Studi Analisis Aplikasi Algoritma *One-Time Pad* dan *AES* pada Keamanan Sistem *Cloud Computing*

Haris Amrullah Lubis – 13507038

Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung

Jl. Ganesha No. 10, Bandung, Jawa Barat

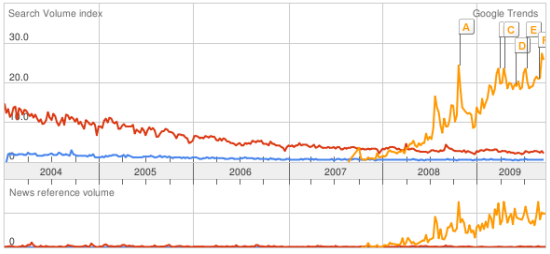
e-mail: haris.lubis@gmail.com

ABSTRAK

Cloud computing merupakan sebuah tren di masa kini. Hal ini disebabkan karena cloud computing sangat ekonomis dan efisien. Namun, di balik kelebihan tersebut, para pakar kriptografi di dunia sedang berlomba-lomba untuk memecahkan permasalahan utama dari sistem *cloud computing*, yaitu masalah keamanan. Masalah keamanan menjadi masalah utama terkait dengan sifat cloud computing yang serba berbagi. Komponen-komponen seperti CPU, media penyimpanan, dan memori perlu diatur sedemikian rupa sehingga walaupun berbagi, namun tetap menjaga privasi penggunanya. Melihat sifat berbaginya ini, penulis melihat sebuah kesempatan bagi *one-time pad* untuk berevolusi sehingga meminimisasi inefisien yang timbul dari pemakaian *one-time pad*. Kemudian, penulis juga menyoroti pemakaian algoritma AES, algoritma yang saat ini masih dipakai di sistem *cloud computing*. Apakah di dunia yang baru ini algoritma AES masih cukup aman digunakan. Jika tidak, apakah perlu dilakukan evolusi algoritma AES merespon dunia yang baru ini.

Kata kunci : cloud computing, AES, one-time pad

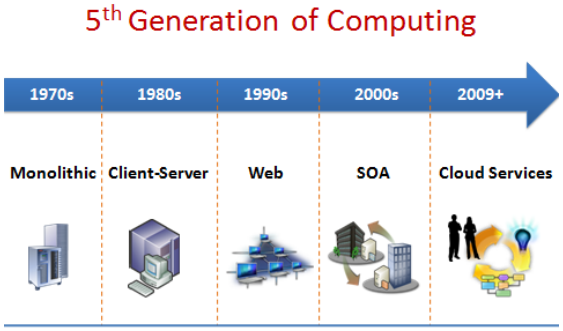
1. *CLOUD COMPUTING*



Gambar 1 : Tren dari *Cluster Computing*, *Cloud Computing*, dan *Grid Computing*

Dari gambar 1, terlihat bahwa orang-orang sekarang sedang memperhatikan *cloud computing*. Tampak bahwa sejak tahun 2007, orang-orang mulai mencari banyak hal tentang *cloud computing* melalui *Google*, khususnya sejak awal tahun 2009. Terlihat bahwa jenis-jenis komputasi lain, seperti *cluster computing* dan *grid computing* tetap sama setiap tahunnya.

Lalu, apakah itu *cloud computing*? Sebenarnya, *cloud computing* masih baru dan belum memiliki definisi standar. Oleh karena itu, penulis mengambil definisi *cloud computing* dari Wikipedia. *Cloud computing* adalah sebuah komputasi berbasis internet, sehingga sumber daya yang sifatnya berbagi, *software*, dan informasi disediakan kepada komputer dan kakas lain melalui permintaan, seperti fasilitas publik. *Cloud computing* sangat dinamis dan sumber daya sering divirtualisasi dan disediakan sebagai sebuah layanan di internet.



Gambar 2 : Sejarah Komputasi

Dari gambar 2, terlihat bahwa *cloud computing* adalah generasi kelima dari komputasi, setelah *monolithic*, *client-server*, *web*, *service-oriented architecture* dan sekarang *cloud service*.

Untuk lebih sederhana, *cloud computing* dapat dikarakteristikkan menjadi *CLOUD (Common, Location-independent, Online, Utility, Demand)*. Jadi, *cloud computing* merupakan model berbasiskan permintaan, pelayanan mandiri, dan pembayaran setiap penggunaan. Infrastrukturnya dapat diprogram secara fleksibel. Aplikasinya dibuat sehingga aplikasi tersebut dapat disusun ulang. Kemudian layanannya dikirim melalui jaringan atau internet.

Infrastruktur dari *cloud* dapat dikategorikan menjadi dua, yaitu publik dan privat. Pada *cloud* privat, infrastruktur dikelola dan dimiliki oleh *customer* dan diletakkan di suatu tempat yang berada pada wilayah kendali *customer*. Dengan kata lain, hak akses ke data *customer* hanya dimiliki oleh *customer* itu sendiri dan organisasi yang *customer* percaya. Sedangkan, pada *cloud* publik, infrastruktur dikelola dan dimiliki oleh penyedia layanan *cloud* dan diletakkan di suatu tempat yang berada pada wilayah kendali *provider*. Dengan kata lain, data *customer* berada di luar kendali *customer* dan berpotensi diakses oleh pihak-pihak yang tidak dipercaya.

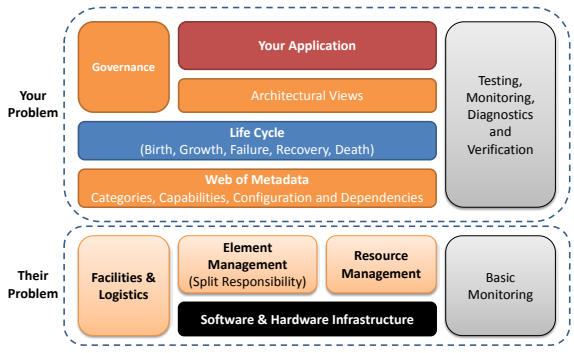
Arsitektur dari *cloud computing* dapat dibagi menjadi tiga. Pertama, *Software as a Service* (SaaS), yaitu

customer menggunakan perangkat lunak yang berjalan di dalam infrastruktur *provider*. Sebuah instansi dari perangkat lunak berjalan di *cloud* dan melayani berbagai pengguna atau organisasi. Kedua, *Platform as a Service* (PaaS) membungkus berbagai perangkat lunak dan menyediakannya sebagai sebuah layanan. Seseorang yang membuat PaaS akan menggabungkan sistem operasi, *middleware*, aplikasi, dan *development environment* menjadi sebuah layanan. Sedangkan, seseorang yang menggunakan PaaS akan melihat layanan sebagai sesuatu yang telah terbungkus dan dapat diakses melalui API. Ketiga, *Infrastructure as a Service* memberikan *storage* sederhana dan kemampuan komputasi sebagai sebuah layanan standar melalui jaringan. *Server*, sistem penyimpanan, *switch*, *router*, dan sistem lain dikumpulkan menjadi satu dan dibuat tersedia untuk menangani beban kerja mulai dari komponen aplikasi hingga aplikasi yang membutuhkan komputasi berperformansi tinggi.

Secara ekonomis, *cloud computing* jauh lebih hemat daripada teknologi informasi tradisional. Pengguna *cloud computing* dapat menghindari pengeluaran kapital pada *hardware*, *software*, dan layanan karena mereka cukup membayar penyedia layanan untuk setiap penggunaan sumber daya. Konsumsi perusahaan dapat dilihat sebagai sebuah tagihan pada penggunaan (setiap sumber daya yang digunakan, seperti listrik) atau pemesanan (berdasarkan waktu, seperti koran) tanpa pembayaran di muka.

Dari apa yang telah dipaparkan di atas, penulis mencoba merangkum kelebihan dan kekurangan dari *cloud computing*. Kelebihan dari *cloud computing* adalah, (1) biayanya yang murah, (2) arsitektur di masa yang akan datang, (3) kecepatannya yang tinggi, dan (4) adanya kejelasan tingkat manajemen. Kekurangan dari *cloud computing* adalah, (1) kehandalannya masih dipertanyakan, (2) kekurangan kendali, dan (3) masalah keamanan. Karena *cloud computing* masih lemah tingkat keamanannya, maka penulis menyoroti masalah keamanan pada tulisan ini.

2. KEAMANAN PADA *CLOUD COMPUTING*



Gambar 3 : Model Referensi Teknologi *Cloud*

Pada *cloud computing*, kita tidak akan pernah mampu mengendalikan semuanya. Selalu terdapat dua masalah, masalah *customer (your problem)* dan masalah *provider (their problem).* Pada arsitektur SaaS, masalah *customer* terdapat pada media penyimpanan atau datanya. Pada arsitektur PaaS, masalah *customer* terdapat pada aplikasi atau programnya. Pada arsitektur IaaS, masalah *customer* menjadi lebih luas, yaitu terletak pada lingkup aplikasi, aplikasi server, *middleware*, basis data, dan sistem operasi. Dari hal tersebut tampak bahwa semakin tinggi arsitektur yang ingin digunakan oleh pengguna, maka semakin tinggi pula tingkat masalah yang ada.

Penulis mencoba mengkategorikan masalah keamanan pada *cloud computing* menjadi dua bagian, masalah dalam mengatur *cloud* dan masalah dalam beroperasi di dalam *cloud*. Untuk masalah pertama, masalah keamanan dapat berupa manajemen resiko, legalitas, *electronic discovery*, audit, manajemen siklus hidup informasi, portabilitas, dan interoperabilitas. Untuk masalah kedua, masalah keamanan dapat berupa keamanan tradisional, operasi sentral data, respon terhadap insiden, virtualisasi, manajemen akses, manajemen identitas, penyimpanan, keamanan aplikasi, enkripsi, dan manajemen kunci. Dalam tulisan ini, penulis lebih memfokuskan pada empat masalah terakhir, yaitu penyimpanan, keamanan aplikasi, enkripsi, dan manajemen kunci, jika dihubungkan dengan disiplin ilmu kriptografi.

2.1. ENKRIPSI DAN MANAJEMEN KUNCI

Cloud computing mengubah cara kita berpikir tentang komputasi dengan menghilangkan atribut lokasi dari sumber daya. Maksudnya, dalam *cloud computing* lokasi sumber daya boleh di mana saja. Kita tidak perlu tahu di mana sumber daya tersebut disimpan atau diolah. Dengan kata lain, semua sumber daya komputasi dan jaringannya terabstraksi dengan sangat baik. Namun, pemisahan ini menimbulkan suatu permasalahan di bidang keamanan. Ketika kita tidak tahu di mana sumber daya tersebut diolah atau disimpan, kita tidak bisa seratus persen percaya apakah sumber daya tersebut aman atau benar diprosesnya.Di dalam dunia seperti itu, hanya terdapat satu cara untuk mengamankan sumber daya komputasi, yaitu dengan cara memperkuat enkripsinya dan manajemen kunci yang dapat dinilai kualitasnya.

Dari perspektif manajemen resiko, data yang tidak terenkripsi dapat dianggap telah hilang atau rusak oleh *customer*. Penyedia layanan aplikasi yang tidak mengendalikan sistem dasarnya seharusnya dapat menjamin bahwa data terenkripsi dengan baik ketika disimpan ke media penyimpanan. Jadi, enkripsi data berguna untuk memisahkan data yang disimpan dan data yang digunakan.

Namun, untuk masalah manajemen kunci, seharusnya pihak penyedia layanan *cloud* tidak turut campur dalam mengaturnya. Hal ini berguna untuk melindungi baik pihak pengguna maupun pihak penyedia agar tidak terjadi konflik saat dituduh menyediakan data yang tidak legal dan dapat berpotensi memecahkan banyak masalah.

Saat menentukan enkripsi apa yang digunakan saat perjanjian kontrak antara penyedia layanan dan pengguna, enkripsi harus mengikuti standar pemerintah atau industri dan dapat diaplikasikan dengan baik. Hal ini memicu pemakaian teknik enkripsi yang popular semacam RSA di algoritma kunci asimetrik atau AES di algoritma kunci simetrik.

Namun sub bab ini lebih berkaitan dengan masalah manajemen kunci dan aplikasi protokol keamanan dalam sistem *cloud computing*. Lagipula penggunaan algoritma enkripsi banyak berkaitan dengan fungsi *hash* untuk menjaga integritas data. Oleh karena permasalahan ini berada di luar lingkup bahasan tulisan ini, maka permasalahan tersebut hanya sebatas diperkenalkan dalam tulisan ini.

2.2. KEAMANAN APLIKASI

Berbeda dengan masalah enkripsi dan manajemen kunci, keamanan aplikasi lebih berkaitan dengan keamanan *cloud computing* di sisi aplikasinya. Dalam artian, fokus keamanannya berkaitan dengan proses komputasi (pemakaian sumber daya CPU), kode program, intrusi *software* atau *hardware*, dll. Misalkan data telah disimpan dengan sangat baik, telah terenkripsi, telah didistribusikan dengan cukup baik, maka kita bisa mengklaim bahwa sistem telah cukup aman. Namun, serangan dari pihak ketiga bisa melewati jalur aplikasi atau jaringan yang notabene belum terlindungi. Kita bisa melihat beberapa teknik-teknik penyerangan melalui modus aplikasi dan CPU dengan menggunakan studi kasus algoritma AES. Hal ini akan dipaparkan pada bagian 4 pada tulisan ini. Berbeda dengan penyerangan melalui aplikasi dan CPU pada sistem komputasi tradisional, sistem *cloud computing* dapat dianggap sebagai sistem komputasi yang *hyper threading*, sehingga solusi dalam sistem komputasi tradisional harus dimodifikasi. Pemaparan tentang hal tersebut akan dibahas juga dalam bab 4.

2.3. PENYIMPANAN

Seperti yang telah diungkapkan sebelumnya, bahwa penyimpanan data dalam sistem *cloud computing* sifatnya *location-independent*. Suatu data yang dimiliki oleh seseorang atau perusahaan, tidak jelas berada di mana. Untuk data yang satu dengan data yang lainnya yang dimiliki oleh satu organisasi bisa berada dalam dua media yang berbeda wilayah. Sebuah kenyataan ini menimbulkan keinginan bagi para *engineer* untuk menemukan suatu sistem penyimpanan yang mudah dicari namun aman disimpan. Metode penyimpanan yang aman dan mudah untuk dibaca maupun ditulis (*read*/*write*).

Jika permasalahan ini dikaitkan dengan kriptografi, maka kita akan menemukan benang merah pada proses mengamankan data-data yang disimpan. Penulis mengusulkan beberapa saran untuk mengamankan data di media penyimpanan ini menggunakan algoritma kunci simetrik yang akan dijelaskan pada bab 5. Selain itu, penulis juga mencoba menggunakan *one-time pad* untuk memecahkan masalah penyimpanan pada sistem *cloud computing*. Hal ini terangkum dalam bab 3.

3. ALGORITMA *ONE-TIME PAD* PADA SISTEM *CLOUD COMPUTING*

Algoritma *one-time pad* ditemukan oleh Mayor Joseph Mauborgne dan Gilbert Vernam pada tahun 1917. Skema enkripsi ini merupakan skema enkripsi yang sempurna. Gagasan dasar dari skema enkripsi ini adalah, kunci yang digunakan untuk mengenkripsi atau mendekripsi harus tidak boleh berulang, besar, dan satu karakter kunci mengenkripsi tepat satu karakter plainteks. Fungsi enkripsi adalah penjumlahan karakter plainteks dengan karakter kunci, kemudian dimodulo 26.

Setiap karakter kunci digunakan tepat satu kali. Pengirim mengirim pesan dan kemudian menghancurkan halaman kunci yang telah digunakan. Penerima menerima kunci yang identik dan menggunakan kunci tersebut untuk mendekripsi setiap karakter pada cipherteks. Kemudian penerima menghancurkan halaman kunci yang telah terpakai. Sebuah pesan baru sama dengan sebuah kunci baru.

One-time pad merupakan sebuah kriptosistem yang sangat sederhana, sangat rahasia, namun sangat tidak efisien. Pada kriptografi modern, representasi karakter diubah menjadi bit dan operasi penjumlahan diubah menjadi operasi XOR. Misalkan terdapat kunci k yang terdiri atas string bit $k_0k_1\ldots k_N$ dan digunakan untuk mengenkripsi plainteks $m = m_0m_1\ldots m_N$ dengan melakukan operasi XOR per bitnya. Jadi, cipherteks $c = c_0c_1..c_N$ didapat dari

$$c_i = k_i \oplus m_i, \text{ for } i = 0,1, \ldots, N$$

Setiap kunci hanya digunakan satu kali dan kemudian dibuang. Karena setiap kunci hanya digunakan satu kali dengan probabilitas yang sama dan hanya terdapat tepat satu kunci yang mengenkripsi *m* ke *c*, maka *one-time pad* memiliki kerahasiaan yang sempurna.

Namun, jika pengirim dan penerima ingin menggunakan *one-time pad* untuk bertukar pesan sebanyak *N* bits, maka mereka harus terlebih dahulu tahu informasi sebesar *N* bits yang menjadi kuncinya. Hal inilah yang menyebabkan *one-time pad* sangat tidak efisien untuk komunikasi jaringan berskala besar. Sehingga sampai saat ini, aplikasi *one-time*

pad hanya digunakan untuk komunikasi yang benar-benar harus rahasiadan pesan berukuran kecil.

Selain itu, *one-time pad* masih aman jika dan hanya jika kunci tidak pernah digunakan kembali. Ketika kunci digunakan lebih dari satu kali, maka kriptosistem menjadi mudah diserang oleh kriptanalis.

Masalah lain adalah, karena kunci harus benar-benar acak dan tidak digunakan kembali, maka panjang kunci harus sesuai dengan panjang plainteks. Hal ini menimbulkan inefisiensi dalam penyimpanan. Selain itu, terdapat masalah bagaimana mendistribusikan kunci tersebut sehingga aman sampai tujuan.

Bahkan jika kriptografer berhasil memecahkan masalah pendistribusian kunci dan penyimpanan, kriptografer masih harus menjamin bahwa pengirim dan penerima harus tersinkronisasi. Jika penerima kehlangan satu buah bit saja, pesan menjadi tidak berarti. Atau, jika beberapa bit diganti saat transmisi, hanya bit-bit tersebut yang akan didekripsi secara salah. Dengan kata lain, *one-time pad* tidak menyediakan fitur autentikasi.

Berdasarkan penjelasan singkat mengenai *one-time pad*, kelebihan utama dari algoritma ini adalah jaminan keamanannya yang sangat tinggi. *One-time pad* merupakan kriptosistem ideal yang tidak dapat dipecahkan. Namun, terdapat kekurangan mendasar yang menyebabkan algoritma ini tidak mungkin diaplikasikan, seperti masalah (1) pembuatan kunci, (2) pendistribusian kunci, (3) pemborosan penyimpanan, dan (4) deteksi kesalahan dan autentikasi.

Tugas dari kriptografer adalah meminimalisasi tingkat masalah dari *one-time pad* atau dengan kata lain meningkatkan efisiensi pemakaian algoritma *one-time pad*. Jika kita mengorelasikan karakteristik dari *cloud computing* dengan masalah yang ada pada *one-time pad*, maka terdapat suatu simbiosis mutualisme. Kelebihan dari sistem *cloud computing* dapat menaikkan efisiensi dari *one-time pad*. Sebaliknya, *one-time pad* dapat menambah keamanan dari *cloud computing* (salah satu kekurangan *cloud computing*).

Untuk itu, penulis mencoba merancang kriptosistem yang serupa dengan *one-time pad*, namun terkhususkan pada sistem *cloud computing*. Secara garis besar, sistem yang diusulkan penulis ini hanya mencoba memecahakan permasalahan yang ada pada *one-time pad* yang lama.

Konsep utama dari kriptosistem ini adalah, berangkat dari pengertian awal *one-time pad*, bahwa panjang plainteks, kunci, dan cipherteks adalah sama. Sehingga, penulis berpikir bahwa proses enkripsi sama dengan proses menduplikasi berkas menjadi dua buah berkas lain yang saling tidak berhubungan. Kedua buah berkas ini masing-masing dapat berperan baik sebagai cipherteks maupun sebagai kunci. Proses dekripsi adalah proses menggabungkan kedua buah berkas ini menjadi satu dengan menggunakan operasi XOR.

Sekarang kita coba menganalisis tingkat keamanannya. Pertama, karena kriptosistem ini dipakai pada tingkat penyimpanan (bukan pada tingkat pengiriman pesan), maka kita bisa menganggap bahwa baik pengirim maupun penerima adalah orang yang sama, sehingga masalah pendistribusian kunci telah terselesaikan. Kedua, masalah pembuatan kunci juga telah terselesaikan karena kunci tidak perlu repot-repot dibuat. Kita hanya secara acak membuat dua file yang berbeda. Ketiga, pemborosan penyimpanan terselesaikan karena pihak pengguna meletakkan kedua berkas tersebut di *cloud* sehingga penyimpanan data tidak boros.

Untuk lebih jelasnya, misal terdapat sebuah plainteks m. Kriptosistem cukup melakukan proses sedemikian hingga terbentuk dua buah file yang berbeda. Kedua buah file ini kita sebut c1 dan c2. Baik c1 dan c2 mampu berperan ganda, yaitu sebagai cipherteks dan plainteks. Kemudian, pengguna tinggal menyimpan kedua buah file tersebut di *cloud*. Akibat dari penyimpanan di *cloud* ini, kedua file tersimpan secara acak, dan karena telah terenkripsi, maka kedua file ini tidak memiliki arti di dunia *cloud*. Pengguna menyimpan alamat kedua buah file secara privat di penyimpanan lokal, sehingga jika pengguna ingin menggunakan file tersebut, cukup menggabungkan kedua buah file, c1 dan c2. Bagi pihak-pihak yang tidak memiliki wewenang, mereka tidak mampu membaca isi berkas c1 maupun c2, jika mereka tidak mengetahui di mana berkas pasangannya disimpan. Melakukan kriptanalisis dengan cara biasa juga nyaris tidak mungkin dikarenakan pengacakannya tersebut dan tidak memiliki pola.

Kemudian, jika kita ingin mengirimkan pesan tersebut ke seseorang, maka kita cukup mengirimkan alamat dari pesan tersebut yang disimpan di *cloud*. Namun, kriptosistem ini tidak mampu memecahkan masalah autentikasi dan integritas data karena untuk memecahkan masalah tersebut dibutuhkan fungsi *hash*.

Bahkan, kriptosistem ini dapat dimodifikasi dengan cara membagi plainteks menjadi beberapa file dan disimpan di tempat yang berbeda di *cloud*. Hal ini akan mempersulit kriptanalis untuk menemukan di mana saja cipherteks yang bersesuaian disimpan.

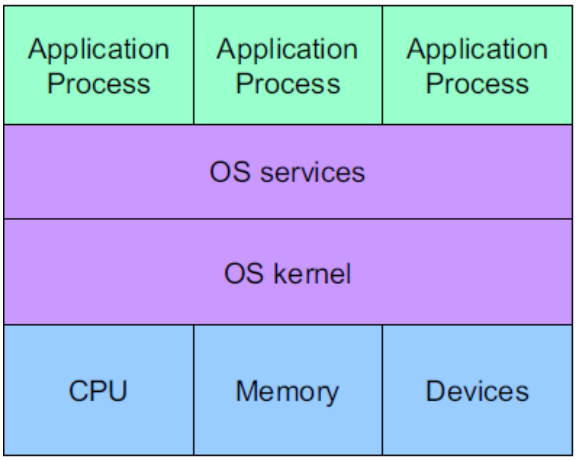
4. ALGORITMA AES PADA SISTEM CLOUD COMPUTING

4.1. JENIS-JENIS SERANGAN UMUM PADA KRIPTOSISTEM

Kriptosistem dapat diukur tingkat keamanannya berdasarkan pembuktian bahwa cipherteks tersebut hanya dapat dipecahkan satu-satunya cara dengan menggunakan teknik *brute force*. Banyak algoritma-

algoritma yang mengklaim bahwa tekniknya aman dikarenakan hanya mampu dipecahkan dengan cara *brute force*. Sedangkan, bagi kriptanalis, mereka akan berusaha sekuat tenaga untuk memecahkan cipherteks tanpa menggunakan teknik *brute force*. Untuk itu, kriptanalis dengan kreatif menggunakan cara-cara untuk menekan pemakaian *brute force*. Cara tersebut dapat dibagi menjadi tiga, yaitu serangan fisik, serangan tradisional, dan serangan arsitektural. Dalam studi kasus algoritma AES dalam sistem komputasi tradisional, kami lebih focus pada serangan arsitektural, karena serangan ini lebih ampuh daripada serangan-serangan lainnya.

Idealnya, arsitektur computer, seperti tampak pada gambar 4, terdiri atas proses aplikasi, layanan sistem operasi, kernel sistem operasi, CPU, memori, dan perangkat. Setiap proses aplikasi menggunakan CPU untuk menjalankan prosesnya. Komunikasi antar proses ini berpotensi menimbulkan *side channels* dan *covert channels*. Di dunia saat ini, CPU sudah memiliki banyak *core*, sehingga menunjang paralelisme CPU, seperi *multicore* dan *simultaneous multithreading*. Paralelisme ini juga berpotensi menimbulkan *side channels* skala mikro karena terdapat komunikasi antar inti dan komunikasi antar *thread*. Masing-masing *channels* ini berpotensi menimbulkan masalah keamanan, seperti *cache attack*.



Gambar 4 : Arsitektur Komputer

4.2. CACHE ATTACK

Cache attack merupakan serangan yang murni dari perangkat lunak dan sama sekali tidak berhubungan dengan kode kriptografi. Serangan ini terbukti sangat efisien dan menyebabkan terjadinya serangan terhadap *side-channels*. Mengapa menggunakan serangan terhadap *cache*? Pertama, karena semakin maju, kecepatan inti prosesor naik sebesar 60 persen dan kecepatan memori hanya naik sebesar 7-9 persen. Perbedaan kecepatan pemrosesan ini dapat menyebabkan bocornya data saat CPU hendak mengambil data dari memori atau sebaliknya. Kedua, selisih *latency* dari CPU dan memori yang sangat besar, yaitu 0,3 ns dan 50-150 ns. Adanya perbedaan waktu ini juga berpotensi menyebabkan munculnya *side channels*.

Hal-hal di atas, jika dimanfaatkan dengan baik akan menyebabkan kebocoran informasi, khususnya informasi alamat memori. Pertama, cache merupakan sumber daya yang dibagi. Status cache memperNgaruhi dan dipengaruhi oleh semua proses, sehingga menyebabkan adanya *crosstalk* antar proses. Data yang berada di cache telah dilindungi dengan baik. Namun, metadata membcoorkan informasi tentang pola akses memori, dengan cara alamat-alamat dari memori yang diakses. Jadi, daripada menyerang data di dalam *cache* yang notabene telah terlindungi, maka sebaiknya menyerang metadatanya (didapat dari info status dari cache) yang mengandung alamat dari akses memori.

Pada gambar 5, tampak bahwa 64 bytes blok memori berasosiasi dengan 64 bytes cache line. Jika, table S-Box pada AES disimpan pada memori, maka penyerang akan berusaha mendeteksi cara akses ke table AES. Hal ini bisa dilakukan berdasarkan implementasi standar AES pada perangkat lunak, yaitu

char p[16], k[16]; // plainteks dan kunci

int32 T0[256], T1[256], T2[256], T3[256]; //table lookup

int32 Col[4]; //status perantara

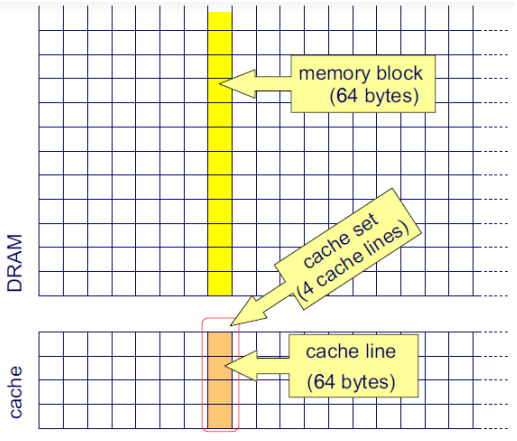
//Round 1

Col[0] ← T0[p[0] ⊕k[0]] ⊕ T1[p[5] ⊕k[5]] ⊕ T2[p[10] ⊕k[10]] ⊕ T3[p[15] ⊕k[15]] ;

Col[1] ← T0[p[4] ⊕k[4]] ⊕ T1[p[9] ⊕k[9]] ⊕ T2[p[14] ⊕k[14]] ⊕ T3[p[3] ⊕k[3]] ;

Col[2] ← T0[p[8] ⊕k[8]] ⊕ T1[p[13] ⊕k[13]] ⊕ T2[p[2] ⊕k[2]] ⊕ T3[p[7] ⊕k[7]] ;

Col[3] ← T0[p[12] ⊕k[12]] ⊕ T1[p[1] ⊕k[1]] ⊕ T2[p[6] ⊕k[6]] ⊕ T3[p[11] ⊕k[11]] ;



Gambar 5 : Asosiasi Cache Memori

4.3. SERANGAN SINKRON

Sebuah software melakukan enkripsi AES atau dekripsi menggunakana kunci rahasia. Kemudian, penyerang juga menjalankan proses di CPU yang sama. Penyerang sedemikian rupa sehingga dapat mengubah plainteks yang diketahui. Contoh dari serangan sinkron ini adalah partisi enkripsi disk dan sistem file dan VPN. Sehingga, dengan serangan ini, penyerang dapat mengetahui kunci rahasia.
Pengukuran *noisy* dari penggunaan cache dilakukan berdasarkan banyaknya enkripsi yang dilakukan pada plainteks yang diketahui. Kemudian, penyerang menebak satu byte pertama dari kunci. Untuk setiap hipotesis, untuk setiap plainteks contoh, prediksi cache line mana yang diakses oleh **T0[p[0]XORk[0]]**. Setelah itu, identifikasi hipotesis yang menghasilkan maksimum korelasi antara prediksi dan pengukuran. Lalu, lanjutkan untuk bytes sisa dan gunakan hasil sementara ini untuk menentukan kunci di putaran selanjutnya hingga mampu menemukan kunci asli. Cara terakhir dapat dilakukan dengan metode penyerangan S-Box secara non-linear.

4.4. SERANGAN ASINKRON

Seseorang menjalankan komputasi enkripsi dengan menggunakan kunci rahasia tertentu. Penyerang menjalankan proses di CPU yang sama pada waktu yang belum tentu sama. Plainteks atau cipherteks memiliki distribusi yang tidak seragam, seperti menggunakan bahasa Inggris, ada data yang diformat, adanya header, dll. Contohnya adalah menggunakan kriptografi pada sistem banyak pengguna.

Metodenya adalah membandingkan dua distribusi, yaitu antara statistic akses memori yang diukur dan statistic akses memori yang diprediksi, yang didapat dari distribusi plainteks dan hipotesis key. Kunci dapat ditemukan dengan mencari korelasi terbaik antara keduanya.

4.5. SERANGAN LAIN

Serangan lain yang dapat dilakukan sehingga mengeksploitasi kerja CPU adalah *hyper attack*. Metode ini mengeksploitasi paralelisme dalam sistem CPU modern. Penyerang mencari sistem yang hyperthreading, multi-core, shared cache, dan interrupt-based. Selain itu, terdapat serangan yang menyerang *covert channels*, menggunakan bantuan hardware seperti melacak daya CPU, penyerangan berdasarkan waktu, dll.

Pemaparan di atas lebih terfokus pada penyerangan pada *cache*. Untuk penyerangan arsitektural lain antara lain, eksploitasi sumber daya ALU (khususnya unit perkalian), prediksi operasi perkalangan (eksploitasi pemakaian if pada code dan kenyataan bahwa state BP merupakan sumber daya yang digunakan bersama), dll.

4.6. IMPLIKASI

Implikasi dari penyerangan berbasis arsitektural tersebut adalah penyerangan berfokus pada sistem berbasis banyak pengguna. Sistem multiuser menjadi sasaran empuk serangan ini. Selain itu, virtual machine juga lemah terhadap serangan ini. Serangan-serangan juga dapat disamarkan melalui kode yang ridak dipercaya, seperti ActiveX, Java applet, Javascript,dll. Selain kode yang tidak dipercaya, bentuk serangan lain adalah serangan berbasis remote network.

Jika kita menghubungkannya dengan *cloud computing*, maka *cache attack* menjadi resiko keamanan utama dari sistem ini. Pertama, prinsip dari *cloud computing* yang serba berbagi sumber daya komputasi menjadi boomerang jika kita melihat daya serang cache attack yang terletak pada kemampuannya mengeksploitasi sumber daya yang berbagi tersebut. Kedua, *cloud computing* merupakan sistem yang *hyper threading* sehingga menimbulkan banyaknya *side channel* yang siap dimanfaatkan untuk penyerangan-penyerangan. Apalagi implementasi *cloud computing* tidak lepas dari penggunaan Virtual Machine sebagai media virtualisasinya. Padahal virtual machine merupakan sasaran empuk bagi serangan-serangan berbasis arsitektural.

4.7. SOLUSI

Untuk mengatasi masalah tersebut, penulis mengajukan beberapa saran, yang penulis bagi menjadi tiga bagian, yaitu berdasarkan hardware atau platform, perubahan kriptosistem, dan perlindungan terhadap fungsi-fungsi khusus.

Solusi yang melibatkan hardware atau platform meliputi, pertama, mengunci cache. Dengan mengunci cache, maka serangan-serangan menjadi percuma karena dia tidak dapat melakukan pengukuran terhadap akses data antara cache dan memori. Kedua, mengacak mapping antara memori dan cacheline. Ketiga, dibuat mode secure saat melakukan enkripsi atau dekripsi. Keempat menambahkan instruksi AES ke dalam chip prosesor baru. Dengan ditambahkannya instruksi AES yang telah terintegrasi langsung, menyebabkan intrusi ke memori menjadi tidak berguna lagi, karena tidak melibatkan kode program lagi.

Solusi yang melibatkan pengubahan kriptosistem meliputi, pertama, enkripsi homomorpik penuh. Algoritma ini masih dalam tahap pembuatan karena algoritma ini merupakan algoritma ideal dan sampai saat ini belum ada orang yang mampu membuatnya. Pakar kriptografi semacam Ron Rivest telah mencoba memecahkan permasalahan ini selama 30 tahun dan beliau belum mampu memecahkannya.

Solusi yang melibatkan perlindungan terhadap fungsi-fungsi spesifik, meliputi, pertama, memilih primitive yang tepat. Maksudnya, terkadang algoritma Rijndael tidak tepat digunakan dalam suatu

permasalahan dan perlu menggunakan algoritma Serpent dalam memecahkan permasalahan tersebut. Kedua, AES dengan menggunakan bitslicing. Maksudnya, terkadang perlu memecah bit-bit dalam byte sehingga memberikan kerumitan pada kriptosistem.

5. KESIMPULAN

- *Cloud computing* merupakan arsitektur masa depan dunia computer yang memberikan banyak keuntungan, seperti kemurahan, efisiensi, dan tingginya tingkat ekonominya.
- Namun, di balik kelebihannya, *cloud computing* menyimpan kelemahan yang besar dalam keamanannya.
- Masalah terbesar dalam keamanannya dikarenakan prinsip utama *cloud computing* yang membagi sumber dayanya, sehingga jika tidak dijaga dengan baik, maka setiap orang dapat melakukan akses ke seiap sumber daya.
- Untuk itu, diperlukan enkripsi yang kuat dan adanya manajemen kunci yang baik.
- Algoritma one-time pad dapat digunakan pada sistem *cloud computing*, dengan prinsip rancangan modifikasinya adalah membagi plainteks menjadi cipherteks dan kunci, dan mendistribusikan kedua berkas tersebut secara acak di dunia *cloud*. Pengguna atau pemakai diberikan akses alamat kedua file tersebut.
- Algoritma AES masih tetap dapat dipakai di *cloud computing* karena daya keamanannya yang tinggi dan efisien. Jadi, memang pada dasarnya sistem *cloud computing* ini masih memakai dua algoritma yang popular, yaitu AES dan RSA.
- Namun, perlu ditinjau bahwa karakteristik dari *cloud computing* yang berbagi ini menyebabkan diperlukan adanya modifikasi dari algoritma AES. Tampak bahwa algoritma AES sangat rentan terhadap serangan berbasis arsitektural, khususnya yang menyerang *cache* atau memori yang terbagi.

6. DAFTAR PUSTAKA

Cobb, Chey. *Cryptography for Dummies.*

Hoffstein, Jeffrey, etc. *An Introduction to Mathematical Cryptography*.

Lauter, Kristin. *Cryptographic Cloud Storage.*

Munir, Rinaldi, Diktat Kuliah IF5054 Kriptografi, Penerbit ITB 2006.

Nguyen, Ninh. *Cloud Computing Security.*

Schneier, Bruce. *Applied Cryptography, 2nd Edition: Protocols, Algorithms, and Source Code in C.*

Tromer, Eran. *Architecural Side Channels in Cloud Computing*

Tromer, Eran. *Efficient Cache Attacks on AES, and Countermeasures*.