Implementasi Kriptosistem RSA pada Manajemen *Dump* Memori Sistem di Skenario Kegagalan

Muhammad Fauzan Rafi Sidiq Widjonarto - 13518147[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13518147@std.stei.itb.ac.id

***Abstract*—Sistem komputer sangat integral di kehidupan masyarakat. Setiap data, proses, dan kegiatan dilakukan melalui sebuah sistem komputer, baik langsung maupun tidak langsung. Karena interaksi data dan informasi yang besar, sistem tidak boleh mengalami kegagalan agar terjaga integritas datanya. Namun, hal ini tidak mungkin di kehidupan nyata, karena kegagalan pasti terjadi. Apabila terjadi kegagalan, sistem akan membuat *dump* memori yang berisi semua data dan informasi sensitif sebelum kegagalan agar bisa direstorasi nantinya. Karena mengandung informasi rahasia, maka *dump* ini harus diamankan dari akses orang yang tidak berhak. Pada makalah ini, akan dibahas implementasi kriptosistem RSA dalam manajemen *dump* memori.**

***Kata kunci*—Asimetrik, *Dump*, Kriptosistem, Memori, RSA**

I. Pendahuluan

Pada era modern, revolusi industri 4.0 yang sedang berlangsung membuat sebuah gelombang digitalisasi pada setiap industri yang ada. Digitalisasi ini meliputi berbagai bidang di industri, termasuk automasi berbagai pekerjaan, pengolahan data industri, serta lapangan produk dan jasa baru yang dihasilkan di relevansi yang muncul akibat revolusi tersebut.

Menurut data yang dikumpulkan dari tahun 2009 hingga 2015, kenaikan penggunaan sistem komputer di penggunaan domestik naik lebih dari dua kali lipatnya [1]. Hal ini menunjukkan bahwa semakin banyak masyarakat yang menggunakan sistem komputer sebagai suatu hal yang integral di kehidupan sehari-hari. Tentunya, dalam melakukan hal-hal yang menjadi kebutuhan masyarakat, sistem komputer menggunakan berbagai proses dan program yang membantu mengolah data pengguna. Salah satu contohnya adalah penggunaan sistem operasi, editor teks, sebuah klien surel, serta program *streaming* video sebagai bentuk hiburan. Dalam pengaksesannya, pengguna akan selalu memasukkan data-data pribadinya ke sistem tersebut, lalu akan diolah untuk memberikan hal yang diinginkan oleh pengguna.

Bila ada suatu kegagalan yang ada di sistem komputer tersebut, sistem operasi yang digunakan memiliki mekanisme untuk membuat suatu berkas *dump* memori dari sistem sebelum terjadi kegagalan. Hal ini bertujuan untuk mempermudah *recovery* dari kegagalan agar pengguna tidak rugi di sisi sumberdaya dan informasi yang hilang. Tentunya berkas *dump* ini disimpan di suatu tempat yang aman di dalam perangkat sistem komputer. Namun, masalah yang muncul adalah untuk sebuah sistem terdistribusi yang perangkat kerasnya berbeda tempat dengan perangkat klien yang mengaksesnya. Bisa saja suatu pihak lain masuk ke perangkat tersebut dan mengakses *dump* memori dari sistem untuk mengubah dan mengambil data sensitif pengguna. Oleh karena itu dibutuhkan suatu mekanisme untuk mengamankan berkas *dump* memori ini agar hanya dapat diakses oleh pengguna yang bersangkutan saja untuk menjaga kerahasiaan dan integritas dari setiap data yang ada.

II. Dasar Teori

A. Kriptosistem RSA

RSA (Rivest – Shamir – Adleman) adalah sistem kriptografi kunci publik yang banyak digunakan untuk transmisi data yang aman. Itu juga salah satu yang tertua. Akronim RSA berasal dari nama belakang Ron Rivest, Adi Shamir, dan Leonard Adleman, yang secara terbuka menjelaskan algoritma tersebut pada tahun 1977. Sistem yang setara dikembangkan secara diam-diam, pada tahun 1973 di GCHQ (badan intelijen sinyal Inggris), oleh ahli matematika Inggris Clifford Cocks. Sistem itu dideklasifikasi pada tahun 1997 [2].

Dalam sistem kriptografi kunci publik, kunci enkripsi bersifat publik dan berbeda dari kunci dekripsi, yang dirahasiakan (privat). Pengguna RSA membuat dan menerbitkan kunci publik berdasarkan dua bilangan prima besar, bersama dengan nilai tambahan. Bilangan prima dirahasiakan. Pesan dapat dienkripsi oleh siapa saja, melalui kunci publik, tetapi hanya dapat diterjemahkan oleh seseorang yang mengetahui bilangan prima [3].

Keamanan RSA bergantung pada kesulitan praktis dalam memfaktorkan hasil kali dua bilangan prima besar.. Memutus enkripsi RSA dikenal sebagai masalah RSA. Apakah sesulit masalah pemfaktoran adalah pertanyaan terbuka. Tidak ada metode yang diterbitkan untuk mengalahkan sistem jika kunci yang cukup besar digunakan.

RSA adalah algoritma yang relatif lambat. Karena itu, ini tidak umum digunakan untuk mengenkripsi data pengguna secara langsung. Lebih sering, RSA digunakan untuk mengirimkan kunci bersama untuk kriptografi kunci simetris, yang kemudian digunakan untuk dekripsi-enkripsi massal.

Berikut merupakan properti dari kriptosistem RSA [4]:

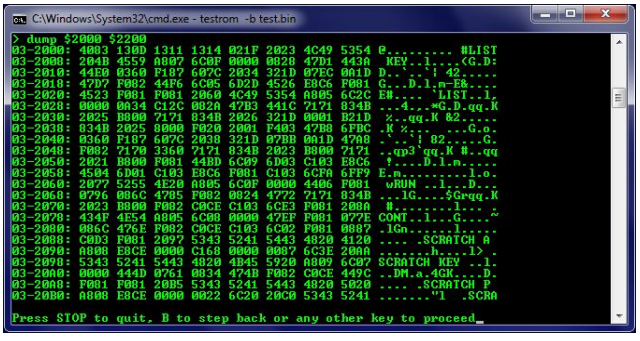
1. Dua bilangan prima p dan q (rahasia)
2. Modulus n, hasil perkalian dari p dan q (tidak rahasia)
3. Fungsi totient euler (p–1) * (q–1) (rahasia)
4. Kunci enkripsi e (tidak rahasia), dengan syarat e dan fungsi totient euler saling prima
5. Kunci dekripsi d (rahasia), dihitung dari invers modulo e dan fungsi totient euler
6. Plainteks m
7. Ciperteks c

Untuk mengenkripsi plainteks m, hasilnya merupakan perpangkatan m dengan e modulo n. Lalu untuk melakukan dekripsi ciperteks c, hasilnya merupakan perpangkatan c dengan d modulo n. Karena kedua metode untuk enkripsi dan dekripsi berbeda, maka RSA termasuk ke kriptosistem asimetrik sistem kunci publik.

RSA melibatkan kunci publik dan kunci privat. Kunci publik dapat diketahui oleh semua orang, dan digunakan untuk mengenkripsi pesan. Tujuannya adalah agar pesan yang dienkripsi dengan kunci publik hanya dapat didekripsi dalam waktu yang wajar dengan menggunakan kunci privat. Kunci publik diwakili oleh bilangan bulat n dan e. Kunci privat, dengan bilangan bulat d (meskipun n juga digunakan selama proses dekripsi, jadi mungkin dianggap sebagai bagian dari kunci privat juga).

B. Dump Memori

Dalam sistem komputer, *dump* memori adalah suatu *record* yang terdiri dari status rekaman dari memori kerja program komputer pada waktu tertentu, umumnya ketika program mengalami kegagalan atau jika tidak dihentikan secara tidak normal. Dalam praktiknya, bagian kunci lain dari status program biasanya dibuang pada saat yang sama, termasuk register prosesor, yang mungkin mencakup pencacah program dan penunjuk tumpukan, informasi manajemen memori, dan penanda serta informasi prosesor dan sistem operasi lainnya. Snapshot *dump* (atau snap *dump*) adalah *dump* memori yang diminta oleh operator komputer atau oleh program yang sedang berjalan, setelah itu program dapat melanjutkan. Dump memori sering digunakan untuk membantu mendiagnosis dan mendebug kesalahan dalam program komputer.



Gambar 1. Contoh bentuk *dump* memori dalam berkas heksadesimal. Sumber: https://sites.google.com/site/yr9icthexadecimal/when-is-hexadecimal-use-for/memory-dumps

Pada banyak sistem operasi, pengecualian fatal dalam sebuah program secara otomatis memicu pembuangan inti. Dengan ekstensi, frase "to dump core" dalam banyak kasus berarti, setiap kesalahan fatal, terlepas dari apakah ada catatan dari memori program. Istilah-istilah seperti "*Dump* memori" atau hanya "*dump*" juga telah menjadi jargon untuk menunjukkan keluaran dari sejumlah besar data mentah untuk pemeriksaan lebih lanjut atau tujuan lain.

Dump memori dapat berfungsi sebagai bantuan debugging yang berguna dalam beberapa situasi. Pada sistem mandiri atau pemrosesan batch awal, *Dump* memori memungkinkan pengguna untuk men-debug program tanpa memonopoli fasilitas komputasi (yang sangat mahal) untuk debugging. Hasil cetakan juga bisa lebih nyaman. Pada komputer bersama, baik sistem berbagi waktu, pemrosesan batch, atau server, *Dump* memori memungkinkan debugging off-line sistem operasi, sehingga sistem dapat segera kembali beroperasi.

Dump memori memungkinkan pengguna menyimpan error untuk analisis nanti atau di luar situs, atau perbandingan dengan error lainnya. Untuk komputer yang disematkan, mungkin tidak praktis untuk mendukung debugging pada komputer itu sendiri, sehingga analisis *dump* dapat dilakukan di komputer yang berbeda. Beberapa sistem operasi seperti versi awal Unix tidak mendukung pemasangan debugger ke proses yang sedang berjalan, jadi *dump* memori diperlukan untuk menjalankan debugger pada konten memori proses.

Dump memori dapat digunakan untuk menangkap data yang dibebaskan selama alokasi memori dinamis dan dengan demikian dapat digunakan untuk mengambil informasi dari program yang tidak lagi berjalan. Dengan tidak adanya debugger interaktif, *Dump* memori dapat digunakan oleh programmer yang tekun untuk menentukan kesalahan dari pemeriksaan langsung.

Core dump merepresentasikan konten lengkap dari wilayah yang dibuang dari ruang alamat dari proses yang dibuang. Tergantung pada sistem operasi, dump mungkin berisi sedikit atau tidak ada struktur data untuk membantu interpretasi wilayah memori. Dalam sistem ini, interpretasi yang berhasil mengharuskan program atau pengguna yang mencoba menafsirkan dump memahami struktur penggunaan memori program.

Debugger dapat menggunakan tabel simbol, jika ada, untuk membantu pemrogram menafsirkan dump, mengidentifikasi variabel secara simbolis dan menampilkan kode sumber; jika tabel simbol tidak tersedia, interpretasi dump lebih sedikit dimungkinkan, tetapi mungkin masih cukup mungkin untuk menentukan penyebab masalah. Ada juga alat dengan tujuan khusus yang disebut penganalisis dump untuk menganalisis dump. Salah satu alat yang populer, tersedia di banyak sistem operasi, adalah objdump GNU binutils.

Pada sistem operasi mirip Unix modern, administrator dan pemrogram dapat membaca file core dump menggunakan pustaka GNU Binutils Binary File Descriptor (BFD), dan GNU Debugger (gdb) dan objdump yang menggunakan pustaka ini. Library ini akan menyediakan data mentah untuk alamat tertentu di wilayah memori dari core dump; itu tidak tahu apa-apa tentang variabel atau struktur data di wilayah memori itu, jadi aplikasi yang menggunakan perpustakaan untuk membaca dump inti harus menentukan alamat variabel dan menentukan tata letak struktur data itu sendiri, misalnya

dengan menggunakan tabel simbol untuk program yang menjalani proses debug.

Dalam sistem operasi yang lebih lama dan lebih sederhana, setiap proses memiliki ruang alamat yang berdekatan, sehingga file dump terkadang hanya berupa file dengan urutan byte, digit, karakter, atau kata. Pada mesin awal lainnya, file dump berisi catatan diskrit yang masing-masing berisi alamat penyimpanan dan konten terkait. Pada mesin awal, dump sering kali ditulis oleh program dump yang berdiri sendiri dan bukan oleh aplikasi atau sistem operasi.

Di sebuah sistem operasi contoh seperti IBM System / 360, sistem operasi standar menulis ABEND dan SNAP dump yang diformat, dengan alamat, register, dan konten penyimpanan. Semuanya dikonversi ke bentuk yang dapat dicetak. Rilis selanjutnya menambahkan kemampuan untuk menulis dump yang tidak diformat, yang disebut pada saat itu sebagai *core image dumps*.

Dalam sistem operasi modern, ruang alamat proses mungkin memiliki celah, dan berbagi halaman dengan proses atau file lain, sehingga representasi yang lebih rumit digunakan; mereka juga dapat menyertakan informasi lain tentang status program pada saat pembuangan.

C. Sistem Operasi dan Manajemen Memori

Sistem operasi (OS) adalah perangkat lunak sistem yang mengelola perangkat keras komputer, sumber daya perangkat lunak, dan menyediakan layanan umum untuk program komputer.

Sistem operasi pembagian waktu menjadwalkan tugas untuk penggunaan sistem yang efisien dan mungkin juga termasuk perangkat lunak akuntansi untuk alokasi biaya waktu prosesor, penyimpanan massal, pencetakan, dan sumber daya lainnya.

Untuk fungsi perangkat keras seperti input dan output dan alokasi memori, sistem operasi bertindak sebagai perantara antara program dan perangkat keras komputer, meskipun kode aplikasi biasanya dieksekusi langsung oleh perangkat keras dan sering membuat panggilan sistem ke Fungsi OS atau terganggu olehnya. Sistem operasi ditemukan di banyak perangkat yang berisi komputer - dari telepon seluler dan konsol video game hingga server web dan superkomputer.

Antara lain, kernel sistem operasi multiprogramming harus bertanggung jawab untuk mengelola semua memori sistem yang saat ini digunakan oleh program. Ini memastikan bahwa program tidak mengganggu memori yang sudah digunakan oleh program lain. Sejak program berbagi waktu, setiap program harus memiliki akses independen ke memori.

Manajemen memori kooperatif, yang digunakan oleh banyak sistem operasi awal, mengasumsikan bahwa semua program menggunakan manajer memori kernel secara sukarela, dan tidak melebihi memori yang dialokasikan. Sistem manajemen memori ini hampir tidak pernah terlihat lagi, karena program seringkali mengandung bug yang dapat menyebabkannya melebihi memori yang dialokasikan. Jika suatu program gagal, itu dapat menyebabkan memori yang digunakan oleh satu atau lebih program lain terpengaruh atau ditimpa. Program berbahaya atau virus dapat dengan sengaja mengubah memori program lain, atau dapat mempengaruhi pengoperasian sistem operasi itu sendiri. Dengan manajemen memori kooperatif, hanya dibutuhkan satu program yang tidak berfungsi dengan baik untuk merusak sistem.

Perlindungan memori memungkinkan kernel untuk membatasi akses proses ke memori komputer. Berbagai metode perlindungan memori tersedia, termasuk segmentasi memori dan paging. Semua metode memerlukan beberapa tingkat dukungan perangkat keras (seperti 80286 MMU), yang tidak ada di semua komputer.

Baik dalam segmentasi dan paging, register mode terlindung tertentu menentukan ke CPU alamat memori apa yang harus diizinkan untuk diakses oleh program yang sedang berjalan. Upaya untuk mengakses alamat lain memicu interupsi yang menyebabkan CPU masuk kembali ke mode supervisor, menempatkan kernel yang bertanggung jawab. Ini disebut pelanggaran segmentasi atau singkatnya Seg-V, dan karena keduanya sulit untuk menetapkan hasil yang berarti untuk operasi semacam itu, dan karena biasanya merupakan tanda program yang tidak berfungsi, kernel umumnya memilih untuk menghentikan program yang melanggar , dan melaporkan kesalahan tersebut.

Windows versi 3.1 hingga ME memiliki beberapa tingkat perlindungan memori, tetapi program dapat dengan mudah menghindari kebutuhan untuk menggunakannya. Kesalahan proteksi umum akan dihasilkan, menunjukkan pelanggaran segmentasi telah terjadi; namun, sistem akan sering macet.

III. Implementasi

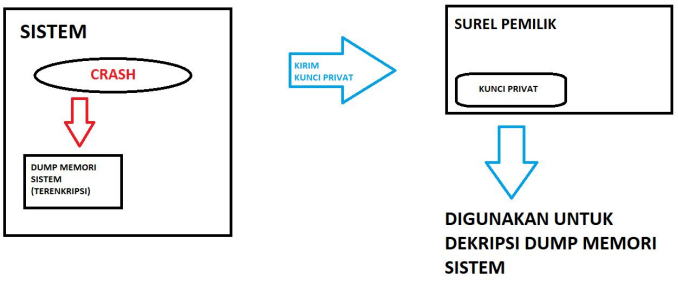
Sistem implementasi yang diajukan adalah penggunaan kriptosistem RSA pada manajemen penyimpanan *dump* memori yang sudah dibuat apabila terjadi kegagalan. Pertama-tama sesaat setelah terjadi kegagalan, sistem operasi membangkitkan sebuah kunci publik di sistem internalnya dan melakukan enkripsi berkas *dump* memori yang telah dihasilkan oleh sistem. Lalu, sistem operasi akan membangkitkan kunci privat yang dikirimkan ke suatu *channel* khusus dari pemilik sistem, misalkan sebuah alamat surel, dengan pengiriman protokol yang aman seperti SSL atau HTTPS. Saat sistem sudah kembali pulih, pemulihan data yang telah dibuat *dump* harus dilakukan menggunakan private key yang dikirim oleh sistem ke pemilik sistem (dalam hal ini, asumsikan saja dikirim melalui surel di *channel* yang aman).

Dalam perlakuan enkripsi, setiap data di dalam *dump* akan dienkripsi dalam bentuk penggalan-penggalan byte yang satu per satu dienkripsi dengan kunci publik RSA. Hal ini dilakukan dengan konsiderasi bahwa algoritma RSA yang cenderung lambat dibandingkan dengan algoritma kriptografi simetrik. Penggalan-penggalan yang telah terenkripsi ini akan digabungkan kembali ke berkas *dump* yang ada, lalu akan diletakkan di folder *dump* yang sudah disediakan oleh sistem operasi.

Berikut merupakan *pseudocode* dari implementasi sistem pengiriman yang diajukan:

```
deteksi_kegagalan()
generate_key_pair()
kirim_private_key(email_pengguna)
encrypt_dump(public_key)
if mempunyai private key:
```

```
decrypt_dump(private_key)
```



Gambar 2. Skema manajemen *dump* memori. Dibuat dengan perangkat lunak Paint

Berikut merupakan kode bantuan yang dibuat untuk melakukan percobaan kriptosistem RSA pada suatu berkas *dump*.

```
def modular_expo(base, power, mod):
    else:    ret = 1
    while power > 0:
        if power % 2 == 1:
            ret = ret * base % mod
        base = base * base % mod
        power = int(power / 2)
    return ret

def euler_totient(n : int) -> int:
    if n == 1:
        return 1
    ret = n
    temp = 2
    while temp * temp <= n:
        if n % temp == 0:
        ret -= ret // temp
        while n % temp == 0:
            n = n // temp
        temp += 1
    if n > 1:
        ret -= ret // n
    return ret

def extended_gcd(a : int, b : int):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = extended_gcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modular_inverse(a : int, mod : int):
    g, x, y = extended_gcd(a, mod)
    if g != 1:
        print('modular inverse does not exist')
        return -1

    return x % mod
```

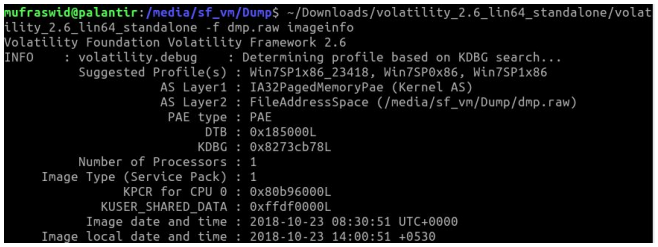
Fungsi-fungsi tersebut digunakan untuk membantu eksekusi dari kelas RSA yang akan dibuat untuk enkripsi dan dekripsi berkas. Berikut merupakan kelas RSA yang dibuat dengan bahasa Python versi 3.

```
def encrypt(self, path_in : str, path_out :
str, pub_key_path : str):
    dmp = open(path_in, 'rb')
    ilen = len(dmp.read())
    dmp.close()
    benc = b''
    e, n = read_public_key(pub_key_path)
    for i in range(0, ilen, 8):
        block = int.from_bytes(dmp[i:i+8],
byteorder='big')
        benc += pow(block, e, n).to_bytes(8,
byteorder='big')

    save = open(path_out, 'wb')
    save.write(benc)
    save.close()

def decrypt(self, path_in : str, path_out :
str, priv_key_path : str):
    dmp = open(path_in, 'rb')
    ilen = len(dmp.read())
    dmp.close()
    benc = b''
    d, n = read_private_key(priv_key_path)
    for i in range(0, ilen, 8):
        block = int.from_bytes(dmp[i:i+8],
byteorder='big')
        benc += pow(block, d, n).to_bytes(8,
byteorder='big')

    save = open(path_out, 'wb')
    save.write(benc)
    save.close()
```

Kelas diatas merupakan kelas yang menghandle enkripsi dan dekripsi dari file *dump* yang akan diinput oleh program utama. Pengguna tinggal memasukkan sebuah kunci publik atau kunci privat yang akan selanjutnya diolah oleh program untuk mengenkripsi dan mendekripsi berkas *dump* yang dimaksud.

IV. Studi Kasus

Berikut merupakan studi kasus yang akan dilakukan untuk implementasi RSA pada sebuah berkas dump. Untuk menguji bahwa *dump* memori dapat dibaca, akan digunakan sebuah alat *open source* Volatility [5] yang digunakan untuk membaca *dump* pada forensik memori. Sampel merupakan dump memori dari sistem Windows yang mengalami kegagalan di tengah beberapa proses yang sedang dijalankan. Menggunakan Volatility, didapatkan skema dari *dump*:



Gambar 3. Spesifikasi image dump menggunakan Volatility

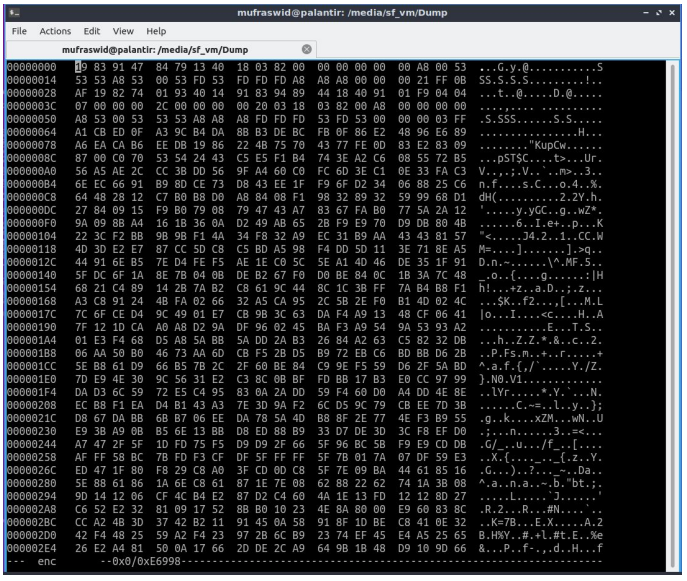
Asumsikan terjadi kegagalan yang menyebabkan sistem menghasilkan berkas *dump*, maka dilakukan enkripsi pada *dump* tersebut, sehingga tidak terbaca. Lalu dibangkitkan sebuah kunci publik dengan spesifikasi berikut menggunakan derivasi dari private key yang dibangkitkan dengan bantuan openssl:

```
-----BEGIN PUBLIC KEY-----
MIGeMA0GCSqGSIb3DQEBAQUAA4GMADCBiAKBgHWJGgG30
OvGHLZ1Sl/2tvubdn29
0kKgJUa10Akp04FSzWg+iBH9PzwPFuXwJq/qRupzER97R
Os6rv++PAgyO6fYA8Uu
E5PSzQVuPoYILv2lK1jw/hxbNHZ9TNiWtffWoBh3EYas4
2JBfu5xqSnes2PbmU0O
2t/to+lHqUI71vNDAgMBAAE=
-----END PUBLIC KEY-----
```

Serta private key yang dikirimkan ke pengguna (tidak disimulasikan pengirimannya) menggunakan openssl genrsa:

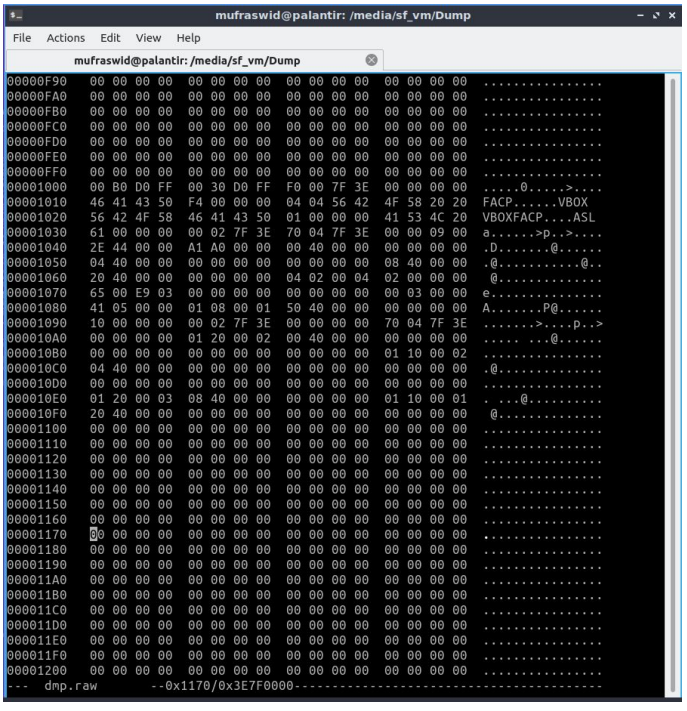
```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgHWJGgG30OvGHLZ1Sl/2tvubdn290kKgJ
Ua10Akp04FSzWg+iBH9
PzwPFuXwJq/qRupzER97ROs6rv++PAgyO6fYA8UuE5PSz
QVuPoYILv2lK1jw/hxb
NHZ9TNiWtffWoBh3EYas42JBfu5xqSnes2PbmU0O2t/to
+lHqUI71vNDAgMBAAEC
gYAy67AJuIJZ812zKsOLKc8C2MqQFXIHjWw440H7BlTsv
VviYtz+tjH0KjFocw+/
vQb2Qd3UPsqQAr+YaumFP4YaThv2RAh4Q0N0Wz2b6d6t6
s8QsOSIEOE+HKSly3AD
QghBYOQecqUAI3dPW9LafWGiwpfmwy4MsH33ksmDzsMu2
QJBALLHl9BbMTZ1s3Dp
lwb1SvnmlAIFUu4yvXHGY+dFJaZRxPLGrva7R0T0Qz7J+
2c5xLXNxjBkkD9SH0ly
t/9akEUCQQCoTX6A/pMRFjeJ2enqT0IBgDD1TuuDQ+TB0
a8VSmfCgn/5kB7xSBRG
Ls2c3ZeBKp6Yd6wGQwcV9L17CLRpfIHnAkEAmAWA4nuo7
VZIr8rx0cjjaHzZfp5o
VoRu80Bv00jJKxR9FLQ+MzAs+kV6qPm3/Z7x1maUELJyt
mVulVYnf8vSwQJAL2Fx
pbsFiB/rDsHsWmxptqZyIf1PzCqG8hGn6qSBxX6brJRoQ
aD7JC70755qkpB/CYlX
QoQIMKQv2709nRzSLQJBAKTlxzae4b36opk+l6C1m/lUC
pxZpo1/DfyKubUUL79Z
1CHnmyrr48MlGIENbqqybzH7QmwOqBTtpz823ZdkOSI=
-----END RSA PRIVATE KEY-----
```

Berikut merupakan hasil enkripsi dari *dump* tersebut yang diakses lewat hex editor (untuk melihat representasi biner):



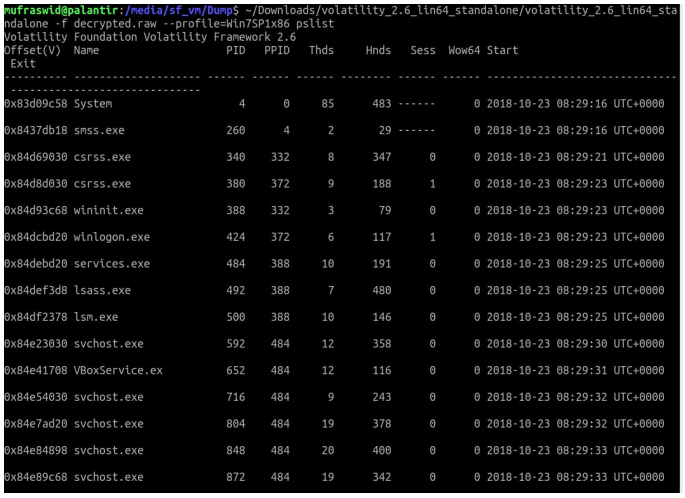
Gambar 4. Spesifikasi image dump pada satu sektor tertentu setelah dienkripsi

Semua isi berkas sudah rusak dan tidak mengikuti kaidah sebuah dump. Bandingkan hal ini dengan image *dump* sebelum dienkripsi. Karena setiap 8-byte dienkripsi dengan RSA, lalu di-append. Untuk setiap byte tersebut akan memiliki sebuah tingkat random yang khusus untuk setiap bytenya. Pada hex editor, terlihat spesifikasi header binary untuk image dump yang dibuka lewat Virtualbox:



Gambar 5. Spesifikasi *header* image *dump* sebelum di enkripsi

Hal ini menunjukkan bahwa *dump* yang awalnya bisa dibaca sudah tidak dapat dibaca lagi. Apabila dilakukan dekripsi menggunakan private key, maka dump dapat diakses kembali kontennya. Berikut merupakan pengaksesan konten *dump* yang berisi proses-proses yang terhenti saat kegagalan:



Gambar 6. Proses dari *dump* yangs duah didekripsi berhasil dibaca oleh pemilik. Terlihat banyak proses sensitif seperti winlogon yang memuat data login dari pengguna.

Lalu untuk lama enkripsi, cenderung lama untuk 1 GB *dump* memori dibutuhkan 20 menit untuk melakukan enkripsi dan dekripsi dari file *dump* tersebut. Hal ini merupakan konsekuensi dari algoritma RSA yang cenderung lambat dibandingkan dengan algoritma kriptosistem simetrik.

V. Kesimpulan

Skema dari manajemen *dump* memori sistem dengan kriptosistem RSA cukup aman dari serangan dalam membaca isi data di *dump* memori. Walaupun waktu performa yang cenderung lambat, hal ini bisa ditoleransi dengan keamanan yang diberikan oleh kriptosistem ini. Hal ini dapat diperbaiki dengan memadukan sistem dengan sebuah kriptosistem simetrik atau menggunakan ECC yang menempati ruang kunci yang lebih sedikit.

VI. Github

https://github.com/mufraswid/dump-enc

VII. Acknowledgment

Penulis ingin berterima kasih dan bersyukur kepada Allah SWT Tuhan Yang Maha Esa, karena-Nya penulis bisa berkuliah di ITB jurusan Teknik Informatika. Penulis berterima kasih kepada keluarga penulis yang mendukung keberjalanan kuliah. Penulis juga ingin berterima kasih ke Bapak Dr. Ir. Rinaldi Munir, M.T. atas bimbingan dan ilmu yang beliau berikan di mata kuliah IF4020 Kriptografi.

Referensi

[1] https://www.statista.com/statistics/748551/worldwide-households-with-computer/

[2] Smart, Nigel (February 19, 2008). "Dr Clifford Cocks CB". Bristol University.

[3] Rivest, R.; Shamir, A.; Adleman, L. (February 1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" (PDF). *Communications of the ACM*. **21** (2): 120–126. CiteSeerX 10.1.1.607.2677. doi:10.1145/359340.359342. S2CID 2873616.

[4] Munir, Rinaldi. Bahan Mata Kuliah IF4020 Kriptografi 2020/2021 Algoritma RSA. Program Studi Informatika Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[5] https://www.volatilityfoundation.org/

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, 21 Desember 2020

Muhammad Fauzan Rafi Sidiq Widjonarto 13518147