yang berbeda, mengaplikasikan *Chinese remainder theorem* akan menghasilkan dua macam kongruen

$$n^{ed} \equiv n \pmod{pq}.$$

serta

$$c^d \equiv n \pmod N.$$

2.2.2 Padding Scheme

Dalam prakteknya, RSA biasanya digabungkan dengan sebuah metode *padding*. Tujuannya adalah untuk mencegah serangan yang bisa bekerja terhadap RSA tanpa *padding* :

- Saat mengenkripsi dengan nilai *e* dan *m* yang kecil (misal., $e = 3$, $m < n^{1/e}$) hasil dari m^e akan jauh lebih kecil dari modulus *n*. Dalam hal ini, kunci enkripsi akan mudah didapat dari pengambilan akar ke-*e* dari dari *ciphertext c*.
- Karena RSA adalah algoritma yang deterministik (tidak punya bagian acak), penyerang dapat menggunakan serangan plainteks terpilih, dengan mengenkripsi plainteks yang mungkin dan mencocokkannya dengan cipherteks. Sebuah sistem dikatakan aman jika penyerang tidak dapat membedakan dua enkripsi satu sama lainnya bahkan jika si penyerang tahu (atau memilih) plainteks yang bersangkutan. Seperti dijelaskan diatas, RSA tanpa *padding* tidak dianggap aman.
- RSA memiliki sifat yang mengakibatkan hasil kali dua *ciphertext* bernilai sama dengan

hasil enkripisi dua *plaintext* yang bersangkutan, seperti yang digambarkan :

$$m_1^e m_2^e \equiv (m_1 m_2)^e \pmod{n}.$$

Karena sifat perkalian ini, dimungkinkanlah serangan *ciphertext*-terpilih. Misalnya penyerang yang ingin tahu kunci dekripsi *ciphertext* $c=m^e \bmod n$ bisa meminta pemilik kunci rahasia untuk mendekripsikan *ciphertext* $c' = cr^e \bmod n$ untuk sebuah nilai *r* yang dipilih oleh si penyerang. Dengan menggunakan sifat perkalian, maka *c*' adalah hasil enkripsi dari *mr mod n*. Dari sini ia dapat memulihkan *m* dengan mengalikan *mr* dengan invers modular dari *r mod n*.

Untuk menghindari masalah ini, penerapan RSA praktis biasanya menggabungkan sebuah metode *Padding*, yang teratur dan acak, sebelum mengenkripsikannya. Ini memastikan *m* tidak berada di daerah *plaintext* yang tidak aman, dan sebuah pesan, setelah di-*padding* , akan dienkripsi menjadi banyak kemungkinan *ciphertext*.

Padding Scheme harus dibangun secara hati-hati sehingga tidak ada nilai dari pesan *m* yang menyebabkan masalah keamanan. Sebagai contoh, jika kita ambil contoh sederhana dari penampilan ASCII dari *m* dan menggabungkan bit-bit secara bersama-sama akan menghasilkan *n*, kemudian pesan yang berisi ASCII tunggal karakter NUL (nilai numeris 0) akan menghasilkan *n*= 0, yang akan menghasilkan *ciphertext* 0 apapun itu nilai dari *e* dan *N* yang digunakan. Sama halnya dengan karakter ASCII tunggal SOH (nilai numeris 1) akan selalu menghasilkan *chiphertext* 1. Pada kenyataannya, untuk sistem yang menggunakan nilai *e* yang kecil, seperti 3, seluruh karakter tunggal ASCII pada pesan akan disandikan menggunakan skema yang tidak aman, karena nilai terbesar *n* adalah nilai 255, dan 255^3 menghasilkan nilai yang lebih kecil dari modulus yang sewajarnya, maka proses dekripsi akan menjadi masalah sederhana untuk mengambil pola dasar dari *ciphertext* tanpa perlu menggunakan modulus *N*. Sebagai konsekuensinya, standar seperti PKCS didesain dengan sangat hati-hati sehingga pesan asal-asalan pun dapat terenkripsi secara aman. Dan juga berdasar pada bagian Kecepatan, akan dijelaskan kenapa *m* hampir bukanlah pesan itu sendiri tetapi lebih pada *message key* yang dipilh secara acak.

2.2.2. Pengesahan Pesan

RSA dapat juga digunakan untuk mengesahkan sebuah pesan. Misalkan Alice ingin mengirim pesan kepada Bob. Alice membuat sebuah *hash value* dari pesan tersebut, di pangkatkan dengan bilangan *d* dibagi *N* (seperti halnya pada deskripsi pesan), dan melampirkannya sebagai "tanda tangan" pada pesan tersebut. Saat Bob menerima pesan yang telah "ditandatangani", Bob memangkatkan "tanda tangan" tersebut dengan bilangan *e* dibagi *N* (seperti halnya pada enkripsi pesan), dan membandingkannya dengan nilai hasil dari *hash value* dengan *hash value* pada pesan tersebut. Jika keduanya cocok, maka Bob dapat mengetahui bahwa pemilik dari pesan tersebut adalah Alice, dan pesan pun tidak pernah diubah sepanjang pengiriman.

Harap dicatat bahwa *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.

2.3. 1 Pertimbangan Praktis

Penyerangan yang paling umum pada RSA ialah pada penanganan masalah faktorisasi pada bilangan yang sangat besar. Apabila terdapat faktorisasi metode yang baru dan cepat telah dikembangkan, maka ada kemungkinan untuk membongkar RSA.

Pada tahun 2005, bilangan faktorisasi terbesar yang digunakan secara umum ialah sepanjang 663 bit, menggunakan metode distribusi mutakhir. Kunci RSA pada umumnya sepanjang 1024—2048 bit. Beberapa pakar meyakini bahwa kunci 1024-bit ada kemungkinan dipecahkan pada waktu dekat (hal ini masih dalam perdebatan), tetapi tidak ada seorangpun yang berpendapat kunci 2048-bit akan pecah pada masa depan yang terprediksi.

Semisal Eve, seorang *eavesdropper* (pencuri dengar—penguping), mendapatkan *public key N* dan *e*, dan ciphertext *c*. Bagimanapun juga, Eve tidak mampu untuk secara langsung memperoleh *d* yang dijaga kerahasiannya oleh Alice. Masalah untuk menemukan *n* seperti pada $n^e=c \bmod N$ di kenal sebagai permasalahan RSA.

Cara paling efektif yang ditempuh oleh Eve 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*. Tidak ada metode waktu polinomial untuk melakukan faktorisasi pada bilangan bulat berukuran besar di komputer saat ini, tapi hal tersebut pun masih belum terbukti.

Masih belum ada bukti pula bahwa melakukan faktorisasi *N* adalah satu-satunya cara untuk memperoleh *n* dari *c*, tetapi tidak ditemukan adanya metode yang lebih mudah (setidaknya dari sepengatahuan publik).

Bagaimanapun juga, secara umum dianggap bahwa Eve telah kalah jika *N* berukuran sangat besar.

Jika *N* sepanjang 256-bit atau lebih pendek, *N* akan dapat difaktorisasi dalam beberapa jam pada Personal Computer, dengan menggunakan perangkat lunak yang tersedia secara bebas. 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. Sangat 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.

2.3.2 Pembangkitan kunci

Menemukan bilangan prima besar *p* dan *q* biasanya didapat dengan mencoba serangkaian bilangan acak dengan ukuran yang tepat menggunakan probabilitas bilangan prima yang dapat dengan cepat menghapus hampir semua bilangan bukan prima.

p dan *q* seharusnya tidak "saling-berdekatan", agar faktorisasi fermat pada *N* berhasil. Selain itu pula, jika *p*-1 atau *q*-1 memeiliki faktorisasi bilangan prima yang kecil, *N* dapat difaktorkan secara mudah dan nilai-nilai dari *p* atau *q* dapat diacuhkan.

Seseorang seharusnya tidak melakukan metoda pencarian bilangan prima yang hanya akan memberikan informasi penting tentang bilangan prima tersebut kepada penyerang. Biasanya, pembangkit bilangan acak yang baik akan memulai nilai bilangan yang digunakan. Harap diingat, bahwa kebutuhan disini ialah "acak" ***dan*** "tidak-terduga". Berikut ini mungkin tidak memenuhi kriteria, sebuah bilangan mungkin dapat dipilah dari proses acak (misal, tidak dari pola apapun), tetapi jika bilangan itu mudah untuk ditebak atau diduga (atau mirip dengan bilangan yang mudah ditebak), maka metode tersebut akan kehilangan kemampuan 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 penyerang dalam mendapatkan bilangan tersebut. 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 efisien dari *N* dan *e*. Kunci enkripsi *e* = 2 sebaiknya tidak digunakan.

2.3.3 Kecepatan

RSA memiliki kecepatan yang lebih lambat dibandingkan dengan DES dan algoritma simetrik lainnya. Pada prakteknya, Bob menyandikan pesan rahasia menggunakan algoritma simetrik, menyandikan kunci simetrik menggunakan RSA, dan mengirimkan kunci simetrik yang dienkripsi menggunakan RSA dan juga mengirimkan pesan yang dienkripasi secara simetrik kepada Alice.

Prosedur ini menambah permasalahan akan keamanan. Singkatnya, Sangatlah penting untuk menggunakan pembangkit bilangan acak yang kuat untuk kunci simetrik yang digunakan, karena Eve dapat melakukan *bypass* terhadap RSA dengan menebak kunci simetrik yang digunakan.

2.3.4 Distribusi kunci

Sebagaimana halnya *cipher*, bagaimana *public key* RSA didistribusikan menjadi hal penting dalam keamanan. Distribusi kunci harus aman dari *man-in-the-middle attack* (*penghadang-ditengah-jalan*). Anggap Eve dengan suatu cara mampu memberikan kunci sebarang kepada Bob dan membuat Bob percaya bahwa kunci tersebut milik Alice. Anggap Eve dapat "menghadang" sepenuhnya transmisi antara Alice dan Bob. Eve mengirim Bob *public key* milik Eve, dimana Bob percaya bahwa *public key* tersebut milik Alice. Eve dapat menyadap seluruh *ciphertext* yang dikirim oleh Bob, melakukan dekripsi dengan kunci rahasia milik Eve sendiri, menyimpan salinan dari pesan tersebut, melakukan enkripsi menggunakan *public key* milik Alice, dan mengirimkan *ciphertext* yang baru kepada Alice. Secara prinsip, baik Alice atau Bob tidak menyadari kehadiran Eve diantara transmisi mereka. Pengamanan terhadap serangan semacam ini yaitu menggunakan sertifikat digital atau komponen lain dari infrastuktur *public key*.

2.3.5 Penyerangan waktu

Kocher menjelaskan sebuah serangan baru yang cerdas pada RSA di tahun 1995: jika penyerang, Eve, mengetahui perangkat keras yang dimiliki oleh Alice secara terperinci dan mampu untuk mengukur waktu yang dibutuhkan untuk melakukan dekripsi untuk beberapa *ciphertext*, Eve dapat menyimpulkan kunci dekripsi *d* secara cepat. Penyerangan ini dapat juga diaplikasikan pada skema "tanda tangan" 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. Sebagai ganti dari menghitung $c^d \bmod N$, Alice pertama-tama memilih nilai bilangan acak *r* dan menghitung $(r^e c)^d \bmod N$. Hasil dari penghitungan tersebut ialah $rm \bmod N$ kemudian efek 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.

2.3.6 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 mengungkap kunci-kunci yang digunakan. Sebagai hasilnya, 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.

3. PEMBAHASAN

Metode mengalikan dua bilangan prima memiliki kekuatan yang besar, karena algoritma untuk mengetes dan mengalikan dua bilangan prima jauh lebih cepat daripada algoritma untuk memfaktorkannya. Seiring dengan perkembangan teknologi, metode untuk memecahkan pemfaktoran (dan konsekuensinya, RSA) memang semakin cepat, tetapi kita cukup mengatasinya dengan menggunakan kunci yang semakin panjang!

Dengan menggunakan bilangan-bilangan prima yang sangat besar, metode *brute force* juga menjadi jauh lebih lama untuk mendapatkan nilai yang cocok. Sebagai contoh, dengan menggunakan perkiraan bahwa ada sekitar $N/\ln N$ buah bilangan prima dari 1 sampai *N*, jika dipilih nilai N di sekitar 2^{100} , maka akan ada $2^{100}/(200 \ln 2) \sim 9.10^{27}$ buah bilangan prima. Ini menyebabkan pengujian untuk mencari bilangan prima yang cocok dengan kunci dekripsi emerlukan waktu yang sangat lama.

3.1 *Quantum Computing*

Salah satu cara memecahkan masalah RSA, dan yang paling "futuristik" diantaranya, adalah *quantum computing*. Dengan menggunakan arsitektur komputer dimana teori kuantum memegang peranan penting, diharapkan dengan teknik ini pemfaktoran bilangan hasil kali dua bilangan prima akan selesai dalam waktu polinomial.

4. KESIMPULAN

Berdasarkan sulitnya pemfaktoran sebuah bilangan yang sangat besar, dan proses enkripsi dan dekripsi yang selalu lebih mudah dibandingkan proses sebelumnya, metode RSA tetap merupakan sebuah cara untuk merahasiakan data yang aman. Walaupun metode pemecahan masalah RSA yang efisien sudah dirumuskan, implementasinya masih membutuhkan pengembangan untuk beberapa tahun lagi. Sampai saat ini, tahun 2008, lebih dari 30 tahun semenjak RSA dicetuskan pertama kali, sudah banyak waktu para kriptanalis di seluruh dunia yang dihabiskan untuk memecahkan sebuah metode perahasiaan data yang pada dasarnya mudah ini. Untuk beberapa tahun ke depan, masih banyak usaha yang harus dilakukan sampai akhirnya RSA terpecahkan.

DAFTAR REFERENSI

[1] http://en.wikipedia.org/wiki/Rsa, 26 Desember 2007

[2] http://id.wikipedia.org/wiki/Rsa, 31 Desember 2007

[3] http://scottaaronson.com/blog/?p=208, 1 Januari 2008

[4] Munir, Rinaldi, *Matematika Diskrit*, Program Studi Teknik Informatika STEI ITB, 2007

[5] Munir, Rinaldi, *Kriptografi*, Program Studi Teknik Informatika STEI ITB, 2006