Optimasi Algoritma RSA dengan Menggunakan Pembangkit Bilangan Acak ISAAC

Shauma Hayyu Syakura (13507025)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*if17025@students.if.itb.ac.id*

***Abstrak*— Seperti yang kita ketahui, tingkat keamanan algoritma RSA bergantung pada panjang kunci dan kunci itu sendiri. Untuk mendapatkan kunci bilangan prima yang besar, sulit untuk membuatnya sendiri, oleh karena itu digunakan pembangkit bilangan acak untuk menghasilkan kunci tersebut. Namun pembangkit bilangan random konvensional yang tersedia pada program untuk pengembangan seperti Java, C#, dan lain-lain, memiliki masalah tersendiri. Bilangan acak yang dihasilkan sebenarnya adalah bilangan pseudorandom (tidak benar-benar acak), dan bilangan-bilangan tersebut akan muncul lagi secara berulang setelah periode tertentu, sehingga mengurangi keamanan dari bilangan random tersebut. Oleh karena itu, pembangkit bilangan random khusus untuk kriptografi dibuat, dan pembangkit ini disebut *Cryptographically Secure Pseudorandom Generator*. ISAAC adalah sebuah *Cryptographic Pseudorandom Generator* yang dikenal cepat. ISAAC merupakan singkatan dari *Indirect, Shift, Accumulate, Add,* dan *Count*. Dalam makalah ini, akan dilakukan studi dan analisis optimasi RSA dengan menggunakan ISAAC. Pseudorandom generator ini diharapkan dapat menghasilkan RSA yang lebih aman dan cepat.**

***Index Terms*—Algoritma Kunci Publik, RSA, *pseudorandom generator*, ISAAC, optimasi**

I. PENDAHULUAN

Kriptografi adalah ilmu untuk menyamarkan pesan agar pesan tersebut hanya diketahui oleh pihak pemberi pesan dan pihak penerima pesan, dan tidak diketahui oleh pihak lain. Pesan disamarkan dengan cara mengganti pesan asli menjadi pesan lain yang hanya bisa dimengerti oleh pihak pemberi pesan dan pihak penerime pesan. Pesan yang disamarkan tersebut, dapat diubah kembali menjadi pesan asli dengan menggunakan kunci atau cara-cara tertentu.

Kriptografi sudah digunakan sejak zaman dahulu, salah satu penggunaannya adalah untuk menyampaikan pesan saat perang. Pengiriman pesan dalam sebuah peperangan biasanya menggunakan kriptografi agar pesan rahasia tersebut tidak dapat dibaca dan diketahui oleh pihak musuh. Kriptografi dalam peperangan mulai digunakan sejak zaman pemerintahan Julius Caesar yang terkenal dengan Caesar Chiper-nya hingga Perang Dunia II.

Kriptografi pada zaman tersebut masih menggunakan cara manual, seperti menggunakan *cryptex* atau mesin. Namun sejak lahirnya komputer dan dimulainya era digital, kriptografi mulai berkembang pesat, berbagai macam algoritma kriptografi untuk menyembunyikan pesan dibuat. Dan penggunaannya tidak lagi hanya untuk menyampaikan pesan dalam peperangan, karena saat ini, privasi dalam penyampaian pesan dan penyimpanan data menjadi penting. Oleh karena itu, saat ini kriptografi juga digunakan dalam penyampaian pesan-pesan digital, seperti e-mail, sms, atau chatting, dan juga untuk penyimpanan data-data penting seperti *password* (kata kunci), tanda tangan digital, dan lain-lain.

Ada dua algoritma kriptografi digital yang berkembang hingga saat ini, yaitu algoritma kunci simetri dan algoritma kunci publik. Proses algoritma kriptografi terdiri dari proses enkripsi dan dekripsi. Proses enkripsi adalah proses pengubahan pesan asli (disebut *plaintext*) menjadi pesan yang tersamarkan (disebut *chipertext*), sedangkan proses dekripsi adalah proses sebaliknya.

Algoritma kunci simetri adalah algoritma kriptografi yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi pesan. Kunci tersebut hanya boleh diketahui oleh pihak pemberi dan penerima pesan. Beberapa contoh algoritma yang menggunakan kunci simetri adalah algoritma *vigenere chiper*, algoritma block chiper, dan *one time pad*.

Sedang algoritma kunci publik adalah algoritma yang memiliki 2 buah kunci, yaitu kunci publik dan kunci privat. Kunci publik digunakan untuk mengenkripsi pesan, dan dapat diketahui oleh siapa saja. Sedangkan kunci privat digunakan untuk mendekripsi pesan dan tidak boleh diketahui oleh siapapun, hanya pemberi dan penerima pesan. Beberapa contoh algoritma yang menggunakan kunci publik adalah algoritma RSA, ElGamal dan DSA.

Salah satu algoritma kunci publik yang populer digunakan adalah Algoritma RSA. Algoritma RSA banyak digunakan untuk pengamanan pesan dan pembuatan tanda tangan digital (*digital signature*). Keamanannya terletak pada sulitnya memfaktorkan bilangan prima yang menjadi kunci dari algoritma tersebut.

Keamanannya juga terletak pada besarnya bilangan prima yang menjadi kunci.

Karena bilangan prima yang dibutuhkan besar dan tidak mungkin untuk dimasukkan secara manual, oleh karena itu dibutuhkan pembangkit bilangan acak untuk menghasilkan bilangan prima yang besar tersebut.

Namun, pembangkit bilangan acak biasa memiliki kelemahan, karena sebenarnya pembangkit tersebut merupakan pembangkit bilangan acak semu (atau disebut dengan *pseudorandom*). Hal tersebut dikarenakan pembangkit bilangan tersebut menghasilkan bilangan-bilangan yang sama secara periodik. Hal ini tentu akan mengurangi keamanan dari kunci bilangan acak yang digunakan, sebab kriptanalis dapat lebih mudah menemukan bilangan acak yang menjadi kunci.

Oleh karena itu, dibuatlah algoritma-algoritma pembangkit bilangan acak untuk mengurangi kelemahan tersebut, frekuensi bilangan sama yang muncul secara periodik dikurangi. Dan pembangkit bilangan acak tersebut disebut dengan *cryptographycally secure pseudorandom generator*.

Salah satu pseudorandom generator yang terkenal adalah ISAAC. ISAAC dibuat oleh Robert J. Jenkins Jr. ISAAC adalah pembangkit bilangan acak yang terkenal dengan kecepatannya, persebarannya merata, tidak bisa ditebak, dan aman secara kriptografi. Algoritma ISAAC mirip dengan algoritma pembangkit bilangan acak RC4, namun ISAAC tiga kali lebih cepat[1].

Dengan melihat kelebihan-kelebihan ISAAC, diharapkan kelebihan tersebut dapat digunakan untuk melakukan optimasi terhadap algoritma RSA dalam pembangkitan bilangan prima acak sebagai kunci. Dengan ISAAC diharapkan didapatkan bilangan kunci yang lebih aman dari bilangan kunci yang dihasilkan dari pembangkit bilangan kunci biasa.

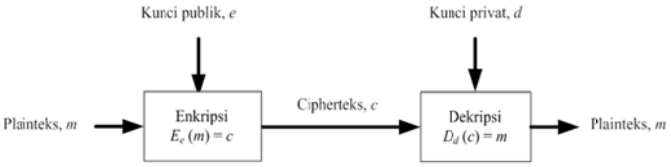
Algoritma RSA sendiri membutuhkan komputasi yang besar karena modulasi dan perpangkatan bilangan-bilangan besar dalam proses enkripsi dan dekripsinya. Komputasi yang digunakan ISAAC lebih rendah dari algoritma pembangkit bilangan acak biasa, sehingga dapat lebih meningkatkan performansi dari algoritma RSA dan menurunkan biaya komputasi.

II. ALGRORITMA KUNCI PUBLIK

Algoritma kunci publik adalah algoritma yang menggunakan kunci asimetri, yaitu kunci yang digunakan untuk mengenkripsi dan mendekripsi pesan berbeda. Terdapat dua jenis kunci yaitu kunci publik dan kunci privat. Pengirim pesan mengenkripsi pesan menggunakan kunci publik, dan kunci publik ini dapat diketahui oleh banyak orang. Namun apabila pengirim pesan ingin mengirimkan pesan tersebut ke penerima pesan, pengirim pesan mengirimkan pesan yang terenkripsi. Kemudian pesan terenkripsi tersebut didekripsi dengan kunci privat yang dimiliki penerima pesan.

Algoritma kunci publik ditemukan Diffie-Helman. Algoritma kunci publik muncul karena terdapat masalah pada algoritma kunci simetri. Masalahnya adalah bagaimana mengirim kunci rahasia kepada penerima pesan? Mengirim kunci melalui jalur pengiriman biasa tidak aman, sedangkan menggunakan jalur kedua umumnya lambat dan sangat mahal.

Prosedur enkripsi dan dekripsi algoritma kunci publik adalah sebagai berikut:



Gambar 1 Prosedur Algoritma Kunci Asimetri

Kunci publik dan pesan terenkripsi dapat dikirimkan melalui saluran tidak aman. Keuntungan algoritma kunci publik adalah tidak perlu melakukan pengiriman kunci rahasia sehingga jumlah kunci dapat ditekan.

Pembangkitn sepasang kunci pada kriptografi kunci publik didasarkan pada persoalan integer yaitu pemfaktoran dan logaritma. Semakin besar bilangan yang harus difaktorkan, semakin sulit faktorisasi yang harus dilakukan dan dibutuhkan waktu yang sangat lama. Contoh algoritma yang memakai prinsip ini adalah RSA. Begitu pula dengan logaritma bilangan besar, semakin sulit bilangannya semakin sulit dan lama komputasinya. Algoritma yang menafaatkan prinsip ini adalah ElGamal dan DSA.

Kelebihan algoritma kunci asimetri adalah hanya kunci privat yang dijaga kerahasiannya, dan tidak ada kebutuhan untuk mengirimkan kunci privat sebagaimana pada algoritma kunci simetri. Kelemahannya adalah proses enkripsi dan dekripsi lebih lambat daripada algoritma kunci simertri, karena enkripsi dan dekripsi menggunakan bilangan besar dan melibatkan operasi perpangkatan bilangan besar. Algoritma kunci publik menggunakan bilangan-bilangan besar sebagai inti dari keamanan algoritma tersebut. Ukuran chiperteks dapat menjadi jauh lebih besar dari plainteks, begitu pula ukuran kuncinya pun lebih besar dari kunci simetri. Oleh karena itu biasanya kunci publik digunakan untuk mengamankan pengiriman kunci simetri dan digunakan untuk memberi tanda tangan digital.

III. CRYPTOGRAPHYCALLY SECURE PSEUDORANDOM GENERATOR

Pembangkit bilangan acak banyak digunakan dalam kriptografi, seperti pembangkitan kunci pada algoritma kunci publik yang membutuhkan bilanga besar, pembangkitan *initialization vector* (IV) pada algoritma kunci simetri, dan sebagainya.

Sebenarnya, algoritma pembangkit bilangan acak yang umum tidak benar-benar menghasilkan deret bilangan acak. Bilangan acak yang dihasilkan sebenarnya menggunakan rumus-rumus matematika, yang disebut dengan bilangan acak semu (*pseudorandom*). Deret bilangan acak

yang dihasilkan dari rumus matematika tidak benar-benar acak karena dapat berulang secara periodik. Pembangkit bilangan acak semacam itu disebut dengan *pseudorandom number generator* (PRNG). Salah satu contoh algoritmanya adalah *Linear Congruental Generator*.

Hal tersebut menyebabkan *pseudorandom number generator* biasa tidak dapat digunakan untuk kriptografi. Kemunculan periodik dari deret bilangan acak tersebut dapat dimanfaatkan oleh kriptanalis untuk menemukan bilangan acak yang digunakan dalam enkripsi atau dekripsi suatu pesan rahasia lebih mudah. Hal ini tentu menyebabkan penggunaan pembangkit bilangan acak semu biasa menjadi tidak benar-benar aman.

Pembangkit bilangan acak yang cocok untuk kriptografi disebut *cryptographycally secure pseudrandom generator* (CSPRNG). Pembangkit bilangan acak yang aman secara kriptografi harus lolos uji keacakan statistik dan ttahan terhadap serangan yang serius, yaitu serangan untuk memprediksi bilangan acak yang dihasilkan. Contoh CSPRNG adalah algoritma BBS (*Blum Blum Shut*).

IV. ALGORITMA RSA

Algoritma RSA adalah algoritma kunci publik paling terkenal dan paling banyak aplikasinya. Ditemukan oleh tiga peneliti MIT yaitu Ron Rivest, Adi Shmir, dan Len Adleman. Keamanannya terletak pada sulitnya menmfaktorkan bilangan-bilangan prima besar.

Proses enkripsi dan dekripsi pesan RSA, terdiri dari pembangkitan kunci terlebih dahulu, setelah itu baru dilakukan proses enkripsi menggunakan kunci publik, dan dekripsi menggunakan kunci privat. Proses enkripsi dan dekripsi filenya mirip dengan blok chiper, yaitu dilakukan per blok, dengan panjang blok sesuai dengan panjang kunci.

Proses pembangkitan kunci RSA adalah sebagai berikut:

1. Pilih 2 buah bilangan prima p dan q.
   Untuk tujuan keamanan, bilangan integer p dan q dipilih secara random, dan harus memiliki panjang bit yang sama. Biasanya panjang bit yang dipilih untuk bilangan p dan q ukurannya besar, agar kunci semakin aman.
2. Hitung n, yaitu $n = pq$.
   Bilangan n akan digunakan sebagai modulus untuk kunci publik dan kunci privat.
3. Hitung $\varphi(n) = (p-1)(q-1)$, dimana φ adalah fungsi totient Euler.
4. Pilih bilangan integer e, dimana $1 < e < \varphi(n)$, fpb(e, $\varphi(n)$) = 1, e dan $\varphi(n)$ adalah *coprime* (e relatif prima terhadap $\varphi(n)$). Bilangan e adalah bilangan yang menjadi kunci publik.
5. Hitung kunci dekripsi d, dengan persamaan:
   $ed \equiv 1 \pmod{\phi(n)}$ atau $d \equiv e^{-1} \bmod (\phi(n))$
6. Dari perhitungan di atas didapatkan kunci publik dan kunci privat:
   - Kunci publik adalah pasangan (e, n)
   - Kunci privat adalah pasangan (d, n)

Proses enkripsi dan dekripsi algoritma RSA:

1. Nyatakan pesan menjadi blok-blok plainteks m_1, m_2, m_3, ..., m_n, dengan syarat $0 < m_i < n-1$.
2. Hitung blok chiperteks c_i untuk blok plainteks p_i dengan persamaan $c_i = m_i^e \bmod n$.
3. Proses dekripsi dilakukan dengan menggunakan persamaan $m_i = c_i^d$ **mod** n.

Kekuatan algoritma RSA terletak pada tingkat kesulitan dalam memfaktorkan bilangan menjadi faktor-faktor prima, dalam hal ini $n = a \times b$. Bila a dan b diketahui, dan karena kunci enkripsi e tidak rahasia, maka kunci dekripsi d dapat diketahui.

Penemu algoritma RSA menyarankan nilai a dan b panjangnya lebih dari 100 digit, dengan demikian hasil $n = ab$ bisa berukuran lebih dari 200 digit. Usaha untuk mencari faktor dari bilangan 200 digit membutuhkan waktu komputasi yang sangat lama, hinga bermilyar-milyar tahun. Oleh karena itu, RSA hanya aman jika nilai n cukup besar.

Kelemahan RSA adalah komputasinya lebih lambar dari pada algroitma kunci simetri seperti DES dan AES. Karena komputasinya yang sangat lambat, RSA dalam praktek tidak digunakan untuk mengenkripsi pesan, melainkan kunci simetri, atau digunakan untuk mengenkripsi pesan yang ukurannya pendek. Selain itu RSA juga digunakan dalam tanda tangan digital, yaitu untuk mengenkripsi *message digest* hasil fungsi *hash* file yang ingin dibuat tanda tangan digitalnya.

V. PEMBANGKIT BILANGAN ACAK ISAAC

ISAAC merupakan algoritma pembangkit bilangan acak yang dirancang oleh Robert J. Jenkins Jr. Jenkins membuat algoritma ISAAC bersaamaan dengan dua algoritma pembangkit bilangan acak lainnya, yaitu IA dan IBAA, dimana ISAAC merupakan hasil dari pengembangan keduanya. ISAAC membutuhkan 18,75 instruksi untuk menghasilkan nilai sebesar 32-bit.

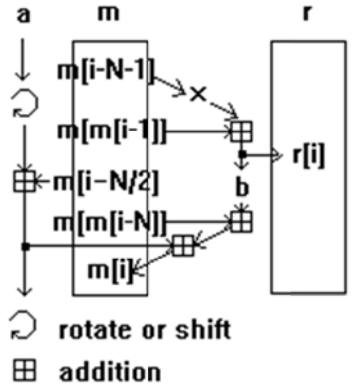
Pembangkit bilangan acak IA (Inderection, Addition), lebih mudah diprediksi, tetapi aman, dan tahan terhadap serangan eliminasi Gaussian. IA didesain dengan menyimpulkan bahwa keadaan internal dari hasilnya tidak dapat ditelusuri, agar kodenya mudah disimpan di memori, dan beroperasi secepat mungkin.

Sedangkan lebih banyak kebutuhan ditambahkan dalam pembuatan IBAA (*Indirection, Barrelshif, Accumulate, Add*). IBAA didesain agar aman secara kriptografi (*cryptographically secure*), tidak dapat diprediksi (tidak terdapat bilangan sama muncul secara periodik) sepanjang keseluruhan siklus, dan kemunculan siklus pendek jarang. IBAA dibuat dengan mengkombinasikannya dengan IA tanpa mengurangi keamanan IA atau menghasilkan *bias*.

ISAAC merupakan pengembangan dari IBAA, dengan tambahan agar mudah disimpan di memori. Selain itu kode ISAAC dioptimisasikan untuk kecepatan, state-state yang berurutan akan diubah menjadi tidak berurutan dengan cepat, dan tidak ada siklus pendek sama sekali.

ISAAC lebih cepat dari IBAA, dijamin tidak menghasilkan nilai seed yang buruk, mengubah state berurutan menjadi state tidak urut lebih cepat, dan tidak ada siklus pendek[]. ISAAC membutuhkan 18,75 instruksi mesin untuk menghasilkan nilai sebesar 32-bit, dan 19 instruksi mesin untuk menghasilkan nilai sebesar 64-bit. Jumlah operasi tersebut akan berjalan sangat cepat pada komputer 32-bit.

Algoritma ISAAC memiliki kemiripan dengan algoritma PRNG RC4. ISAAC menggunakan *array* 256 integer 32-bit sebagai state internal, menulis hasilnya ke array 256 integer lain, dimana mereka dibaca satu per satu hingga *array*-nya kosong, sampai suatu saat mereka di rekomputasi. Kom-putasinya terdiri dari mengubah mm[i] dengan mm[i^128], 2 elemen dari mm ditemukan dengan indireksi, akumulator, dan *counter*, untuk semua nilai i dari 0 hingga 255. Diagram di bawah ini menunjukan bagaimana IBAA atau ISAAC memproduksi sebuah hasil dalam r[] dan menggantikan satu nilai dalam m[].



Gambar 2 Proses Komputasi Algoritma ISAAC

Banyak implementasi dari ISAAC sangat cepat, hingga mereka dapat bersaing dengan algoritma PRNG lain, bahkan dengan PRNG yang didesain untuk kecepatan bukan keamanan[3].

VI. RANCANGAN OPTIMASI ALGORITMA RSA

Kekuatan dari algoritma RSA terletak dari bilangan prima dan panjang bilangan prima yang digunakan sebagai kunci. Semakin besar bilangan prima yang digunakan, maka kunci tersebut akan semakin aman, karena semakin sulit diprediksi oleh kriptanalis. Sampai saat ini, untuk nilai *n* sepanjang 512-bit sudah bisa dipecahkan dengan beberapa ratus komputer. Oleh karena itu, bilangan *n* yang digunakan harus di atas 512-bit.

Disinilah letak *pseudorandom number generator* berperan. Karena untuk bilangan prima sebesar itu tidak bisa dilakukan secara manual. Oleh karena itu dibutuhkan algoritma pembangkit kunci untuk RSA menggunakan PRNG.

Karena kekuatan kemanan RSA terletak pada kunci, maka pembangkitan kunci pun menjadi penting. Karena pembangkitan kunci dilakukan dengan pembangkitan bilangan random, oleh karena itu algoritma PRNG yang digunakan harus aman secara kriptografi.

Dari syarat-syarat tersebut, dapat dilihat bahwa ISAAC adalah PRNG yang potensial untuk digunakan sebagai pembangkit kunci RSA. Karena selain memang dirancang sebagai *cryptographically secure* PRNG, algoritma ISAAC berjalan sangat cepat. Hal ini tentunya akan mengurangi beban komputasi komputer dibanding penggunaan PRNG biasa.

Oleh karena itu dalam paper ini, untuk optimasi algoritma RSA, diusulkan untuk menggunakan PRNG ISAAC sebagai alternatif algoritma PRNG untuk pembangkitan bilangan-bilangan kunci.

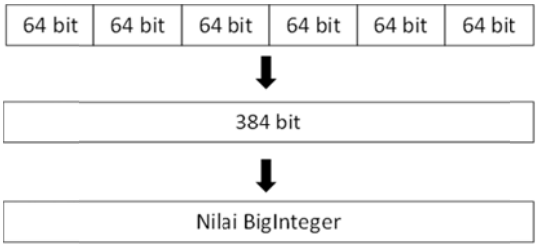
Untuk algoritma RSA, besar bilangan yang digunakan sebagai nilai *p* dan *q* untuk pembangkitan kunci adalah 48 byte, yang berarti 384 bit, untuk keamanan. Karena semakin besar bilangan kunci, semakin sulit untuk diserang. Karena bilangan *p* dan *q* sebesar 48 byte, maka besar nilai *n* dan *d* akan mencapai 96 byte (768 bit). Nilai ini tentu melebihi nilai 512-bit. Algoritma RSA diimplementasikan dalam sebuah program C#.

Dalam proses enkripsinya, plainteks dibagi menjadi blok-blok sepanjang *p* dan *q* yaitu 384 bit. Karena ukuran file umumnya jarang yang habis dibagi 384 bit, digunakan padding untuk memenuhi blok yang jumlah bitnya kurang. Namun hasil enkripsi atau ciperteks ditampung dalam blok-blok sepanjang 768 bit, karena hasil dari perhitungan $c_i = m_i^e \bmod n$ besarnya dapat mencapai lebih dari 384 bit, namun tidak melebihi 768 bit. Oleh karena itu panjang chiperteks menjadi jauh lebih besar dari plainteksnya. Pada proses dekripsi hal yang sebaliknya terjadi. Karena ciperteks ditampung dalam blok-blok 768 bit, oleh karena itu ciperteks dipecah menjadi blok-blok 768 bit. Hasil dekripsi ditampung dalam blok-blok plainteks 384 bit agar kembali menjadi seperti semula. Dan padding pun dihilangkan, dengan cara memberi header panjang file pada file ciperteks, sehinga jumlah file bit yang diambil hanya sebesar panjang file yang disimpan pada header.

Dalam pembuatan algoritma RSA ini, digunakan kelas tambahan yang didapat dari sumber internet yang merupakan kelas BigInteger yang khusus dibuat untuk pengolahan bilangan-bilangan dalam Algoritma RSA. Kelas BigInteger ini digunakan untuk menampung bilangan-bilangan besar seperti bilangan sebesar 48 byte dan 96 byte, beserta pengolahannya.

Sedang proses pembangkitan kuncinya sendiri tidak semerta-merta langsung menggunakan PRNG ISAAC untuk membangkitkan nilai *p* dan *q*. Dalam algoritma RSA ini, digunakan kelas ISAAC yang diimplementasikan oleh Jenkins, dan kelas ISAAC yang diimplementasikan maksimal hanya dapat menghasilkan bilangan sebesar 64 bit, sementara bilangan kunci yang dibutuhkan sebesar 384 bit.

Oleh karena itu dilakukan penggabungan bit-bit 6 buah bilangan random yang dihasilkan oleh ISAAC, sehingga 6 buah 64-bit integer tersebut menjadi sebuah bitarray 384 bit. Kemudian BitArray 384 bit tersebut dikonversikan menjadi BigInteger sebesar 384 bit.



Gambar 3 Diagram penggabungan bit-bit bilangan hasil PRNG ISAAC

Setelah nilai BigInteger tersebut didapatkan, nilai BigInteger tersebut diperiksa apakah bilangan tersebut bilangan prima. Jika bilangan tersebut bilangan prima, maka nilai BigInteger itulah yang akan menjadi nilai *p* dan *q*, yang kemudian akan diolah menjadi kunci.

Berikut ini adalah kode pembangkitan kunci menggunakan PRNG ISAAC dalam kode C#:

```
public void GenerateKeyISAAC()
{

	p = gen48BytePrimeISAAC();
	q = gen48BytePrimeISAAC();
	n = p * q;
	BigInteger T = (p - 1) * (q - 1);
	e = gen48BytePrimeISAAC();
	while (e.gcd(T) != 1)
	{
		e = gen48BytePrimeISAAC();
	}
	d = e.modInverse(T);


}

public BigInteger gen48BytePrimeISAAC()
{


	BigInteger b = new BigInteger();
	while (!b.isProbablePrime())
	{
		b.genRandom48ByteISAAC();
	}
	return b;
}

public BigInteger genRandom48ByteISAAC()
{
	int i=0;

	//variabel untuk menampung hasil
bilangan random ISAAC
	Int64() aInt = new Int64();
	BitArray[] IntBits = BitArray[6];
	BitArray IntBit = new BitArray(384,
false);
	BigInteger b = new BigInteger();;

	//Kelas RSA dipanggil untuk
memanggil fungsi manipulasi bit
copyBitArray dan getBigIntFromBitArray
	RSA R = new RSA();

	//Kelas PRNG ISAAC 64-bit
	Rand r = new Rand();

	for (i = 0; i < 6 ; i++)
	{
		r.Isaac();
		//mendapatkan nilai random
dari kelas r
		aInt = r.val();
		Byte[] intByte =
BitConverter.GetBytes(aInt);
		intBits[i] = new
BitArray(intByte);
	}

	for (i = 0; i < 6 ; i++)
	{

	R.copyBitArray(IntBit,intBits[i],0,(
i * 64), 64);
	}

	b = getBigIntFromBitArray(IntBit);

	return b;

}
```

Fungsi copyBitArray dan getBigIntFromBitArray terletak dalam kelas RSA.cs. Isi dari fungsi tersebut adalah :

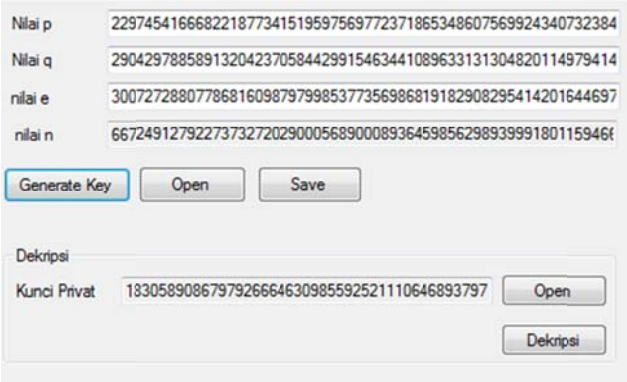
```
public void copyBitAray(BitArray Target,
BitArray Source, int indexAwal, int
indexTarget, int Length)
{
	for (int i = indexAwal, j =
indexTarget; j < (indexTarget + Length);
i++, j++)
	{
		if (i >= Source.Length)
break;
		Target.Set(j, Source.Get(i));
	}
}

public BigInteger
getBigIntFromBitArray(BitArray bitArray)
{
	Byte[] Data = new
Byte[bitArray.Length/8];
	Console.WriteLine(Data.Length);
	bitArray.CopyTo(Data,0);
	BigInteger b = new BigInteger(Data);
	return b;
}
```

Dari algoritma tersebut, kunci publik (*e* dan *n*) dan kunci privat (*d* dan *n*) didapatkan.

VII. PENGUJIAN

Pengujian dilakukan dengan mengenkripsi dan

mendekripsi suatu pesan pendek dalam string. Proses pengujian dilakukan dengan terlebih dahulu membangkitkan kunci menggunakan algoritma pembangkit kunci memakai algoritma PRNG ISAAC yang sudah dibuat



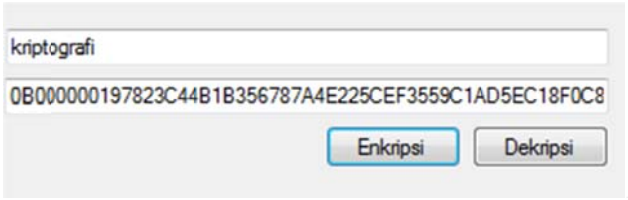
Gambar 4 Pembangkitan kunci menggunakan ISAAC

Nilai kunci publik dan kunci privat berhasil didapatkan menggunakan algoritma yang sudah dibuat. Nilai-nilai tersebut adalah :

```
p =
22974541666822187734151959756977237186534860756992434073238 4
07569924340732384985294018753813081849008321952032793065841372965517309 31

q =
29042978858913204237058442991546344108963313130482011497941 4
14078933872668922698288438910683645377536785771168019813

e =
30072728807786816098797998537735698681918290829541420164469 7
97225272536354104249012842676082838957733445799612649243

n =
66724912792273732720290005689000893645985625629893999180115 9
46612934297699881646416618026129166772884212326806619953968815670928622 6
16812312256117493622325348281607151342970371564554199311459453302396734010350446893351433017685293590 3

d =
18305890867979266646309855925211106468937971055921955857687693387488340292332646945274454846868755379724467354452282585258258089988134873345456613861145512247350347138417662991553077482740291215272452869197421465941209510299121426 7
```

Walaupun secara kasatmata tidak terlihat dan harus 6 kali menghasilkan bilangan random untuk menda-patkan bilangan random 384 bit, hal tersebut dapat diimbangi dengan proses komputasi ISAAC yang cepat. Selain itu hasil bilangan acak yang didapatkan pun lebih aman secara kriptografi.

Kemudian kunci yang dihasilkan digunakan untuk mengenkripsi suatu pesan pendek.

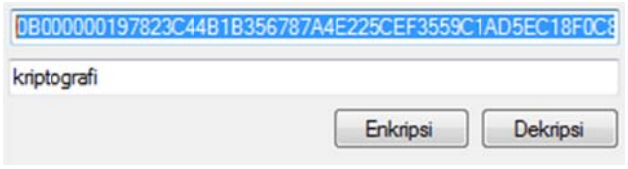


Gambar 5 Contoh enkripsi

Plainteks yang akan dienkripsi adalah kriptografi, sementara hasil enkripsinya merupakan nilai heksadesimal dari chiperteks yang dihasilkan.

```
Hasil chiperteks =
0B000000197823C44B1B356787A4E225CEF3559C1AD
5EC18F0C84A452626FC5148685D9C07F62F8634603C
F65FEC9019EADAB2D78DB74FEA9B30395305624FF83
343A5EA248EF28F4888ECA077C0C35B66DC0F7E4517
A8189FA43531EFAAF16DC3BBB3AF
```

Kemudian chiperteks didekripsi kembali menggunakan kunci privat



.

Hasil dekripsi berhasil mengembalikan chiperteks menjadi plainteks semula.

V. KESIMPULAN

Dari paper ini, kesimpulan yang didapat adalah, proses pembangkitan kunci sangat penting dalam menentukan keamanan algoritma RSA. Sementara proses pembangkitan bilangan kunci yang sangat besar membutuhkan pembangkit bilangan acak, oleh karena itu pembangkit bilangan acak yang cepat, mangkus, dan aman secara kriptografi menjadi dibutuhkan.

Algoritma ISAAC adalah algoritma PNRG yang dapat digunakan untuk mengoptimasikan algoritma RSA, karena kecepatannya dan keamanannya secara kriptografi. Namun, hal ini tidak menutup kemungkinan algoritma PNRG lain yang lebih mangkus, cepat, dan aman, untuk dapat lebih mengoptimasikan algoritma RSA. Diharapkan kedepannya muncul algoritma PNRG lain yang lebih canggih untuk lebih mengoptimasikan algoritma RSA. Untuk saat ini ISAAC termasuk beberapa PNRG yang memiliki kualitas dan kecepatan tinggi dalam penggunaannya[3].

VIII. ACKNOWLEDGMENT

Ucapan terima kasih pertama-tama saya ucapkan sebesar-besarnya kepada Tuhan Yang Maha Kuasa. Kemudian terima kasih saya ucapkan kepada Bapak Rinaldi Munir yang telah membimbing kami selama mempelajari mata kuliah kriptografi. Kemudian kepada teman-teman saya yang telah mengajari saya bagaimana memanipulasi bit dan memberi tahu saya mengenai kelas BigInteger. Dan kepada penulis-penulis sumber referensi yang telah memberikan saya pengetahuan yang saya butuhkan.

REFERENCES

[1]Robert J. Jenkins Jr., *ISAAC. Fast Software Encryption* 1996, pp41–49.
[2]http://burtleburtle.net/bob/rand/isaacafa.html
[3]http://en.wikipedia.org/wiki/ISAAC_(cipher)
[4]http://burtleburtle.net/bob/rand/isaac.html
[5]http://en.wikipedia.org/wiki/RSA

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Mei 2011

Shauma Hayyu Syakura (13507025)