STUDI DAN PERBANDINGAN MENGENAI METODE PERKALIAN MONTGOMERY DAN CHINESE REMAINDER THEOREM (CRT) DALAM MEMPERCEPAT DEKRIPSI RSA

Stevens Jethefer – 13504080

Program Studi Teknik Informatika,
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganeca 10, Bandung
E-mail : if14080@students.if.itb.ac.id

- **Abstrak**

Dalam makalah ini akan dibahas tentang studi dan perbandingan mengenai metode perkalian Montgomery dan Chinese Remainder Theorem (CRT) dalam mempercepat dekripsi RSA. Algoritma Perkalian Montgomery adalah algoritma untuk operasi hitung yang menghasilkan 2 bilangan bulat modulo A dan B serta bilangan bulat N. Algoritma perkalian montgomery sekarang ini digunakan sebagai inti algoritma berbasis kriptosistem dalam aritmatik modular. Selain itu munculnya kelas baru pada pemecahan masalah/attack (*timing attack* dan *power attack*), membuat implementasi algoritma ini menjadi lebih mudah dipelajari. Sedangkan salah satu metode yang aman untuk mempercepat operasi dekripsi algoritma RSA itu sendiri digunakan *Metoda Chinese Remainder Theorem*. Metode ini menggunakan faktor prima *mod n*, yaitu p dan q karena $n = p.q$. Sedangkan analisis untuk operasi eksponensialnya menggunakan metoda montgomery.

Dalam makalah ini juga akan dibahas mengenai algoritma RSA itu sendiri, proses enkripsi dan dekripsi kembali untuk mengembalikan *plaintext* hasil enkripsi menjadi seperti sedia kala dengan menggunakan algoritma RSA ini. Lalu akan dijelaskan pula mengenai metode perkalian montgomery serta chinese remainder theorem itu sendiri serta penerapannya dalam RSA untuk mempercepat proses dekripsi. Selain itu akan dibahas pula mengenai metode perkalian montgomery dan CRT dalam dua versi arsitektur. Arsitektur yang pertama (konvensional) menggunakan versi asli dari algoritma perkalian montgomery. Sedangkan arsitektur yang kedua adalah arsitektur hasil optimisasi arsitektur yang pertama dengan algoritma yang dimodifikasi. Arsitektur yang kedua dianggap lebih baik dari yang pertama. Kedua arsitektur memiliki carry di mana secara logika mengurangi redundansi dan dalam perbandingannya dengan arsitektur yang lainnya memberikan hasil yang menarik dalam hal frekuensi *clock,* waktu perkalian, dan *chip Covered area.*

Pada bagian akhir dari makalah, algoritma RSA ini akan dibandingkan performansinya dengan algoritma RSA yang menggunakan perkalian montgomery dan CRT. Performansi dari penggunaan perkalian Montgomery dan CRT akan dianalisis. Kemudian kedua arsitekur dari perkalian montgomery juga akan dibandingkan sehingga dapat diketahui performansi dari masing-masing arsitektur. Terakhir akan dibandingkan dibandingkan performansi dalam hal memori dan kecepatan bermacam-macam algoritma RSA.

***Kata kunci**: Perkalian Montgomery*, *Chinese Reminder Theorem (CRT), RSA,* dekripsi, enkripsi, modulo, eksponen.

1. Pendahuluan

Dalam dunia komunikasi data global yang dinamis dan selalu berubah, serta ditunjang dengan adanya teknologi koneksi Internet dan perkembangan perangkat lunak yang sangat pesat, membuat faktor keamanan menjadi isu yang semakin penting. Keamanan saat ini menjadi suatu kebutuhan dasar bagi setiap orang karena proses komunikasi global sudah tidak lagi aman. Sebagai contoh, dengan berpindahnya suatu data dari titik A ke titik B pada Internet, data tersebut akan melalui beberapa titik lain selama perjalanannya, sehingga membuka kesempatan bagi pihak lain yang tidak berhak untuk memotong data, merubah data bahkan mengubah arah tujuan data. Beberapa fakta dalam kehidupan sehari-hari yang membuat keamanan sistem data menjadi sesuatu yang penting adalah :

1. Tidak ada sistem komputer yang dapat dikatakan aman secara total, yang dapat dilakukan adalah membuat kesulitan bagi orang lain untuk mengganggu sistem komputer yang kita miliki.
2. Semakin aman sistem kita, semakin intrusif keamanan yang diperlukan. Perlu ditentukan tindakan yang membuat sistem masih dapat digunakan dan aman untuk digunakan. Jika terdapat situs dengan ukuran besar hingga menengah, maka perlu ditetapkan suatu Kebijakan Keamanan (Security Policy) yang berisi tingkat keamanan yang dibutuhkan oleh suatu situs dan jenis kegiatan pemeriksaan apa yang perlu digunakan untuk memeriksanya. Contoh dari kebijakan keamanan model ini dapat dilihat di http://ds.internic.net/rfc/rfc2196.txt.
3. Setiap individu mempunyai tingkat privasi yang berbeda. Sebelum mengamankan suatu sistem, perlu ditentukan tingkat ancaman yang dihadapi, risiko apa yang perlu atau tidak perlu diambil, dan seberapa rentan sistem terhadap gangguan.

Dalam dunia keamanan internet, dikenal beberapa istilah orang-orang yang sering didengar. Contohnya adalah hacker dan cracker. Cracker merupakan individu maupun kelompok orang yang memanfaatkan hasil penemuan penyusupan terhadap komputer lain untuk melakukan eksploitasi dan mengambil manfaat dari hasilnya. Motivasi para cracker sangat beragam, diantaranya adalah untuk propaganda (deface web site / email), kriminal murni, penyerangan destruktif (akibat dendam atau ketidaksukaan terhadap suatu institusi), dan lain-lain. Apapun motif dari cracker, selalu ada pihak yang dirugikan akibat tindakannya. Berbeda dengan Cracker, Hacker adalah entitas yang menemukan kelemahan (vulnerability) sistem dalam konteks *security incidents*. Seorang hacker bisa menjadi seorang cracker, tetapi seorang cracker belum tentu menguasai kemampuan yang dipunyai seorang hacker. Motivasi para hacker untuk menemukan *vulnerability* adalah untuk membuktikan kemampuannya atau sebagai bagian dari kontrol sosial terhadap sistem.

Pada prakteknya, pembentukan suatu sistem yang aman akan mencoba melindungi adanya berbagai kemungkinan serangan yang dapat dilakukan pihak lain terhadap suatu sistem antara lain :

- **Intrusion**. Pada penyerangan ini seorang penyerang akan dapat menggunakan sistem komputer yang diserang. Sebagian penyerang jenis ini menginginkan akses sebagaimana halnya pengguna yang memiliki hak untuk mengakses sistem.
- **Denial of services**. Penyerangan jenis ini mengakibatkan pengguna yang sah tak dapat mengakses sistem. Sebagai contoh adalah Distributed Denial of Services (DDOS) yang mengakibatkan beberapa situs Internet tak bisa diakses.
- **Joyrider**. Pada serangan ini disebabkan oleh orang yang iseng dan ingin memperoleh kesenangan dengan cara menyerang suatu sistem. Mereka masuk ke sistem karena beranggapan bahwa mungkin data yang di dalamnya menarik. Rata-rata dilakukan karena rasa ingin tahu, tetapi kadang-kadang menyebabkan kerusakan atau kehilangan data.
- **Vandal**. Jenis serangan ini bertujuan untuk merusak sistem. Seringkali ditujukan untuk site-site besar.
- **Scorekeeper**. jenis serangan in hanyalah bertujuan untuk mendapatkan reputasi dengan cara merusak sistem sebanyak mungkin. Sebagian besar tertarik pada situs-situs tertentu saja. Sebagian tak begitu peduli dengan data yang ada di dalamnya. Saat ini jenis ini lebih dikenal dengan istilah **script kiddies**
- **Mata-mata**. Jenis serangan ini bertujuan untuk memperoleh data atau informasi rahasia dari pihak kompetitor. Saat ini semakin banyak perusahaan yang memanfaatkan jasa ini.

Serangan pada suatu sistem jaringan komputer memiliki 3 gelombang trend utama, yaitu [1]

- Gelombang pertama adalah serangan **fisik.** Serangan ini ditujukan kepada fasilitas jaringan, perangkat elektronis dan komputer. Sebagai pertahanan terhadap serangan jenis ini biasanya digunakan sistem backup ataupun sistem komputer yang terdistribusi. Tujuannya untuk mencegah kesalahan di satu titik mengakibatkan seluruh sistem menjadi tak bekerja. Cara pemecahan terhadap serangan ini telah diketahui dengan baik. Jaringan Internet sendiri sudah didesain untuk mengatasi permasalahan seperti ini.
- Gelombang kedua adalah serangan **sintatik**
  Serangan ini ditujukan terhadap keringkihan (*vulnerability*) pada suatu perangkat lunak, celah yang ada pada algoritma kriptografi atau protokol. Serangan Denial of Services (DoS) juga tergolong pada serangan jenis ini. Serangan jenis ini yang saat ini paling populer. Tetapi cara penanganannya relatif telah diketahui dan biasanya disebabkan

karena pihak administrator atau pengguna yang lalai menerapkannya.

- Gelombang ketiga adalah serangan **semantik**
  Serangan jenis ini memanfaatkan arti dari isi pesan yang dikirim. Dengan kata lain adalah menyebarkan informasi palsu melalui jaringan, atau menyebarkan informasi tertentu yang mengakibatkan timbulnya suatu kejadian. Pada dasarnya banyak pengguna cenderung percaya apa yang mereka baca. Seringkali keluguan mempercayai berita ini disalah-gunakan pihak tertentu untuk menyebarkan isu-isu yang menyesatkan.

Disebabkan adanya berbagai ancaman keamanan terhadap suatu perangkat lunak atau jaringan, maka diciptakan suatu algoritma perlindungan yang dikenal dengan algoritma kriptografi. Salah satu algoritma kunci publik yang paling terkenal adalah RSA. Algoritma ini sudah dipergunakan secara luas, salah satunya untuk tanda tangan digital dan pada komunikasi transaksi. RSA menawarkan keamanan yang baik walaupun membutuhkan komputasi matematika yang kompleks sehingga mengurangi kecepatan dari algoritma ini bila dibandingkan dengan algoritma kunci simetri. Oleh karena itu dibutuhkan suatu metode untuk mempercepat perhitungan yang terdapat pada RSA. Operasi matematika yang terdapat di belakang algoritma ini dapat dibagi menjadi dua operasi, yaitu operasi modular dan perpangkatan. Sehingga untuk membuat implementasi dari RSA yang lebih efisien, maka diperlukan suatu perpangkatan yang efisien di antara dua bilangan modular. Walaupun demikian, perpangkatan modular memiliki kekurangan, yaitu operasi pembagian yang harus dilakukan untuk mendapatkan nilai remainder. Operasi pembagian merupakan operasi yang memakan waktu dan memiliki kompleksitas yang besar.
Solusi dari permasalahan perpangkatan modular adalah dengan mencari suatu metode yang memotong jumlah percobaan operasi pembagian. Banyak algoritma yang sudah didesain untuk menangani masalah ini. Namun algoritma yang paling popular dan berguna adalah algoritma perkalian Motgomery. Algoritma ini menormalisasi hasil dari perpangkatan dan mengurangi kompleksitas dari perpangkatan modular terutama bila operasi dilakukan dalam jumlah yang besar seperti pada kunci publik pada RSA. Kelebihan utama dari perkalian Montgomery dibandingkan dengan algoritma perpangkatan modular yang lainnya adalah algoritma ini menggunakan operasi matematika yang sederhana (seperti penambahan dan perpangkatan) dan proses yang sederhana, tetapi dapat memberikan hasil yang akurat. Kebutuhan akan operasi pembagian atau bahkan invers dilakukan dengan pergeseran. Sudah banyak implementasi dari algoritma ini yang sudah dibuat, baik untuk hardware maupun software. Beberapa implementasi menggunakan representasi dalam radix besar sehingga meningkatkan kompleksitas pada saat di sisi lain orang memodifikasi algoritma untuk meningkatkan performansi.

Perbedaan utama antara RSA dan algoritma lain berbasis logaritma diskrit adalah modulus yang digunakan dalam enkripsi RSA adalah bilangan hasil dari dua bilangan prima. Sehingga hal ini memungkinkan untuk digunakannya *Chinese Remainder Theorem* (CRT) untuk meningkatkan kecepatan operasi pembangkitan kunci privat. Dengan menggunakan keuntungan Chinese Remainder Theorem (CRT), usaha komputasi yang dilakukan proses dekripsi RSA dapat dikurangi secara signifikan. Jika dua bilangan prima P dan Q dan modulusnya N diketahui, dimungkinkan untuk menghitung perpangkatan modular $M = C^D \bmod N$ secara terpisah mod P dan mod Q dan dengan lebih ringkas.

2. RSA

RSA adalah salah satu dari algoritma kunci publik yang sangat sering digunakan untuk mengotentikasi keaslian suatu data digital. Keamanan enkripsi/dekripsi data dari algoritma kriptografi ini terletak pada kesulitan untuk memfaktorkan modulus n yang sangat besar. Besarnya bilangan yang digunakan mengakibatkan lambatnya operasi yang melibatkan algoritma RSA ini. Dibandingkan dengan algoritma kunci privat seperti *DES*, RSA membutuhkan waktu komputasi yang lebih lambat pada saat implementasi.

Di tahun 1978, Ron Rivest, Adi Shamir dan Leonard Adleman membuat sebuah algoritma untuk teori penomoran pada sebuah kunci publik, algoritma ini dikenal dengan nama *RSA*. Pada perkembangannya RSA banyak digunakan karena kemudahannya dan keamanannya.

Contoh Permasalahan :
Dalam berkomunikasi menggunakan jaringan internet, adanya lubang/celah dalam keamanan menjadi masalah serius karena perkembangan e-mail, e-banking, e-business dan jenis komunikasi lainnya makin pesat untuk mentranfer data-data penting. Sebagai contoh kasus Alice dan Bob yang ingin berkomunikasi dengan privasi tinggi, pada kenyataannya ada pihak ketiga yang dapat mengakses kemunikasi mereka, dalam hal ini misalkan namanya Eve.

- Enkripsi Simetrik
  Salah satu solusi untuk Alice dan Bob dalam kasus ini adalah tukar-menukar data disertai suatu kunci digital, dimana keduanya telah sama-sama mengetahui kunci rahasia tersebut. Alice menggunakan kunci ini untuk mengkodekan pesan yang dia kirimkan, dan Bob merekonstruksi pesan tersebut kembali dengan kunci yang sama. Pesan yang dienkripsi (ciphertexts) tetap dapat diterima Eve tetapi ia tidak dapat membacanya karena tidak mempunyai kunci yang sama untuk mendekripsi kode pesan tersebut.
- Enkripsi Kunci Publik
  Dalam penggunaan model enkripsi ini, kunci publik yang digunakan berbeda untuk enkripsi dan dekripsi. Terdapat dua buah kunci publik, dimana setiap orang dapat mengetahui dan menggunakan kunci publik ini, tetapi hanya orang yang menerima pesan saja yang mengetahui kunci privat miliknya untuk melakukan dekripsi pesan yang diterima. Metoda ini memungkinkan Alice dan Bob tetap dapat berkomunikasi tanpa terganggu oleh pihak ke tiga.

2.1 Algoritma RSA

1. Pilih dua bilangan prima sembarang, p dan q.
2. Hitung n = p·q (sebaiknya p tidak sama dengan q, sebab juka p sama dengan q maka n = p^2 sehingga p dapat diperoleh dengan menarik akar pangkat dua dari n)
3. Hitung $\phi(n) = (p - 1)(q – 1)$.
4. Pilihlah kunci publik, e, yang relatif prima terhadap $\phi(n)$. Maksudnya relatif prima adalah bilangan terbesar yang dapat membagi e dan $\phi(n)$ untuk menghasilkan nilai 1 (pembagi ini dinyatakan dengan gcd --greatest common divisor). Algoritma Euclid's digunakan untuk mencari gcd dua bilangan tersebut.
5. Bangkitkan kunci privat dengan menggunakan persamaan $e \cdot d \equiv 1 \pmod{\phi(n)}$. Perhatikan bahwa $e \cdot d \equiv 1 \pmod{\phi(n)}$ ekivalen dengan $e \cdot d \equiv 1 + k\phi(n)$, sehingga secara sederhana d dapat dihitung dengan $d = (1 + k\phi(n)) / e$.

Hasil dari algoritma di atas adalah :

- Kunci publik adalah pasangan (*e,n*)
  - Dipublikasikan bebas
  - Pengiriman balik pesan kepada pemegang kunci privat untuk mengenkripsi pesan
- Kunci privat adalah pasangan (*d, n*)
  - Rahasia pemegang *(end user)*
  - Digunakan untuk mendekripsi pesan yang ditujukan kepadanya
  - Dapat berfungsi sebagai *digital signature* yang beroperasi dengan menggunakan *privat key*

Catatan : n tidak berifat rahasia, sebab ia diperlukan pada perhitungan enkripsi/dekripsi.

2.2 Enkripsi / Dekripsi RSA

Enkripsi

1. Ambil kunci publik penerima pesan, e, dan modulus n.
2. Nyatakan plainteks m menjadi blok-blok m_1, m_2,...,sedemikian seterusnya sehingga setiap blok merepresentasikan nilai di dalam selang [0, n -1].
3. Setiap blok *m,* dienkripsi menjadi blok c_i, dengan rumus $c_i = m_i^e \bmod n$.

Dekripsi

1. Setiap blok chiperteks c_i didekripsi kembali menjadi blok m_i dengan rumus $m_i = c_i^d \bmod n$.

3 Aritmatik Bilangan Bulat Multipresisi

3.1 Perkalian Multipresisi

Didefinikan x dan y adalah bilangan bulat yang dinyatakan dalam representasikan radiks b : $x = (x_n x_{n-1} \ldots x_1 x_o)_b$ dan $y = (y_t y_{t-1} \ldots x_1 x_o)_b$ basis digit b. Jika kedua bilangan tersebut mempunyai panjang yang sama s hasilnya adalah 2s digit.

Algoritma 1. Perkalian Multi-precision bilangan bulat :
Input : bilangan bulat positif x dan y dengan s digit base b
Output : Hasil $x.y = (w_{2s-1} \ldots w_1 w_0)_b$

1. for $i = 0$ to s -1
2. C <- 0
3. for $j = 0$ to s -1
   3.1. $(C,S) \leftarrow w_{i+j} + x_i + y_i + C$
   3.2. $w_{i+j} \leftarrow S$
4. return $(w_{2s-1} \ldots w_1 w_0)_b$

Contoh :
$x_j = A, y_i = B, w_{i+j} = F, C = E$
$(C,S) = 8B = F + AB + C$ sehingga,
$(C,S) = (8,B)$ dan $w_{i+j} = S = B$, dan pada siklus berikutnya dari *loop j* C = 8.

3.2 Perkalian Modular

x dan *y* ϵ *Zn* dan *n* bilangan bulat, perkalian modular dinyatakan sebagai :
x, y mod n
operasi perkalian modular dapat dilakukan dengan melakukan perkalian standar terhadap x dan y, kemudian diikuti dengan mengambil sisa pembagian *x, y/n*. Metode ini dikenal sebagai algoritma klasik.

3.3 Perkalian Montgomery

Akan dijelaskan pada bagian 4.

4 Perkalian Montgomery

Perkalian Montgomery sekarang ini digunakan sebagai inti algoritma berbasis kriptosistem dalam aritmatik modular. Munculnya kelas baru pada pemecahan masalah serangan / attack (*timing attack* dan *power attack*), membuat implementasi dari algoritma ini menjadi lebih mudah dipelajari.

Selama ini IEEE menggunakan algoritma dari Colin D.Walker sebagai standar kecepatan RSA, dimana hal ini dapat ditingkatkan dengan penambahan metode perkalian montgomery. Selain RSA yang berbasis kriptosistem, perhitungan bilangan eksponen modular sering dilakukan dengan perkalian montgomery. Beberapa implementasi dari algoritma ini telah banyak diterapkan dalam hardware dan software dengan tujuan utama untuk mempercepat proses enkripsi/dekripsi.

Perkalian Montgomery adalah algoritma yang digunakan untuk mengomputasikan hasil dari operasi hitung yang menghasilkan 2 bilangan bulat A dan B modulo bilangan bulat N. Karena A dan B mempunyai komposisi yang besar (dengan alasan keamanan RSA), maka dengan metoda Montgomery Multiplication ini data akan di tata ulang dalam blok-blok kecil. Panjang t ini akan ditata menjadi 8, 16, 32, 64 dan kelipatannya mengikuti perumusan :

$X = \sum_{i=0}^{p-1} x_i 2^{it}$

dimana, p = blok-blok bilangan yang diperlukan untuk merepresentasikan semua bilangan yang digunakan dalam algoritma.

Contoh algoritma metode perkalian montgomery dapat dilihat pada gambar analisis di bawah ini. Pada algoritma di bawah ini tidak membutuhkan pembagian apapun (operasi yang mahal). Selain itu pada algoritma ini, perkalian dilakukan dari kiri (bit urutan tinggi) ke kanan (bit urutan rendah) di mana hal ini merupakan bukan urutan umum yang dilakukan untuk melakukan perkalian.

{Pre-condition: N prime to 2^t}
$S = 0$
for $i = 0$ **to** $p - 1$
  $q_i = (s_0 + a_i b_0) n'_0 \bmod 2^t$
  $S = (S + a_i \times B + q_i \times N) \text{ div } 2^t$
  {Invariant: $0 \leq S < N + B$}
endfor
{Post-condition: $S2^{pt} = A \times B + Q \times N$}

Gambar 1 : Perkalian Montgomery

Bila n adalah bilangan bulat positif *k* bit dan *r* serta *T* adalah bilangan bulat sedemikian rupa sehingga *r* > n, *gcd(n,r) = 1* dan 0 ≤ T < n. Tr^{-1} *mod n* disebut *montgomery reduction* dari *T modulo n*. Dengan representasi dari *residue class modulo n*, algoritma ini menggantikan operasi pembagian oleh pangkat dari 2. Anggap algoritma memerlukan *r* dan n relatif prima yaitu *gcd(n,r) = 1*. Kemudian ditentukan bahwa $r = 2^k$, maka syarat *gcd(n,r)= 1* dipenuhi jika n adalah ganjil.

Dalam aplikasi *RSA* syarat ini dipenuhi, yaitu modulus n adalah selalu ganjil. Anggap a dan c adalah bilangan bulat yang sedemikian rupa sehingga 0 ≤a,c<n. Agar ā =ar mod n dan *č* = *cr mod n*. *Montgomery reduction* dari *ac* adalah acr^{-1} *mod n*. ā dan č disebut *representasi montgomery* dari a dan c. Disini r^{-1} adalah invers *r mod n*, yaitu $r^{-1}r = 1 \bmod n$. Untuk menggambarkan algoritma montgomery diperlukan besaran tambahan *n'*, yaitu bilangan bulat yang memenuhi $r.r^{-1} - n.n' = 1$ r^{-1} dan n' dapat dihitung dengan algoritma *extended euclidean*.

4.1 Eksponensiasi Basis Montgomery

4.1.1 Deskripsi

Perkalian montgomery adalah komponen dasar yang digunakan untuk mengimplementasikan sebuah perpangkatan klasik dan algoritma perkalian yang menghitung sebuah bilangan eksponensial. Hasil perkalian montgomery $(\overline{\times})$ bukan A x B mod N tetapi A x B x 2^{pt} mod N. Agar hasil operasi menjadi benar di akhir eksponensial, maka kita perlu membuat :
Pre-perkalian : $(A \overline{\times} 2^{2pt} \bmod N)$
Post- perkalian : $(A^e \overline{\times} 1 \bmod N)$
Dengan asumsi :
A < 2N, t >= 1, dan $2N < 2^{(p-1)t}$ C

C. Walker berpendapat bahwa ini adalah hasil akhir dimana eksponen (E) akan lebih kecil dari modulus N dan tidak memerlukan reduksi lanjutan, tetapi Quisquater berpendapat bahwa hasil perkalian adalah input untuk proses berikutnya dimana output ini harus mempunyai

batasan seperti input. Pada iterasi kedua terakhir, S' < N + B. Dengan asumsi A < 2N dan 2N < $2^{(p-1)\ t}$ menjamin a_{p-1}=0. Sehingga pada iterasi terakhir kita mendapatkan S < N + 2^{-t} B < 2N. Dan pada perkalian terakhir eksponensial didapat A^{ε} < 2N. Post-multiplication /perkalian dengan 1 akan mengembalikan kepada nilai terakhir yang memungkinkan sehingga hasil akhir : $E2^{pt} = A^{\varepsilon}$ + QN, Q < 2^{nt} dan A^{ε} < 2N mengimplikasikan $E2^{pt} < (2^{pt} + 1)N$.

Dihasilkan S <= N (S adalah bilangan bulat). Pada kasus S = N akan dihilangkan sebab ia harus memenuhi : $A^{\varepsilon} \equiv 0 \bmod N$ dan oleh karena itu $A \equiv 0 \bmod N$. Hal ini menandakan bila A = 0 (tak ada reduksi) atau A = N (didalam kriptosistem klasik, A < N).

4.1.2 Shortcoming

Dengan A, B $\leq$ 2N, t $\geq$ 1 dan 2N < $2^{(p-1)}$, output perkalian dibatasi : S < 2N. Selama hasil ini berlaku maka kondisi tersebut dikatakan fase pre-perkalian. Dalam fase-perkalian, bilangan bulat A dikalikan dengan 2^{2pt} yang secara jelas lebih besar dari 2N dan mengakibatkan tidak adanya jaminan S akan keluar dari 2N setelah fase pre-perkalian. Untuk mengatasi hal ini dapat menggunakan 2 cara, yaitu :

1. pre-compute : 2^{2pt} mod N
2. menggunakan algoritma perkalian modular normal dan hitung A x 2^{2pt} mod N

Akan tetapi inipun menimbulkan masalah kecil yaitu performa akan tergangggu, sehingga akan mengganggu jika ukuran N dan t menggunakan ukuran normalnya.

4.1.3 Optimisasi *Bound*

Dengan batasan (S < 2 N) dimana serta t $\geq$ 2 tidak akan menimbulkan masalah pada implementasi software. Disisi hardware performa ini menjadi masalah. Jika ukuran N lebih kecil dari 2^t hasil ini tidak akan bekerja. Namun hal ini jarang terjadi karena karena ukuran minimum N tidak kurang dari 512 bit. Setiap langkah dari *bound* sesuai dengan : S < N + B. Dari N < $2^{(p-1)\ t}$ dan A < 2N, diketahui a_{p-1} = {0,1}. Jika kita memulai dari iterasi terakhir, maka didapatkan :

$$
\begin{aligned}
S' &= (S + a_{p-1} \times B + q_{p-1} \times N) \text{ div } 2^t \\
S' &\leq (S + B + q_{p-1} \times N) \text{ div } 2^t \\
S' &\leq (S + B + (2^t - 1) \times N) \text{ div } 2^t \\
S' &< (N + B + B + (2^t - 1) \times N) \text{ div } 2^t \\
S' &< (2B + 2^t \times N) \text{ div } 2^t \\
S' &< 2B \text{ div } 2^t + N \\
S' &< 4N \text{ div } 2^t + N \\
S' &< 2N \quad \square
\end{aligned}
$$

4.2 Pertimbangan Keamanan

Dalam prose perkalian montgomery ini terdapat 2 masalah yang dapat mengganggu proses, yaitu : time attack dan *power attack.*

4.2.1 Timing Attack

Algoritma RSA yang original sebelum melalui proses modifikasi apapun sebenarnya telah memiliki proteksi terhadap time attack. Dalam pembahasan ini security bukanlah menambahkan jumlah algoritma keamanan tetapi membuat sebuah desain yang berfungsi sebagai cleaner yang selalu melakukan pengurangan dan menjumlah loop kosong pada bagian akhir eksponensiasi.

Size of N	This paper	Walter's version
512 bits	6.3 %	17.6 %
768 bits	4.3 %	11.9 %
1024 bits	3.3 %	9 %
2048 bits	1.6 %	4.5 %

Tabel 1 : Waktu penambahan rata-rata

Size of N	This paper $(4(p+1)+3)(p+1)$	Walter's version $(4(p+2)+3)(p+2)$
512 bits ($p = 16$)	10.9 %	24 %
768 bits ($p = 24$)	7.3 %	15.9 %
1024 bits ($p = 32$)	5.5 %	11.9 %
2048 bits ($p = 64$)	2.7 %	5.9 %

Tabel 2 : prediksi penambahan waktu

4.2.2 Power Attack

Algoritma kecepatan RSA yang asli, setelah melakukan pengurangan akhir , sebuah instruksi kondisi menentukan hasil di mana pengurangan akhir harus dibuang. Sebab hasil yang sebelumnya tadi dikembalikan oleh nilai yang bukan oleh alamatnya. Jika hasil tersebut terus disimpan , nilai ini akan tercetak ulang (copy). Untuk menghindari terjadinya cetak ulang tersebut sebuah loop kosong mensimulasikan waktu pengambilan oleh nilai copy-an tersebut. Metode ini dapat dideteksi

dengan mudah dalam sebuah *power attack*. Dalam versi baru, sebuah penguat keamanan telah dapat menghindari *power attack* sebab tak ada instruksi kondisi yang eksis selama proses montgomery tersebut.

4.3 Algoritma Montgomery

4.3.1 Algoritma Konvensional

Algoritma perkalian Montgomery menghitung nilai dari $A = X \times Y \times R^{-1} \bmod N$ di mana R adalah sebuat konstanta yang biasanya R=2n. Nilai value N n-bit adalah bilangan bulat yang memenuhi kondisi gcd(R,N)=1. Jika N ganjil, seperti pada RSA, batasan di atas akan selalu benar. Hal ini juga disebabkan algoritma yang digunakan untuk perpangkatan keluaran A selalu tidak pernah lebih dari N sehingga pembagian akhir pada [2] dihilangkan [4]. Sehingga algoritmanya menjadi :

Function MM (X, Y, N)

1. A=0
2. **For** k=0 to n-1 **do begin**
3. $q=(a_0 + x_k y_0) \bmod 2$
4. $A=A+x_k Y+qN$
5. A=A/2

   End
6. **Return** A

Dengan menggunakan logika redundan *Carry Save*, algoritma di atas dapat berubah menjadi :

Function MMcs (X, Y, N)

1. $C_{2in}=0$, $C_{1in}=0$, $S_{in}=0$
2. **For** k=0 to n-1 **do begin**
3. $q=(S_{in0} + C_{1in0}+C_{2in0} + x_k y_0) \bmod 2$
4. $C_1+C_2+S= C_{2in}+C_{1in}+S_{in}+x_k Y+qN$
5. $C_{2in}=(C_2)/2$, $C_{1in}=(C_1)/2$, $S_{in}=(S)/2$

   End
6. **Return** $C_{2in}/2$, $C_{1in}/2$ and $S_{in}/2$

4.3.2 Algoritma Optimisasi

Mengobservasi algoritma asli dari perkalian Montgomery, fungsi MM pada dasarnya terdiri dari loop penambahan A dengan nilai Y, N, Y+N atau nol (0). Pilihan dari nilai mana yang akan ditambahkan pada A tergantung dari nilai dari x0 dan q. Sehingga dengan mengetahui x0 dan q, dapat ditambahkan nilai yang diperlukan pada A.

Menggunakan hasil observasi pada logika redundan Carry - Save, versi yang dimodifikasi pada algoritma perkalian Montgomery dapat diekstrak mirip dengan algoritma perkalian Montgomery yang telah dijelaskan pada bagian sebelumnya.

Function MMCh (X, Y, N)

1. $C_{in}=0$, $S_{in}=0$
2. **For** k=0 to n-1 **do begin**
3. $q=(S_{in0} + C_{in0} + x_k y_0) \bmod 2$
4. **if** (x_0 =0) **then**
   a. **if** (q=0) **then**
   b. I=0
   c. **Else**
   d. I=N
   e. **End**
5. **End.**
6. **if** (x_0 =1) **then**
   a. **if** (q=0) **then**
   b. I=Y
   c. **Else**
   d. I=Y+N
   e. **End**
7. **End.**
8. $C+S= C_{in}+S_{in}+I$
9. $C_{in}=(C)/2$, $S_{in}=(S)/2$

   End
10. **Return** $C_{in}/2$ and $S_{in}/2$

Seperti yang dapat dilihat pada modifikasi di atas, kebutuhan akan penambahan pada satu putaran loop benar-benar diperkecil bila dibandingkan dengan algoritma perkalian Montgomery. Hal ini dilakukan dengan argumen if. Nilai dari Y+N dapat diprekomputasikan sehingga tidak mengubah efisiensi dari algoritma itu sendiri. Secara keseluruhan, walaupun demikian, perolehan yang didapat pada penambahan dengan kondisi if dan perkalian Montgomery dapat dipertimbangkan, sama seperti yang dilihat pada arsitektur, optimisasi dari algoritma perkalian Montgomery.

4.4 Arsitektur Perkalian Montgomery

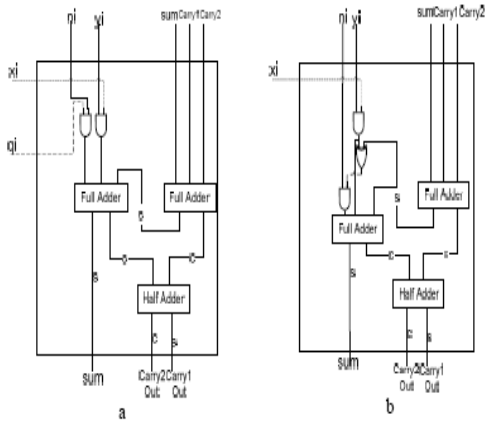
Pada bagian ini, dua arsitektur akan dijelaskan. Arsitektur konvensional berdasarkan pada algoritma fungsi MMcs dan arsitektur optimisasi berdasarkan algoritma MMCh

4.4.1 Arsitektur Konvensional

Algoritma MMcs menggunakan aritmatika Carry – Save untuk mengurangi jalur-jalur genting pada hasil. Walaupun demikian, seperti yang bisa dilihat pada tahap 4 dari algoritma, sebuah penambahan lima angka diperlukan untuk menghasilkan hasil dengan tiga nilai. Penambahan pada tahap tiga adalah modulo 2. Oleh karena itu dapat dilakukan

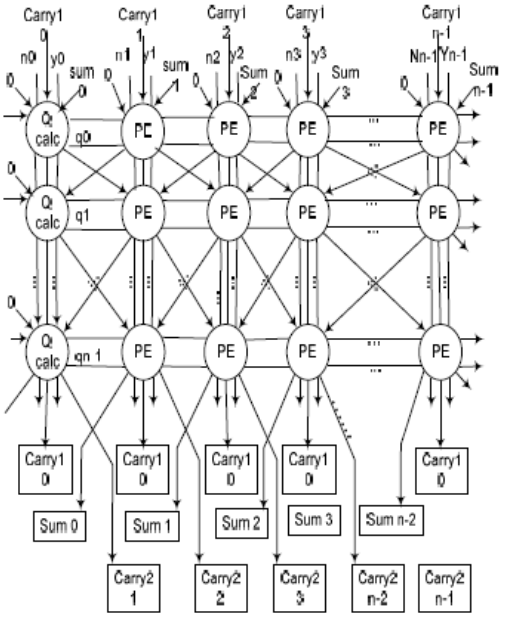
operasi XOR yang lebih sederhana. Walaupun demikian, tidak dapat dilakukan pada tahap 4 dari algoritma di mana sebuah penambahan dari lima angka diperlukan untuk menghasilkan hasil dengna tiga nilai.

Arsitektur yang dijelaskan adalah arsitektur sistolik. Setiap elemen proses (EP) harus melakukan 5 – 3 penambahan menggunakan nilai x dan q sekarang ini. Walaupun demikian. EP pada bit 0 memiliki sebuah aturan penambahan untuk dilakukan, perhitungan dari nilai q. Oleh karena itu EP dari bit 0 memiliki dua gerbang yang lebih banyak dari yang lainnya. Dua tipe dari EP dapat dilihat pada gambar di bawah ini.



Gambar 2(a) : Gambar Proses Elemen
Gambar 2(b) : Gambar Proses Elemen dalam mencari nilai q

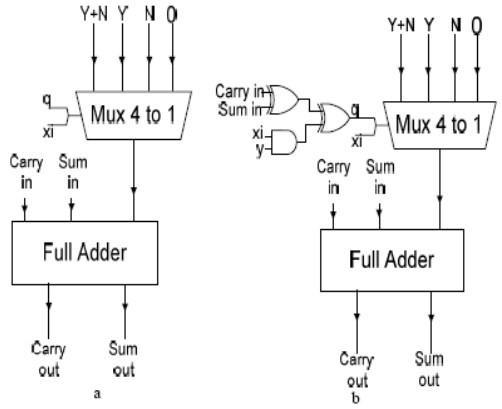
Pada arsitektur sistolik pada gambar 3 di bawah ini, hubungan dari elemen EP memiliki hubungan logika sebagai berikut :
Setiap kolom yang berbeda merepresentasikan nilai dari bit yang sama melewati perhitungan algoritmik pada saat semua kolom yang berbeda merepresentasi nilai pada putaran yang lainnya dari fungsi MMcs.
Signal Carry2 dari EP dihubungkan dengan EP selanjutnya pada kolom berikutnya dari EP, Signal Carry1 dihubungkan dengan EP yang sama dari baris berikutnya EP (kolom yang sama)
Pada saat signal penjumlahan dihubungkan dengan EP sebelumnya dari baris berikutnya yang dimulai dengan bit 0.
Menggunakan formula ini, diatur sedemikian rupa untuk menghindari operasi pergeseran pada langkah ke-5. Hasilnya adalah format Carry – Save yang melewati signal Carry2, Carry1, dan Sum.



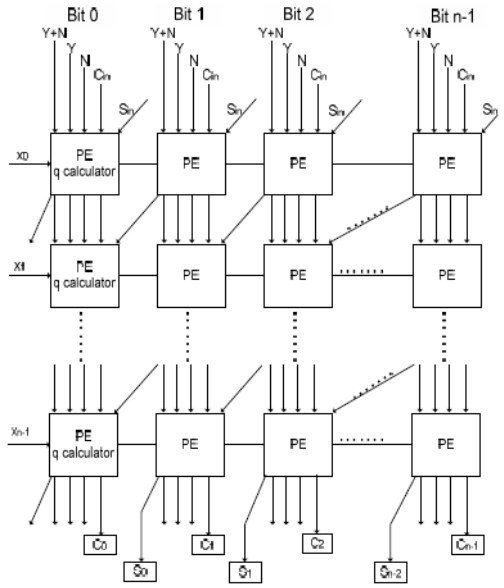
Gambar 3 : Gambar Arsitektur konvensional menggunakan fungsi MMc.

4.4.2 Arsitektur Dengan Optimisasi

Seperti yang bisa dilihat, pada fungsi MMCh, jumlah penambahan pada setiap EP lebih kecil daripada yang terdapat pada fungsi MMcd. Hal ini menandakan fungsi ini bisa dibuat menjadi arsitektur dengan logika yang jauh lebih sederhana. Walaupun demikian, masih diperlukan dua jenis dari elemen proses, EP sederhana dan EP yang menghitung nilai q. Perbedaannya EP pada arsitektur optimasi adalah arsitektur ini hanya memiliki satu *Full Adder* dan sebuat *4-to-1Multiplexer*, seperti yang bisa dilihat pada gambar 4, di mana jauh lebih sederhana daripada terdapat 2 Full Adder dan 1 Half Adder elemen proses dari arsitektur konvensional yang sudah dijelaskan sebelumnya.



Gambar 4(a) : gambar Elemen Proses
Gambar 4(b) : gambar EP yang menghitung q pada arsitektur optimisasi.



Gambar 5 : Gambar Aristektur optimisasi menggunakan fungsi MMCh.

Arsitektur optimisasi sistolik yang telah dijelaskan memiliki struktur yang mirip dengan stuktur arsitektur konvensional. Walaupun demikian, interaksi pada signal Carry – Save jauh lebih sederhana. Dimulai dari bit 0, signal Carry dihubungkan dengan EP yang sama dari baris berikutnya dari EP (kolom yang sama) pada saat signal Sum dihubungkan dengan EP sebelumnya pada baris berikutnya. Dengan cara tersebut, operasi pergeseran dilakukan melewati signal interkoneksi yang benar. Struktur sistolik dari arsitektur ini ditunjukan pada gambar 5 di atas.

5 Chinese Remainder Theorem

Biasanya kunci publik e dari RSA adalah nilai yang relatif rendah, contohnya 216 + 1 (sebuah nilai yang standar). Sehingga, pada proses chiper (bukan pada proses pemberian tanda tangan digital) tidak akan diperoleh masalah dengan kecepatan chiper karena bilangan pangkat e akan relatif kecil

Pada saat bilangan n = p*q menjadi jauh lebih besar, dengan urutan $2^{1.024}$ jika membicarakan tentang kunci dengan panjang 1.024 bit, kunci privat d biasanya akan jauh lebih besar daripada nilai e dan nilai ini akan jatuh sangat dekat dengan nilai dari 1.024 bit. Oleh karena itu, akan sangat mahal bagi penerima pesan untuk mendekripsi pesan dengan kunci privat yang dimilikinya atau untuk menandatangani suatu dokumen dengan kunci privat tertentu.

Solusi yang ditawarkan adalah dengan menggunakan Chinese Remainder Theorem (CRT): daripada bekerja dengan nilai n, akan lebih baik bekerja dengan nilai p dan q sehingga perpangkatan modular akan dapat dilakukan dengan p dan q, jauh lebih cepat daripada menggunakan n.

5.1 Definisi CRT

Suatu residu, sisa suatu pembagian dari bilangan tertentu yang disebut suatu modulus (yaitu residu 5/7 adalah 2). Bentuk penyajian residu suatu jumlah x yang diatur pada suatu bentuk residu { r1, r2,..., rk} berkenaan dengan modulo { m1, m2,…, mk}.

5.2 Latar belakang matematika

Pada bagian ini akan dijelaskan fakta-fakta dasar dan kesimpulan dari CRT. Latar belakang pengetahuan adalah dasar yang sangat penting untuk realisasi efisien dari dekripsi RSA.

Theorem 1 (Chinese Remainder Theorem) *Terdapat bilangan-bilangan $n_1,n_2,...,n_k$ adalah bilangan bulat positif di mana relatif prima pada pasangan. Contohnya :gcd(n_i,n_j) = 1 di mana $i \neq j$. Lebih jauh lagi,*
$n = n_1n_2...n_k$ dan $x_1, x_2;..., x_k$ adalah bilangan bulat. Maka sistem kongruen

$$\begin{aligned} x &\equiv x_1 \bmod n_1 \\ x &\equiv x_2 \bmod n_2 \\ &\vdots \\ x &\equiv x_k \bmod n_k \end{aligned}$$

memiliki solusi yang simultan pada semua kongruen dan dua solusi apapun adalah saling kongruen modulo. Lebih jauh lagi terdapat tepatnya satu solusi antara 0 dan n-1.

Solusi unik dari kongruen simultan memenuhi 0 <= x <= n dapat dihitung dengan :

$$\begin{aligned} x &= \left(\sum_{i=1}^{k} x_i r_i s_i \right) \bmod n \\ &= (x_1 r_1 s_1 + x_2 r_2 s_2 + \ldots + x_k r_k s_k) \bmod n \end{aligned}$$

Persamaan 1

di mana $r_i = \frac{n}{n_i}$ dan $s_i = r_i^{-1} \bmod n_i$ untuk i = 1,2,....,k.

Jika bilangan bulat n_1, n_2,…, n_k adalah pasangan relatif prima dan $n = n_1n_2…n_k$, maka untuk semua bilangan bulat
a, b pasti akan valid di mana a ≡ b mod n jika dan hanya jika a ≡ b mod n_i untuk setiap i = 1, 2,...,k..

Sebagai konsekuensi dari CRT, setiap bilangan bulat positif a < n dapat direpresentasikan secara unik sebagai sebuah k-tuple [a_1, a_2, ...,a_k] dan sebaliknya. Di mana a_i menunjukan sisa / residu a mod n_i untuk setiap i = 1, 2, …,k. Konversi a menjadi sistem residu didefinisikan dengan n_1, n_2, ..., n_k dilakukan secara sederhana dengan reduksi modular a mod n^i. Konversi balik dari representasi sisa menjadi "angka-angka standar" adalah lebih sulit seperti yang dibutuhkan dalam kalkulasi pada persamaan di atas.

5.3 Algoritma dan Struktur

Kelebihan Chinese Remainder Theorem adalah sebagai berikut :

- Mempercepat untuk operasi kunci pribadi (dekripsi, pemberian tandatangan digital).
- Dua n/2-bit eksponensials mod P dan mod Q. sebagai ganti satu n-bit eksponensial mod N (N=P*Q).
- Split n-bit multiplier hardware ke dalam dua n/2-bit pengali, melaksanakan n/2-bit eksponensials paralel.
- Kombinasi hasil menurut CRT.
- CRT meningkat/kan decryption melewati suatu faktor aproksimasi 3- 3.5.

Kalkulasi penting di dalam rencana enkripsi RSA adalah eksponensial modular $M= E^d$ (Mod n). Ini dilakukan setiap kali bagian dari pesan dilakukan enkripsi/dekripsi. d dan n adalah bilangan bulat yang sangat besar, oleh karena itu operasi ini sangat *computationaly intensive*. Sehingga harus ditemukan alternatif metoda biner untuk eksponensial modular.

Keuntungan dasar dengan menggunakan Chinese Remainder Theorem adalah memungkinkan untuk membagi modulo eksponensial yang besar ke dalam dua eksponensial yang jauh lebih kecil, satu di atas p dan satu di atas q. Dua modulo ini adalah faktor utama dari n yang dikenali. Kemudian masalah mengurangi ukuran dengan penggunaan teoreme Fermat's yang lebih kecil. Metoda ini pertama diusulkan oleh Quisquater dan Couvreur.

Jika kita menggunakan penyajian residu { r1,r2,…, rk} pada x, CRT memungkinkan untuk menentukan | x| yang disajikan faktor umum terbesar mengenai seluruh pembawa modulo 1 (yaitu (r_i, r_j)= 1, i ≠ j). Modulo dikenal sebagai operasi memasangkan bilangan prima secara relatif (M. Shahkarami Dan G.A. Jullien, 2002, Universitas Calgary).

Hal penting yang terdapat pada bagian ini adalah bahwa jika jumlah x dibagi ke dalam bentuk residunya, operasi pada residu itu dapat dilaksanakan bebas tiap satu dan lainnya. Sekali ketika menyelesaikan proses residu, jawaban akhir menggunakan CRT dapat direkonstruksi. Diketahui bahwa residu x adalah jauh lebih kecil dari x dirinya sendiri, oleh karena itu operasi individu akan memiliki kompleksitas yang jauh lebih kecil.

Membagi operasi modulo adalah dua kalkulasi mandiri. Sebagai ganti dilakukannya eksponensial (mod n), x dibagi ke dalam (mod p) dan (mod q). Karena kedua-duanya p dan q adalah utama, mereka mencukupi kebutuhan untuk merekonstruksi menggunakan CRT. Hal tersisa yang harus dilakukan kini tinggal melakukan eksponensial keduanya sebagai berikut:

$M_p = E^d \pmod p$
$M_q = E^d \pmod q$
Kita kemudian bisa menerapkan CRT untuk merekonstruksi pesan akhir dari Mp dan Mq.

Algoritma dari CRT dengan $x_1,…,x_k$ sebagai residu dan moduli $n_1,…,n_k$ sebagai input dan mengomputasikan x, maka
solusi dari sistemnya adalah :

INPUT : $x_1,…,x_k, n_1,…,n_k;$
OUTPUT : x memeriksa $x = x_i \bmod m_i$ untuk i = 1 sampai k;

x <- 0
n <- n_1
for i =2 to k do
    n <- n * n_i
for i = 1 to k do
    N_i <- n/n_i;
    y_i <- $N_i^{-1} \bmod n_i$;
    G <- y_i * M_i mod m;
    G <- G * x_i mod m;
    X <- x + G mod m;
Return (x);

5.3.1 Kasus khusus untuk N = PQ

Kasus khusus dari CRT bila modulus adalah hasil dari dua bilangan prima : N = PQ. Jika ingin dikomputasikan M menjadi seperti M = M_P mod P dan M = M_Q mod Q maka :

Input: M_P ,M_Q, P,Q,N;
Output: M;

Y <- Q^{-1} mod P;
M <- y * Q mod N;
M <- M * M_P mod N;

y <- P^{-1} mod Q;
G <- y * P mod N;
G <- G * M_Q mod N;
M <- M + G;
return(M);

Pada langkah 1 dan 4 telah dikomputasikan dua invers dari bilangan bulat n/2-bit dan empat perkalian dari bilangan bulat n-bit pada langkah 2, 3, 5 dan 6. Setiap invers sama dengan 20 perkalian dari bilangan bulat n/2-Bit. Sehingga akhirnya diperoleh kompleksitas dari $28n^2 + o(n^2)$ dan sebuah pemanfaatan memoru dari 4n (parameter sistem) + 3n/2 (akumulator).

5.4 Sistem bilangan kongruen modulo

Jika terdapat persamaan a $\equiv$ b mod n, maka dikatakan *'a congruent terhadap b mod n'* bila selisih a-b dapat dibagi oleh n sehingga : n | (a -b) atau a – b = k.n.

Beberapa sifat dari *congruent* adalah :

- *a $\equiv$ b mod n jika dan hanya jika a dan b menghasilkan sisa yang sama jika dibagi dengan n*
- *reflexive, jika a $\equiv$ a mod n*
- *symmetry, jika a $\equiv$ b mod n, maka b $\equiv$ a mod n*
- *transive, jika a $\equiv$ b mod n dan b $\equiv$ c mod n, maka a $\equiv$ c mod n*
- *jika a $\equiv$ a_1 mod n dan b $\equiv$ b_1 mod n, maka a + b $\equiv$ a_1 + b_1 (mod n)*

Kelas ekivalen dari bilangan bulat adalah semua bilangan bulat yang *congruent* ke a modulo n. Sedangkan bilangan bulat modulo n, dinyatakan Zn, adalah set dari kelas bilangan bulat ekivalen. Penjumlahan, pengurangan dan perkalian berlaku pada himpunan ini.

Contoh :Z_{25} ={0,1,2,....,24}
Pada operasi modular eksponensial, $M=C^d$ mod n intinya adalah perulangan *modular multiflication*, nilai M tidak akan lebih dari n - 1.

5.5 Implementasi CRT untuk mempercepat RSA

5.5.1 Single Radix Conversion (SRC)

Implementasi CRT untuk mempercepat kriptografi RSA tidak hanya dilakukan dengan membagi kode pesan menjadi dua bagiantetapi dapat juga menggunakan satu langkah konversi dinamakan SRC. Berikut adalah langkah-langkah untuk SRC berlaku pula bagi RSA.

Langkah 1: membagi eksponensial kedalam bentuk (mod p) dan (mod q).
$M_p = E^d \pmod p$
$M_q = E^d \pmod q$

Langkah 2: kurangi kompleksitas dengan menerapkan Teorema Fermat's kepada eksponen, yang harus dihitung hanyalah:
$M_p = E^{d_1} \pmod p$
$M_q = E^{d_2} \pmod q$
Dimana :
$d_1 = d \bmod (p - 1)$
$d_2 = d \bmod (q - 1)$
Ukuran d1, dan d2 kini separuh d. Karena kompleksitas tumbuh bersifat exponen terhadap d, hal ini akan menghasilkan tabungan data yang sangat besar.

Langkah 3: gunakan CRT untuk merekonstruksi pesan kita.
$M = M_p (q^{-1} \bmod p)q + M_q (p^{-1} \bmod q)p \pmod n$

5.5.2 Mixed Radix Conversion (MRC)

Decryption menggunakan MRC serupa dengan SRC . Satu-Satunya perbedaan untuk aplikasi ini adalah di dalam perhitungan rekonstruksi akhir.

Langkah 1: Bagi eksponensial ke dalam (mod p) dan (mod q). Juga menerapkan Teorema Fermat's untuk mengurangi eksponen.
$M_p = E^{d_1} \pmod p$
$M_q = E^{d_2} \pmod q$

Langkah 2: Rekonstruksi pesan menggunakan cara sebagai berikut :
$M = M_p + [(M_q - M_p) \times (p^{-1} \bmod q) \bmod q] \times p$
atau lebih umum menggunakan :
$M = M_q + [(M_p - M_q) \times (q^{-1} \bmod p) \bmod p] \times q$

5.5.3 Dekripsi RSA menggunakan CRT

Untuk mempercepat operasi dekripsi digunakan *Metoda Chinese Remainder Theorem*. Metode ini menggunakan faktor prima *mod n*, yaitu p dan q karena $n = p.q$, yang menyatakan sebagai berikut :
Agar $m_1, m_2,...,m_r$ relatif prima, maka sistem kongruen
$X \equiv a_1 \bmod m_1$
$X \equiv a_2 \bmod m_2$
$X \equiv a_3 \bmod m_3$
mempunyai solusi unik untuk modulo $M = m_1, m_2, ..., m_r$.

Dikarenakan p dan q adalah bilangan prima, pesan apapun akan direpresentasikan secara unik dengan tuple $[M_P;M_Q]$, di mana $M_P = M$

mod P dan M_Q = M mod Q. Oleh karena itu, dimungkinkan juga untuk memperoleh nilai tersebut dengan komputasi M_P;M_Q dan mengkombinasikan ulang sesuai dengan persamaan 1 di atas, daripada komputasi biasa menggunakan $M = C^D \bmod N$.

$$\begin{aligned} M_P &= M \bmod P = (C^D \bmod N) \bmod P \\ &= C^D \bmod P \ \ (\text{since } N = PQ) \\ &= C^{D \bmod (P-1)} \bmod P \\ &= C^{D_P} \bmod P \text{ with } D_P = D \bmod (P-1) \end{aligned}$$

Persamaan 2

Lebih jauh lagi, akan mudah untuk diobservasi bahwa modulo P chiperteks C dapat dikurangi sebelum melakukan komputasi M_P, sehingga panjang dari semua operan dapat dikurangi menjadi setengahnya. Dengan persamaan C_P = C mod P dan C_Q = C mod Q, sama seperti D_P = D mod (P - 1) dan D_Q = D mod (Q - 1), sehingga didapat persamaan di bawah ini untuk M_P dan M_Q :

$$M_P = {C_P}^{D_P} \bmod P \text{ and } M_Q = {C_Q}^{D_Q} \bmod Q$$

Persamaan 3

Kombinasi dari M_P dan M_Q untuk mendapatkan M dapat dilakukan dengan menggunakan persamaan 1. Untuk kasus khusus untuk k = 2, n_1 = P, n_2 = Q dan n = N = PQ, didapatkan r_1 = N/P = Q dan r_2 = N/Q = P. Lebih jauh lagi, persamaan 1 dapat disederhanakan menggunakan teorema Fermat menjadi:

$$\begin{aligned} M &= \left(M_P Q \left(Q^{-1} \bmod P\right) + M_Q P \left(P^{-1} \bmod Q\right)\right) \bmod N \\ &= \left(M_P Q \left(Q^{P-2} \bmod P\right) + M_Q P \left(P^{Q-2} \bmod Q\right)\right) \bmod N \\ &= \left(M_P \left(Q^{P-1} \bmod N\right) + M_Q \left(P^{Q-1} \bmod N\right)\right) \bmod N \end{aligned} \quad (9)$$

Persamaan 4

Persamaan 4 di atas muncul dari fakta bahwa a (b mod c) = (ab) mod (ac) untuk semua bilangan bulat positif a, b, c. Catatan : koefisien Q^{P-1} mod N dan P^{Q-1} mod N adalah konstan dan dapat di prekomputasi, sehingga usaha mengkombinasikan M_P dan M_Q dikurangi menjadi dua perkalian, satu sebagai penambahan dan satu sebagai pengurangan modulo N.

Ketika mengasumsikan bahwa eksponen D_P = D mod (P - 1) dan D_Q = D mod (Q - 1), sama seperti konstanta di mana diperlukan untuk rekombinasi $R_P = Q^{P-1}$ mod N dan $R_Q = P^{Q-1}$ mod N sudah diprekomputasikan, basis CRT untuk dekripsi RSA dapat dilakukan sesuai dengan langkah-langkah berikut:

1. Hitung C_P = C mod P dan C_Q = C mod Q.
2. Hitung nilai eksponen $M_P = {C_P}^{DP}$ mod P and $M_Q = {C_Q}^{DQ}$ mod Q.
3. Hitung koefisien $S_P = M_P R_P$ mod N dan $S_Q = M_Q R_Q$ mod N.
4. Hitung $M = S_P + S_Q$. Jika M >= N maka hitung
5. M = M - N.

Sekarang dapat dilihat secara jelas bahwa reduksi awal (langkah 1) dan kombinasi (langkah 3 dan 4) tidak mengakibatkan perubahan usaha komputasi yang signifikan bila dibandingkan dengan perhitungan pada langkah kedua. Dua eksponensial (langkah 2) dapat dikomputasikan secara independen satu sama lain dan secara paralel. Dibandingkan dengan dekripsi non-CRT pada n bit hardware, setiap eksponen CRT 4 kali lebih cepat jika n/2 bit hardware digunakan. Peningkatan kecepatan yang dramatis ini disebabkan pengurangan panjang 50 %, baik dari eksponen dan dari modulus. Melakukan dua eksponen seperti langkah 2 secara paralel membutuhkan dua n/2 bit pengali modular, menghasilkan sebuah faktor pemercepat 4 - ε, dengan ε dihitung untuk langkah 1, 2, dan 4.

6 Performansi dan Perbandingan

6.1 Performansi dan perbandingan arsitektur konvensional dan optimisasi

Dua arsitektur tersebut terdapat pada VHDL dan diimplementasikan pada FPGA. Sebuah perbandingan di antara mereka dibuat dalam frekuensi dan *chip covered Area*, seperti yang diperlihatkan pada tabel di bawah ini :

reference	Conventional	Optimized	Reduction Percentage
Function Generators	130946	65408	50%
CLB's	65473	48767	25.5%
Flip Flops	97534	97534	0%
Clock Frequency	156.3 MHz	168.7MHz	7.35%

Tabel 3 : Tabel implementasi dari kedua arsitektur pada 128 bit FPGA

Tabel di atas adalah hasil pembuktian dari versi modifikasi fungsi MMCh dari algoritma perkalian Montgomery yang dioptimisasi dari fungsi MMcs dan MM seperti yang telah dijelaskan di bagian atas. Sekarang jelas bahwa area tutupan dari fungsi MMCh menggunakan 50% lebih kecil dari fungsi generator dan sekitar 25% lebih kecil daripada CLB dan dalam frekuensi *clock* dari fungsi MMCh adalah lebih cepat 7.35%. Hal ini adalah sebuah peningkatan pada saat melakukan perhitungan pada perkalian yang merupakan bagian dari modul perpangkatan RSA. Pada

kasus-kasus area kecil, frekuensi *clock* yang tinggi adalah faktor yang sangat penting bagi efisiensi modul.

reference	Multiplication Time	Clock Frequency (MHz)
Optimized Architecture	5.12 nsec	168.7
Ors et al. [5]	3.974 μsec	97.63
Nedjah [6]	23 nsec	43.4
Daly [7]	11.2 nsec	88.74

Tabel 4 : Perbandingan dari 128 bit perkalian Montgomery

Seperti perbandingan dengan arsitektur perkalian Montgomery yang lain, seperti yang diperlihatkan pada tabel 4 di atas, arsitektur optimisasi memberikan hasil yang hebat dalam frekuensi *clock* dan waktu perkalian. Arsitektur optimisasi lebih cepat daripada semua arsitektur yang dibandingkan yang ditunjukan pada tabel 4 di atas. Walaupun demikian, semua hasil di atas datang dalam keluaran yang besar pada nilai *high chip covered area*.

6.2 Perbandingan Kecepatan

Library yang digunakan sudah terlindungi dari *timing attacks*. Versi asli dari perkalian Montgomery selalu melakukan pembagian setelah perkalian dan dipilih untuk menghasilkan nilai dari pembagian jika lebih besar dari nol, dalam kata lain hasil tersebut tetap dibiarkan tidak berubah. Sebuah modifikasi dibuat untuk menghindari *timing attack* dengan menambahkan putaran untuk mendapatkan *timing* yang sama ketika hasil dari pembagian harus dibuang.

Value	This paper	Walter's version
q_i	$C_A + 2C_M$	$C_A + 2C_M$
S	$(2C_A + 2C_M)p + 2C_A + 2C_{M'}$	$(2C_A + 2C_M)p + 2(2C_A + 2C_{M'})$

Tabel 5 : Formula untuk memprediksi jumlah putaran *clock* untuk versi algoritma yang berbeda

Size of N	This paper $(14p+6+13)(p+1)$	Walter's version $(14p+12+13)(p+2)$
512 bits ($p=16$)	8.5 %	17.7 %
768 bits ($p=24$)	5.6 %	11.7 %
1024 bits ($p=32$)	4.2 %	8.8 %
2048 bits ($p=64$)	2.1 %	4.4 %

Tabel 6 : Tabel untuk memprediksi pertambahan waktu untuk perkalian (C_A=1 dan C_m=6) relatif terhadap standar pengurangan modular ((14p + 14)p)

Walaupun dengan menambahkan putaran yang berasal dari loop kosong, hal ini tidak melindungi dari *power attacks*. Jika dibandingkan hasil yang diprediksi dari tabel 6 dan hasilnya pada tabel 1, dapat dilihat beberapa perbedaan. Hal ini dapat dikatakan normal karena fakta-fakta berikut ini :

- Prediksi dibuat atas dasar satu perkalian dan didapat hasil eksponen yang lengkap tanpa membutuhkan waktu tambahan pada perhitungan.
- Ini adalah model dasar (tidak ada operasi memori yang dilakukan pada perhitungan)

Catatan penting dari hal ini adalah peningkatan akan jauh lebih besar jika digunakan arsitektur CPU di mana perkalian dilakukan dalam waktu konstan apapun nilai dari operan tersebut. Anggap waktu yang dibutuhkan untuk perkalian adalah sama dengan waktu penambahan dan sama dengan satu putaran *clock*, sehingga hasilnya akan diketahui pada tabel 2.

6.3 Analisis Performansi dari perkalian Montgomery

Modular multiflication a, b mod n dengan cara klasik dilakukan perkalian dengan *problem size* k^2 kemudian diikuti oleh operasi reduksi modulo n, *n = bilangan bulat k bit*, dengan *problem size k(k + c)* dengan suatu nilai c > 1.

Dengan demikian *problem size* untuk *modular multiflication* klasik sama dengan $2k^2 + k$.c . *Modular exponential* dengan metode klasik ini dieksekusi dengan *problem size*.
$log_2 d\ (2k^2 + k.c)$
atau
$2k^2 + k^2 c$
untuk proses dekripsi RSA menggunakan kunci privat *dk* bit.

Dengan menggunakan *problem size* $2k^2 + k$, maka *problem size* modular exponential sama dengan
$log_2 d(2k^2 + k)$
atau
$2k^3 + k^2$
untuk proses dekripsi RSA menggunakan kunci privat *dk* bit.

Kesimpulan , metode *montgomery exponential* dapat lebih efisien dalam waktu eksekusi $\Delta\ tc$ adalah :
$\Delta\ tc = sk^3 + k^2C - 2k^3 - k^2$
$= k^2 (c - 1)$

6.4 Analisis Performansi dari CRT

- **Notation:**

n adalah panjang bit dari $m = \Pi^{k}_{i=0}\ m_i$ dan diasumsikan bahwa panjang bit dari setiap m_i adalah n/k.

- **Estimasi Cost:**

Pada langkah ke-3 dari algoritma CRT biasa, dikomputasikan $m = \Pi^{k}_{i=0}\ n_i$ dam dilakukan perkalian $(k - 1)$ pada Z dari bilangan bulat n/k-bit. Sehingga total dari biaya $(k - 1)n^2/k^2$. Pembagian pada tahap ke-4 bagian yang pertama dalam algoritma CRT adalah pembagian sederhana dari Z. Panjang dari bit m dan m_i merujuk pada n dan n/k. Oleh karena itu setiap pembagian memiliki *cost* $(n^2 + 2kn)(k - 1)/k^2$. Sehingga total pembagian yang dimiliki ada k dan *cost* dari langkah 4 adalah $(n^2 + 2kn)(k - 1)/k$. Pada langkah 4 bagian kedua dikumputasikan setiap waktu dari modulo invers m_i. Hal ini sama dengan dua puluh kali perkalian dari bilangan bulat n/k-bit dan dilakukan dalam k kali: $40(n^2/k + n)$.

Pada langkah 4 bagian ketiga dan keempat, dikomputasikan perkalian dari bilangan bulat n-bit dan dilakukan dalam k kali. Sehingga totalnya, memiliki *cost* $4k(n^2 + n)$. Sifat asymptotik: $n^2 (4k + 1 + 40/k - 1/k^2) + o(n^2)$

- **Total *Cost***

Sistem parameter

Register names	Bits	Number of registers
a_i	n/k	k
m_i	n/k	k
a	n	1
Subtotal	$3n$ bits	

Accumulator

Register names	Bits	Number of registers
m, G	n	2
M_i	n/k	k
y_i	n/k	k
Subtotal	$4n$ bits	

Total *Cost* Memory : 7n bits.

Metode CRT memecah operasi *modular exponential* dengan eksponen *dk* bit dengan 2 operasi *modular exponential* dengan 2 eksponen *dp* dan *dq* yang masing-masing berukuran *k/2* bit, sehingga *problem size* operasi dekripsi menggunakan metode CRT adalah :

$$\frac{2(2\,(k/2)^3 + (k/2)^2 C)}{4} = \frac{2k^3 + 2k^2 C}{4}$$

Dengan membandingkan dua persamaan di atas untuk proses dekripsi dengan kunci privat *d,* metode CRT mengurangi *problem size* dengan faktor pembagi hampir 4.

6.5 Perbandingan Kecepatan pada berbagai Implementasi RSA

	Speed	Speed-up for n=1024
RSA without CRT	$3n^3 + n^2$	1.0
RSA with CRT	$3n^3/4 + 7n^2/2 + o(n^2)$	3.98
Batch RSA, $p = 4$, $k = 16$	$3n^3/4 + 918n^2 + o(n^2)$	7.29
RSA modulo PQR	$n^3/3 + 19n^2/3 + o(n^2)$	8.84
RSA modulo P^2Q, $k = 16$	$2n^3/9 + n^2(4k/3 + 6) + o(n^2)$	12.06
Rebalanced RSA, $k = 160$	$n^2(3k/2 + 2) + o(n^2)$	12.70

Tabel 7 : Tabel Perbandingan kecepatan berbagai implementasi RSA

Pada tabel di atas dibandingkan berbagai macam algoritma RSA biasa dengan atau tanpa CRT yang dijelaskan dalam makalah ini. Selain itu juga dibandingkan dengan implementasi RSA jenis lain hasil dari studi pustaka yang tidak dijelaskan dalam makalah ini. Implementasi RSA lain yang ikut dibandingkan adalah *rebalanced* RSA, *Multi-Prime* RSA, *Multi-Power* RSA, dan *batch* RSA. Setiap algoritma memiliki domain aplikasi yang berbeda-beda sehingga harus dipertimbangkan kelebihan dan kekurangan dari masing-masing algoritma. Walaupun *rebalanced* RSA tampak paling cepat, namun proses enkripsi yang dilakukan algoritma ini jauh lebih lambat. Oleh karena itu algoritma ini hanya didesain untuk tujuan dekripsi. *Batch* RSA tidak secepat seperti *Multi-Prime and Multi-Power* RSA, tetapi faktanya algoritma – algoritma ini dapat dikombinasikan sehingga menambah kecepatan dengan mengorbankan memori yang digunakan. RSA modulo P^2Q tidak hanya cepat, tetapi juga tidak memiliki kekurangan yang banyak.

6.6 Perbandingan Memori pada berbagai Implementasi RSA

	Total Memory	System Parameters	Accumulators
RSA without CRT	$4n$ bits	$4n$ bits	0 bit
n=1024 bits	4096 bits	4096 bits	0 bits
RSA with CRT	$8n$ bits	$11n/2$ bits	$5n/2$ bits
n=1024 bits	8192 bits	5632 bits	2560 bits
Rebalanced RSA	$7n + 2k$ bits	$9n/2 + 2k$ bits	$5n/2$ bits
n=1024, k=160	7200 bits	4640 bits	2560 bits
RSA modulo PQR	$26n/3$ bits	$17n/3$ bits	$3n$ bits
n=1024 bits	8875 bits	5803 bits	3072 bits
RSA modulo P^2Q	$25n/3$ bits	$5n$ bits	$10n/3$ bits
n=1024 bits	8534 bits	5120 bits	3414 bits
Batch RSA (p=4)	$67n/2 + 784$ bits	$13n + 64$ bits	$41n/2 + 720$ bits
n=1024 , k=16	35088 bits	13376 bits	21712 bits

Tabel 8 : Tabel Perbandingan memori berbagai implementasi RSA

Walaupun dekripsi *rebalanced* RSA membutuhkan memori yang lebih sedikit daripada algoritma yang lainnya, proses enkripsi dari algoritma ini sangat lambat. Untuk dekripsi 4 *batched*, *batch* RSA membutuhkan sekitar tiga kali memori lebih banyak daripada memori yang digunakan oleh dekripsi algoritma lain. Kebutuhan memori yang begitu besar membuat algoritma ini hanya berguna pada lingkungan tanpa batasan memori, seperti pada server SSL. *Multi-Prime* dan *Multi-Power* RSA memiliki kebutuhan memori yang hampir sama, tetapi *Multi-Power* RSA masih sedikit lebih baik.

7 Kesimpulan

Dengan uraian-uraian di atas maka dapat diambil beberapa kesimpulan, yaitu

1. Metode CRT mereduksi modular eksponensial dengan menghasilkan dua *sub-keys* dengan ukuran masing-masing 512 bit dari kunci 1024 dengan faktor peningkatan kecepatan secara teoritis = 4.
2. Metode perkalian *montgomery* mereduksi perkalian modular yang dilakukan berulang-ulang dalam modular eksponensial secara teoritis, $\Delta tc = k^2(c - 1)$
3. *Montgomery* dengan mod n dilakukan dengan 3 perkalian standar dan jika hasilnya bukan set bilangan bulat modulo n, n dikurangkan dari hasil perkalian tersebut
4. Hasil dari perbandingan dua arsitektur perkalian Montgomery dalam kecepatan *Clock* dan Area *Chip Covered* membuktikan bahwa arsitektur optimisasi lebih baik sehingga hal ini menunjukan optimisasi yang baik dari algoritma konvensional. Oleh karena itu arsitektur ini sangat menguntungkan.

8 Referensi

[1] Munir, Rinaldi. (2006). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung

[2] Schneier, Bruce. (1996). Applied Cryptography 2nd. John Wiley & Sons.

[3] Adnan,23299001, Perancangan Pengali Integer Multi-presisi untuk Sistem Kriptographi RSA dengan Metode Montgomery Multiplication,Teknik Elektro ITB, Bandung, 2001

[4] Großsch¨adl, Johann. The Chinese Remainder Theorem and its Applicationin a High-Speed RSA Crypto Chip. Graz University of Technology Institute for Applied Information Processing and Communications. Austria

[5] Siddika Berna Ors, Lejla Batina, Bart Preneel and Joos Vandewalle. (2003). "Hardware Implementation of a Montgomery Modular Multiplier in a Systolic Array". *in proc. ofInternational Parallel and Distributed Processingsymposium*.

[6] Nadia Nedjah and Luiza de Macedo Mourelle.(2002). "Two Hardware Implementations for the Montgomery Modular Multiplication: Sequential versus Parallel", *in proc. of 15th Symposium on Integrated Circuits and System Design*.

[7] Alan Daly and William Marnane. (2002). "Efficient Architectures for implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic", *in proc.of 10th International symposium on Field-programmable gate arrays*.

[8] A. P. Fournaris dan O. Koufopavlou. Montgomery Modular Multipliers Architectures And Hardware Implementations For An RSA Cryptosystem. Electrical and Computer Engineering Department University of Patras, Patras, Greece

[9] Cetin Kaya Koc1 dan Tolga Acar. Analyzing and Comparing Montgomery Multiplication Algorithms. Department of Electrical & Computer Engineering Oregon State University Corvallis, Oregon

[10] Kaliski Jr, Burton S. Analyzing and Comparing Montgomery Multiplication Algorithms. RSA Laboratories

[11] Gael Hachez and Jean-Jacques Quisquater. Montgomery Exponentiation with no Final Subtractions: Improved Results. Universit_e Catholique de Louvain, UCL Crypto Group

[12] Vuillaume, Camille. (2003). Efficiency Comparison of Several RSA Variants. Fachbereich Informatik der TU-Darmstadt