Kriptografi pada *HTTP Request* (*Post and Get*) dengan Menggunakan Algoritma Kunci Publik

Joel T.H.P. Hutasoit *(13504144)*
Jurusan Teknik Informatika ITB Bandung, Email: if14144@students.if.itb.ac.id

Abstract *– HTTP Request merupakan suatu mekanisme pengiriman data pada protokol HTTP yang merepresentasikan aksi yang akan dilakukan pada suatu sumberdaya (server) yang sudah diidentifikasikan sebelumnya. Data yang dipertukarkan pada mekanisme ini masih berupa plainteks sehingga tidak akan menjamin kerahasiaan dan integritas datanya jika terjadi serangan seperti penyadapan paket data. Untuk itu pada makalah ini akan dibahas tentang implementasi kriptografi kunci publik serta prinsip manajemen kunci pada mekanisme HTTP Request baik secara POST maupun GET dari suatu aplikasi berbasis web untuk mengatasi serangan tersebut.*

Kata Kunci *: HTTP Request, HTML Form, Kunci Publik, RSA, Manajemen Kunci*

1. PENDAHULUAN

HTTP Request merupakan suatu mekanisme pengiriman data dari ***client*** pada protokol *HTTP* yang merepresentasikan aksi yang akan dilakukan pada suatu sumberdaya (*server*) yang sudah diidentifikasikan sebelumnya. Pihak ***server*** yang menerima *request* akan menjawabnya melalui mekanisme *HTTP Response. HTTP Request* pada dasarnya mendefenisikan 8 metode aksi yang akan dilakukan pada suatu sumber daya yaitu *Head, Get, Post, Put, Delete, Trace, Options* dan *Connect*. Pada eksplorasi kali ini dibatasi hanya pada metode *POST* dan *GET*. Salah satu penggunaanya adalah mekanisme pengiriman data melalui suatu ***HTML FORM*** pada suatu situs web.

Pada *HTTP Request* dengan metode *GET*, data permintaan dikirimkan sebagai representasi alamat dari sumberdaya (*resource*) yang akan dituju. Kelemahan dari *HTTP GET Request* ini adalah keterbatasan dari panjang alamat sumberdaya (URL) serta data yang dikirimkan akan kelihatan pada alamat sumber daya yang dituju. Sedangkan pada metode *POST* data yang akan dikirimkan disisipkan pada *body* dari *request* yang bersangkutan sehingga data yang dikirimkan bisa lebih panjang dan tidak kelihatan secara langsung para alamat sumberdaya (URL). Contoh potongan HTTP GET Request dapat dilihat pada gambar 1.

```
GET /index.php?data=xxxxx HTTP/1.1
Host: www.example.com
```

Gambar 1 : *HTTP GET Request*

Pada dasarnya baik metode *GET* dan *POST* pada *HTTP Request* mengirimkan data *request*-nya yang asli langsung tanpa melalui proses enkripsi (*plaintext*) pada lapisan aplikasi (*application layer*) dari TCP/IP. Hal ini dikarenakan setiap interaksi pada *HTTP Request* merupakan *ASCII request*. Hal ini tentunya akan merugikan pihak pengirim karena data yang dikirimkan tidak terjamin kerahasiaannya dan integritasnya dikarenakan pihak ketiga bisa saja menyadap *HTTP Request* yang dikirim (seperti serangan *packet sniffing*). Untuk itu data *request* yang dikirimkan seharusnya dienkripsi jika data tersebut bersifat rahasia. Hal ini sebenarnya sudah dapat ditanggulangi dengan menggunakan protokol ***HTTPS***. Akan tetapi belum semua aplikasi *browser* yang mendukung protokol ini. Untuk itu pada eksplorasi kali ini akan dicoba menerapkan mekanisme **kriptografi kunci publik** (dengan menggunakan algoritma RSA) pada *request message* dari protokol HTTP sebagai solusi untuk permasalahan diatas. Aktor-aktor yang terkait dalam eksplorasi ini adalah :

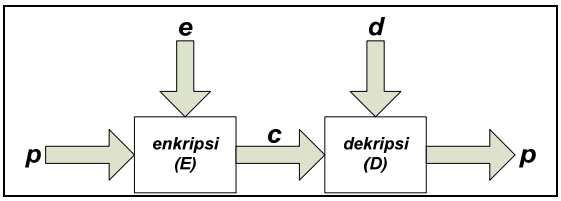
- *SERVER SIDE* A
- *CLIENT SIDE* B
- JARINGAN INTERNET sebagai media pengiriman data yang menggunakan protokol HTTP

Dengan menerapkan kriptografi kunci publik ini maka diharapkan hanya SERVER A dan CLIENT B saja yang mengetahui data yang dikirimkan melalui JARINGAN INTERNET dimana akan digunakan ***protokol pertukaran kunci*** yang tepat.

2. IMPLEMENTASI KRIPTOGRAFI KUNCI PUBLIK SERTA MANAJEMEN KUNCI PADA *HTTP REQUEST*

Kriptografi pada umumnya mempunyai *constraint* dalam pengiriman pesan rahasia antara lain melibatkan dua pihak yaitu pengirim (*A*) dan penerima (*B*) pesan rahasia, melakukan dua proses yaitu enkrispi (*E*) dan dekripsi (*D*) serta terdefenisinya tiga objek yaitu plainteks (*p*), chiperteks (*c*) dan kunci (*k*) yang bersangkutan. Khusus untuk kriptografi kunci publik, baik pengirim pesan rahasia maupun penerima pesan rahasia memiliki sepasang kunci yaitu satu kunci untuk enkripsi dan satu kunci untuk dekripsi. Kunci untuk enkripsi (kunci publik – *e*) akan diumumkan ke publik karena tidak bersifat rahasia sedangkan kunci untuk dekripsi (kunci privat – *d*) bersifat rahasia. Diluar *constraint* ini merupakan

ancaman terhadap kerahasiaan dari pesan yang dikirimkan seperti munculnya proses serangan dari pihak ketiga (seperti penyadap informasi) dan sebagainya. Secara umum kriptografi ini dapat digambarkan pada gambar 2. Adapun algoritma kunci publik yang akan dipakai adalah seperti yang dikemukakan pada bagian pendahuluan yaitu RSA.



Gambar 2 : Skema Kriptografi Kunci Publik

Sebagai langkah awal implementasi maka terlebih dahulu dilakukan abstraksi mekanisme pengiriman data maupun informasi pada *HTTP Request* terhadap *constraint* yang ada pada sistem kriptografi kunci publik. Hal ini akan dijabarkan pada tabel 1.

Tabel 1 : Abstraksi *HTTP Request* terhadap Kriptografi Kunci Publik

Constraint **Kriptografi Kunci Publik**	**Abstraksi *HTTP Request***
Pengirim (*A*)	*Client Computer*
Penerima (*B*)	*Server Computer*
Proses Enkripsi (*E*)	*Client Side*
Proses Dekripsi (*D*)	*Server Side*
Plainteks (*p*)	Data / Informasi yang akan dikirim
Chiperteks (*c*)	Data / Informasi pada *GET / POST Request*

Setelah melakukan abstraksi maka perlu ditetapkan prosedur pertukaran kunci dalam implementasi ini. Dengan memperhatikan abstraksi sebelumnya dimana proses enkripsi terjadi di sisi *client* dan proses dekripsi terjadi di sisi *server* maka ditetapkan prosedur pertukaran kunci sebagai berikut :

- Pasangan Kunci (*e* dan *d*) terlebih dahulu dibangkitkan di sisi *server*.
- Kunci publik *e* dan kunci privat *d* disimpan oleh sisi *server*
- Kunci publik *e* dikirimkan ke sisi *client*

Langkah berikut yang perlu dilakukan adalah menetapkan prosedur komunikasi data (PKD) dimana disesuaikan dengan prosedur pertukaran kunci yang sudah ditetapkan sebelumnya. Prosedur ini akan dibagi dua yaitu pada sisi *server* dan pada sisi *client* Adapun prosedur untuk sisi *server* yaitu :

- Terlebih dahulu diciptakan sesi komunikasi untuk melakukan pertukaran data / informasi melalui *HTTP Request* jika belum terbentuk.
- Bangkitkan pasangan kunci untuk sesi komunikasi yang bersangkutan jika belum ada pasangan kunci yang disimpan. Pasangan kunci ini akan dipegang (tidak akan membangkitkan pasangan kunci yang baru) jika sesi komunikasi data / informasi belum berakhir.
- Jika terjadi pembangkitan kunci, simpan kunci publik *e* dan kunci privat *d* tersebut.
- Kirimkan kunci publik *e* ke sisi *client* bersamaan dengan dokumen HTML yang berkaitan (disisipkan pada dokumen HTML) jika dokumen HTML tersebut akan memungkinkan terjadinya pengiriman data melalui form HTML dengan menggunakan mekanisme *HTTP Request*.
- Jika terjadi *HTTP Request* dari sisi *client* maka terlebih dahulu lakukan proses dekripsi terhadap data / informasi dari *HTTP Request* dengan menggunakan kunci *d* sebelum dilanjutkan ke proses berikutnya dikarenakan data / informasi tersebut masih dalam bentuk chiperteks.

Sedangkan prosedur untuk sisi *client* adalah :

- Jika *user* melakukan *submit* data/informasi yang ada pada form HTML maka terlebih dahulu lakukan proses enkripsi terhadap data / informasi yang ada pada setiap elemen form dengan menggunakan kunci *e* yang sudah disisipkan pada dokumen HTML. Setelah itu baru proses *submit* dilakukan ke sisi *server*.

Hal terakhir yang dilakukan adalah implementasi PKD pada sistem aplikasi berbasis web yang sebenarnya, baik di sisi *server* maupun di sisi *client*. Di dalam implementasi, sisi *server* akan menggunakan bahasa *script* PHP sedangkan di sisi *client* akan menggunakan bahasa *scritp* Javascript. Secara umum implementasi PKD pada sisi *server* dengan menggunakan bahasa *script* PHP adalah sebagai berikut :

- Sesi komunikasi data dibangun dengan menggunakan *session_id* baik pada pembentukannya maupun penghapusannya.
- Kunci *e* dan *d* akan disimpan di sisi *server* yaitu sebagai elemen dari *array session* (`$HTTP_SESSION_VARS`) yang merupakan variabel server.
- Data / informasi *HTTP Request* diterima sebagai salah satu dari dua *array* berikut, yaitu `$HTTP_POST_VARS` jika dikirimkan sebagai *POST request* dan `$HTTP_GET_VARS` jika dikirimkan sebagai *GET request*.
- Pembuatan 5 buah fungsi yaitu fungsi `genKey` untuk proses pembangkitan kunci, fungsi `encDec` untuk proses dekripsi serta fungsi `isPrime`, `relPrime` dan `euler` untuk mendukung fungsi `genKey`. Keseluruhan implementasi fungsi ini akan dijadikan sebagai suatu modul dalam satu file.

Sedangkan untuk implementasi di sisi *client* dengan menggunakan bahasa *script* Javascript adalah sebagai berikut :

- Pembuatan sebuah fungsi yaitu fungsi `encDec` untuk proses enkripsi, dimana fungsi ini akan dipanggil ketika terjadi *event onSubmit* pada form HTML yang bersangkutan.
- Kunci enkripsi pada awal pemuatan (*loading*) dokumen HTML disisipkan sebagai elemen form HTML yang bertipe *hidden*. Elemen form ini nantinya akan dihapus sehingga tidak akan diikutkan saat melakukan *HTTP Request*.

Implementasi dilakukan pada webserver virtual (menggunakan Apache 2.2) yang berjalan diatas sistem operasi Windows. Untuk keperluan pemantauan *HTTP Request (GET* dan *POST)* digunakan sebuah aplikasi bantuan diluar pemantauan dengan melihat *logfile* dari aplikasi webserver. Hal ini dilakukan agar data yang dikirimkan dengan metode *POST* bisa didapatkan. Aplikasi ini akan dijalankan sebagai *service* bersamaan dengan webserver.

3. HASIL DAN PEMBAHASAN

Setelah dilakukan implementasi maka didapatkan hasil dimana baik data yang dikirim dengan metode GET maupun POST sudah dalam bentuk terenkripsi (chiperteks) dan *server* dapat mendekripsikannya kembali ke plainteks semula dengan menggunakan kunci yang tersimpan di sisi *server*. Hal ini menunjukkan bahwa sudah terjadi sinkronisasi yang baik antara sisi *client* dan sisi *server* baik dalam mekanisme pembangkitan dan pertukaran kunci maupun pertukaran data. Hal ini dilihat dari hasil pantauan aplikasi bantuan yang dijalankan sebagai *service*. Adapun *HTTP Request* yang diterima *server* (*localhost*) adalah seperti yang ditunjukkan pada gambar 3 dan 4. Data yang dikirimkan pada percobaan kali ini adalah *"Hanya Mencoba OK"* dengan suatu pasangan kunci yang dibangkitkan saat pembentukan session.

```
    GET
/kripto/test_get.php?data=93115837%203680A078
%203B3849B5%20541FD53F%203680A078%20A5B6F97%2
01C1D0EEA%201F7A94E1%203B3849B5%20169453C7%20
29BFC468%202D59F186%203680A078%20A5B6F97%2067
98EECE%208609B877 HTTP/1.1
    Host: localhost
    Accept: */*
    Accept: image/gif
```

Gambar 3 : *HTTP GET Request* yang dihasilkan

```
    POST /kripto/test_post.php HTTP/1.1
    Host: localhost
    Accept: */*
    Accept: image/gif
    Content-length: 141
    93115837 3680A078 3B3849B5 541FD53F
3680A078 A5B6F97 1C1D0EEA 1F7A94E1 3B3849B5
169453C7 29BFC468 2D59F186 3680A078 A5B6F97
6798EECE 8609B877
```

Gambar 4 : *HTTP POST Request* yang dihasilkan

Dilihat dari ruang hasil *HTTP Request* yang dihasilkan dan waktu proses yang terpakai maka dapat dibuat suatu tabel performansi seperti pada tabel 2 berikut. Pengukuran ini didasarkan dengan melakukan perbandingan terhadap performansi mekanisme *HTTP Request* tanpa implementasi kriptografi kunci publik pada sistem komputer yang sama.

Tabel 2 : Hasil Performansi Implementasi

Atribut	Performansi
Kecepatan Proses	Lebih Lama (dimana *response* yang didapat pada *browser* / *client* yaitu > 3 detik sedangkan tanpa implementasi < 1 detik)
Kerahasiaan Data	Lebih Aman (terenkripsi)
Ukuran Data	Lebih Besar (dimana ukuran awal adalah 16 byte sedangkan chiperteks 141 byte)

Dari tabel di atas terlihat kelebihan utama dari implementasi ini yaitu dari segi ***kerahasiaan*** data yang lebih terjamin. Akan tetapi terdapat 2 kelemahan yang juga merupakan kelemahan dari sistem kriptografi kunci publik pada umumnya, yaitu dari segi ***waktu*** proses (khususnya di sisi *server*) dan ***ukuran*** data yang dihasilkan (dalam *byte*).

Lamanya proses di sisi *server* meliputi proses pembangkitan kunci dan proses dekripsi. Hal ini tentu akan berdampak lebih buruk pada saat implemntasi di server yang sebenarnya dikarenakan sebuah webserver tentu saja akan melayani banyak sesi komunikasi data pada saat yang bersamaan (konkuren). Sedangkan besarnya ukuran data diakibatkan oleh hasil perpangkatan dan modulo terhadap *byte-byte* data yang dienkripsi dengan menggunakan kunci berupa bilangan bulat yang juga bernilai besar. Untuk itu ada beberapa solusi yang dapat dilakukan untuk menekan kelemahan-kelemahan ini yaitu :

- Penyimpanan banyak pasangan kunci ataupun bilangan prima pada suatu basis data. Hal ini tentu akan memberi dampak positif dikarenakan proses pembangkitan kunci di sisi *server* tidak perlu lagi dilakukan tetapi cukup hanya dengan membangkitkan bilangan acak untuk menentukan *id* dari suatu pasangan kunci yang akan dipakai. Tentunya dengan terbatasnya jumlah pasangan kunci yang akan dipakai membuat sistem kriptografi ini menjadi lebih lemah, akan tetapi beban *server* akan jauh menjadi lebih ringan jika aktivitas konkuren semakin banyak dan waktu proses akan menjadi lebih singkat.
- Penggunaan sistem kriptografi dengan mengkombinasikan kriptografi kunci publik dengan kriptografi kunci simetri (*hybrid*).

Dalam hal ini data/informasi akan dienkripsi dengan menggunakan kunci simetri sedangkan kunci simetri itu sendiri akan dipertukarkan dengan menggunakan kriptografi kunci publik. Hal ini tentu akan menghemat waktu proses (enkripsi / dekripsi) dan juga memperkecil ukuran data dari *HTTP Request*.

- Selain menggunakan sistem *hybrid*, metode lain yang memungkinkan adalah dengan penggunaan bilangan bulat yang kecil dalam pembangkitan kunci yang berakibat pada pendeknya kunci yang dihasilkan. Hal ini tentu akan memperkecil ukuran data dan juga waktu proses, akan tetapi keamanan sistem kriptografi menjadi semakin lemah. Hal ini dapat diatasi dengan membuat suatu variabel *integer N* di sisi *server* yang menyatakan status jumlah pemakaian kunci yang akan dibangkitkan. Adapun aturan yang akan dilakukan adalah pemakaian kunci hanyalah untuk *x* operasi komunikasi data. Hal ini dilakukan untuk memperkuat keamanan sistem kriptografi jika dalam suatu sesi komunikasi data banyak melakukan *HTTP Request* dengan membatasi jumlah pemakaian kunci yang pendek tersebut. Dalam hal ini suatu pasangan kunci baru akan dibangkitkan lagi jika nilai variabel N sama dengan x walaupun sesi komunikasi data belum berakhir. Sehingga untuk setiap proses dekripsi yang dilakukan oleh *server* pada suatu sesi komunikasi data maka nilai variabel N akan di-*increment* dan jika suatu pasangan kunci baru dibangkitkan maka nilai N diubah menjadi 0.

Dari ketiga solusi di atas dapat dikombinasikan dimana penggunaan bilangan bulat yang kecil akan dilakukan saat pembangkitan pasangan kunci dalam jumlah yang banyak untuk kemudian disimpan dalam suatu tabel pada basis data. Pasangan kunci ini nantinya akan dipilih secara acak yang kemudian pemakaiannya dalam suatu sesi komunikasi data dikelola dengan menggunakan suatu variabel N seperti yang sudah dikemukakan sebelumnya. Pasangan kunci ini akan dipakai untuk melakukan enkripsi dan dekripsi terhadap ***kunci simetri*** yang digunakan untuk mengenkripsi / mengenkrispi data maupun informasi dari *HTTP Request*. Dengan mengimplementasikan kombinasi solusi ini maka masalah lamanya proses dan besarnya ukuran data akan dapat lebih ditekan.

4. KESIMPULAN

HTTP Request merupakan suatu mekanisme pengiriman data pada protokol HTTP yang digunakan pada saat melakukan pengiriman data dari sisi *client* ke sisi *server*. Interaksi pada mekanisme ini merupakan *ASCII request* sehingga data yang dipertukarkan masih berupa plainteks dan tentunya tidak akan menjamin kerahasiaan dan integritas datanya jika terjadi serangan seperti penyadapan paket data (*packet sniffing*). Untuk itu implementasi kriptografi kunci publik serta prinsip manajemen kunci pada mekanisme *HTTP Request* baik secara *POST* maupun *GET* dari suatu aplikasi berbasis web merupakan salah satu cara yang aman untuk mengatasi serangan tersebut. Akan tetapi implementasi ini memiliki kelemahan dalam hal lamanya proses yang dilakukan dan bertambah besarnya ukuran data yang dipertukarkan. Dengan menerapkan suatu kombinasi solusi mulai dari penggunaan basis data, sistem *hybrid* hingga pembatasan penggunaan kunci maka kelemahan-kelemahan tersebut dapat lebih ditutupi.

DAFTAR REFERENSI

[1] Munir, Rinaldi. *Diktat Kuliah IF5054 Kriptografi*. Institut Teknologi Bandung. 2006

[2] http://tools.ietf.org/html/rfc2616 (*HTTP Specification*)

[3] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html (*HTTP Specification*)