Penerapan Algoritma Kriptografi RSA dalam Pengiriman Data Melalui *Socket* Berbasis TCP/IP

Hafni Syaeful Sulun

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganesha 10 Bandung 40132
email: if15058@students.if.itb.ac.id

Abstrak – *Dalam dunia internet tidak ada yang benar-benar aman. Selalu saja ada celah dalam setiap aplikasi yang dibuat. Begitu juga dalam pengiriman data melalui socket. Untuk meminimalisasi serangan terhadap pengiriman data biasa diterapkan kriptografi. Salah satu algoritma kriptografi yang cukup populer adalah algoritma RSA. Dalam makalah ini akan dibahas penerapan algoritma kriptografi RSA dalam pengiriman data melalui socket berbasis TCP/IP. Untuk mengujinya dibuat sebuah program socket klien-server dengan menggunakan PHP.*

Kata Kunci*: kriptografi, RSA, socket, TCP/IP, PHP.*

1 PENDAHULUAN

Socket merupakan metode yang *protocol independent* untuk membuat koneksi antarproses [3]. Jenis *socket* berdasarkan jenis koneksi yang digunakannya ada dua, yaitu *connectionless* dan *connection-oriented*. Dalam makalah ini *socket* yang digunakan adalah *socket* jenis *connection-oriented* karena jenis ini lebih sering digunakan. Jenis ini menggunakan TCP/IP dalam membangun koneksi sehingga disebut *socket* berbasis TCP/IP.

Dua proses yang melakukan koneksi adalah proses server dan proses klien. Proses server mengikat suatu nomor *port* tertentu dan menunggu permintaan dari klien. Agar dapat tersambung dengan proses server, proses klien harus mengetahui alamat IP atau *hostname* komputer yang menjalankan proses server dan nomor *port* yang digunakan untuk mendengarkan permintaan klien.

Agar pengiriman data antarproses melalui *socket* tersebut lebih aman, diterapkan salah satu algoritma kriptografi untuk mengenkripsi data yang dikirimkan. Algoritma yang dipiilih adalah algoritma RSA. Dalam kriptografi terdapat dua jenis algoritma, yaitu algoritma simetri (kunci enkripsi sama dengan kunci dekripsi) dan algoritma nirsimetri (kunci enkripsi tidak sama dengan kunci dekripsi). Algoritma simetri disebut juga algoritma kunci pribadi (*private key*) karena kunci enkripsi dan dekripsi sama dan harus dirahasiakan. Algoritma nirsimetri disebut juga algoritma kunci publik (*public key*) karena kunci enkripsi berbeda dengan kunci dekripsi dan tidak bersifat rahasia, tetapi kunci dekripsi bersifat rahasia. Algoritma RSA termasuk ke dalam algoritma simetri dan kunci publik.

2 PEMBAHASAN

2.1 Socket

Dalam bahasan *socket* dikenal primitif-primitif untuk melakukan koneksi menggunakan *socket*. *Socket* yang paling populer adalah *socket* Berkeley yang memiliki primitif sebagai berikut [6]:

Primitif	Arti
SOCKET	Membuat komunikasi *socket* baru
BIND	Menyematkan alamat lokal ke *socket*
LISTEN	Mengumumkan bahwa *socket* siap menerima koneksi dan mengalokasikan ruang antrian
ACCEPT	Menunggu hingga ada permintaan koneksi
CONNECT	Mencoba untuk membuat koneksi
SEND	Mengirim data melalui koneksi
RECV	Menerima data dari koneksi
CLOSE	Melepaskan koneksi

Empat primitif pertama dalam tabel di atas dieksekusi secara berurutan oleh server. Pertama, primitif SOCKET digunakan untuk membuat *socket* baru. Primitif ini menggunakan tiga parameter, yaitu format alamat, tipe layanan, dan protokol. Jika *socket* baru berhasil dibuat, primitif ini akan mengembalikan *file descriptor* yang akan digunakan untuk bertukar data antara server dengan klien.

Socket yang baru saja dibuat tersebut belum mempunyai alamat. Untuk menyematkan alamat digunakan primitif BIND.

Primitif selanjutnya adalah LISTEN. Dengan mengeksekusi primitif ini berarti server telah siap untuk menerima koneksi. Setelah itu, server mengalokasikan ruang untuk antrian koneksi yang datang.

Selanjutnya, primitif ACCEPT digunakan untuk menunggu hingga ada permintaan koneksi. Jika sebuah permintaan koneksi datang, maka akan dibuatkan *socket* baru dengan properti yang sama dengan *socket* yang dibuat di awal dan mengembalikan sebuah *file descriptor*. *File descriptor* tersebut nanti digunakan oleh server maupun klien untuk bertukar data.

Sekarang beralih ke sisi klien. Sama dengan server, *socket* juga harus dibuat dengan menggunakan primitif SOCKET, tetapi tidak memerlukan BIND untuk menyematkan alamat. Kemudian primitif CONNECT mencoba untuk membuat koneksi dengan server. Setelah koneksi berhasil dibuat, server dan klien sudah bisa menggunakan SEND dan RECV untuk mengirim dan menerima data. Setelah komunikasi selesai, *socket* ditutup dengan menggunakan primitif CLOSE.

2.2 Algoritma RSA

Algoritma RSA mendasarkan proses enkripsi dan dekripsinya pada konsep bilangan prima dan aritmetika modulo. Secara ringkas, algoritma RSA dapat dituliskan sebagai berikut [2]:

1. Pilih dua buah bilangan prima sembarang, sebut *a* dan *b*. Jaga kerahasiaan *a* dan *b* ini.
2. Hitung $n = a \times b$. Besaran *n* tidak dirahasiakan.
3. Hitung $m = (a - 1) \times (b - 1)$. Sekali *m* dihitung, *a* dan *b* dapat dihapus untuk mencegah diketahuinya oleh pihak lain.
4. Pilih sebuah bilangan bulat untuk kunci publik, sebut namanya *e*, yang relatif prima terhadap m.
5. Bagkitkan kunci dekripsi, *d*, dengan kekongruenan $ed \equiv 1 \pmod{m}$. Lakukan enkripsi terhadap isi pesan dengan persamaan $c_i = p_i^e \bmod n$, yang dalam hal ini p_i adalah blok plainteks, c_i adalah cipherteks yang diperoleh , dan *e* adalah kunci enkripsi (kunci publik). Harus dipenuhi persyaratan bahwa nilai pi harus terletak dalam himpunan nilai 0, 1, 2, ..., *n*-1 untuk menjamin hasil perhitungan tidak berada di luar himpunan.
6. Proses dekripsi dilakukan dengan menggunakan persamaan $p_i = c_i^d \bmod n$, yang dalam hal ini *d* adalah kunci dekripsi.

2.3 Pemrograman *Socket* dalam PHP

Selain untuk memrogram situs, PHP juga bisa digunakan untuk membangun aplikasi yang memanfaatkan *socket*. Saat ini *socket programming API* dalam PHP mendukung semua hal yang dibutuhkan untuk membuat komunikasi klien-server berbasis *socket* melalui TCP/IP dan dapat digunakan untuk membangun aplikasi klien-server sederhana dengan cepat [5]. Primitif-primitif untuk *socket* pun sudah disediakan. Primitif-primitif tersebut dibuat berdasarkan *socket* Berkeley [1]. Berikut ini primitif-primitif *socket* dalam PHP beserta padanannya dalam *socket* Berkeley.

Primitif *Socket* dalam PHP	Primitif *Socket* Berkeley
`socket_create`	SOCKET
`socket_bind`	BIND
`socket_listen`	LISTEN
`socket_accept`	ACCEPT
`socket_connect`	CONNECT
`socket_send`	SEND
`socket_recv`	RECV
`socket_close`	CLOSE

Program server yang dibuat hanya untuk mengirim nomor urut koneksi kemudian menerima pesan dari klien. Agar sewaktu-waktu bisa menerima permintaan koneksi, maka program server harus berjalan terus. Secara sederhana, kode program untuk server adalah:

```
 1  <?php
 2    error_reporting(E_ALL);
 3    $address = "127.0.0.1";
 4    $port = 10000;
 5
 6    $mysock =
    socket_create(AF_INET,
    SOCK_STREAM, SOL_TCP);
 7
 8    socket_bind($mysock, $address,
    $port);
 9
10    socket_listen($mysock, 5);
11
12    echo "Server started, accepting
    connections ...\n";
13
14    $i = 0;
15
16    while (true) {
17      $client =
    socket_accept($mysock);
18      $i++;
19
20      echo "\n===== Connection # "
    . $i . " =====\n";
21
22      $msg = socket_read($client,
    2048);
23      echo "\nMessage from
    client:\n" . $msg . "\n";
```

```
24
25      echo "\nSending response
    message to client ...";
26      $t = "Thanks. Your connection
    # is " . $i . ".";
27      socket_write($client, $t,
    strlen($t));
28      echo "\nMessage sent\n";
29      echo "\n===== End of
    connection # " . $i . " =====\n";
30    }
31
32    echo "\nClosing sockets...\n";
33
34    socket_close($client);
35    socket_close($mysock);
36  ?>
```

Program klien yang dibuat hanya untuk mengirim pesan satu kali saja, yakni melalui argumen program. Setelah mengirim pesan, klien akan menerima pesan tanggapan dari server sebagai tanda bahwa pesan yang dikirimkan telah diterima server. Kode program tersebut dapat dituliskan sebagai berikut:

```
 1  <?php
 2    error_reporting(E_ALL);
 3
 4    $address = "127.0.0.1";
 5    $port = 10000;
 6
 7    $socket =
    socket_create(AF_INET,
    SOCK_STREAM, SOL_TCP);
 8    if ($socket === false) {
 9      echo "socket_create() failed:
    reason: " .
    socket_strerror(socket_last_error
    ()) . "\n";
10    } else {
11      echo "Socket successfully
    created\n";
12    }
13
14    echo "\nAttempting to connect
    to " . $address . ":" . $port . "
    ...";
15    $result =
    socket_connect($socket, $address,
    $port);
16    if ($result === false) {
17      echo "\nsocket_connect()
    failed.\nReason: (" . $result .
    ") " .
    socket_strerror(socket_last_error
    ($socket)) . "\n";
18    } else {
19      echo "\nSuccessfully
    connected to " . $address . ":" .
    $port . "\n";
20    }
21
22    $msg = $argv[1];
23
24    echo "\nSending message to
    server ...\n";
25    socket_write($socket, $msg,
    strlen($msg));
26    echo "Message sent\n";
27
28    $response =
    socket_read($socket, 2048);
29    echo "\nResponse from
    server:\n" . $response . "\n";
30
31    echo "\nClosing socket ...\n";
32    socket_close($socket);
33    echo "Socket closed";
34  ?>
```

2.4 Penerapan Algoritma RSA dalam *Socket*

2.4.1 Sisi Klien

Proses enkripsi diterapkan sebelum pesan dikirimkan kepada tujuan. Untuk itu, proses ini dilakukan di program klien. Dalam proses enkripsi dibutuhkan besaran *n* dan *e* (lihat bab 1.2). Kedua besaran ini dibuat oleh pihak yang akan dikirimi pesan (dalam hal ini program server) kemudian kedua besaran tersebut disebarluaskan agar pihak yang ingin mengirim pesan (dalam hal ini program klien) dapat mengirim pesan dengan benar.

Langkah pertama untuk mengenkripsi pesan yaitu menentukan pesan yang akan dienkripsi.

```
1  <?php
2    $p_msg = $argv[1];  // plain
    message
3  ?>
```

Kemudian tentukan besaran *n* dan *e*.

```
1  <?php
2    $n = 3337;
3    $e = 79;  // encryption key
4  ?>
```

Kemudian pesan sudah siap untuk dienkripsi. Pesan dienkripsi per karakter dan berdasarkan nilai ASCII-nya. Setelah dienkripsi, karakter-karakter tersebut disusun kembali menjadi sebuah pesan dengan ditambahi sebuah karakter pemisah khusus, yakni karakter spasi.

```
 1  <?php
 2    // explode plain message to
    plain blocks, convert them to
    ASCII value (3 digits), and
    encipher them to cipher blocks
 3    $p = "";
 4    $i = 0;
 5    while ($i < strlen($p_msg)) {
 6      if (strlen(ord($p_msg[$i]))
    == 3) {
 7        $p[] = ord($p_msg[$i]);
 8      } else if
    (strlen(ord($p_msg[$i])) == 2) {
 9        $p[] = "0" .
    ord($p_msg[$i]);
```

```
10      } else if
    (strlen(ord($p_msg[$i])) == 1) {
11        $p[] = "00" .
    ord($p_msg[$i]);
12      }
13      $c[$i] = bcmod(bcpow($p[$i],
    $e), $n);
14      $i++;
15    }
16
17    // implode the cipher blocks to
    cipher message
18    $c_msg = implode($c, " ");
19  ?>
```

2.4.2 Sisi Server

Proses dekripsi digunakan untuk membaca pesan yang dikirimkan oleh klien kepada server. Untuk itu, proses ini diletakkan di program server. Program server sebagai penerima pesan merupakan pihak yang menentukan besaran-besaran yang dibutuhkan dalam proses enkripsi dan dekripsi.

Langkah pertama untuk mendekripsi pesan yaitu menentukan pesan yang akan dienkripsi.

```
1  <?php
2    $c_msg = socket_read($client,
    2048);  // cipher message
3  ?>
```

Kemudian definisikan besaran-besaran yang diperlukan.

```
1  <?php
2    $a = 47;
3    $b = 71;
4
5    $n = $a * $b;
6    $m = ($a - 1) * ($b - 1);
7
8    $e = 79;  // encryption key
9
10   $k = 1;
11   do {
12     $d = (1 + ($k * $m)) / $e;
    // decryption key
13     $k++;
14   } while (!is_int($d));
15 ?>
```

Kemudian pesan yang terenkripsi sudah siap untuk didekripsi. Sama seperti proses enkripsi, dalam proses dekripsi pesan juga harus dipecah per karakter. Pemisah antarkarakter dalam pesan terenkripsi sudah ditentukan dalam proses enkripsi, yakni karakter spasi. Dengan demikian, dalam proses dekripsi untuk memecah pesan tersebut juga berdasarkan karakter spasi. Setelah itu, setiap karakter terenkripsi tersebut didekripsi dan langsung disambungkan ke dalam sebuah pesan sehingga terbentuk pesan yang sesungguhnya dari pengirim.

```
1  <?php
2    // explode cipher message to
    cipher blocks
3    $c = explode(" ", $c_msg);
4
5    // decipher the cipher blocks
    and concat them into plain
    message
6    $p_msg = "";
7    foreach ($c as $x) {
8      $p_msg .= chr(bcmod(bcpow($x,
    $d), $n));
9    }
10 ?>
```

2.5 Uji Coba

Uji coba yang dilakukan menggunakan PHP versi 5.2.6 pada komputer bersistem operasi Windows Vista™ Home Premium Service Pack 1. Aplikasi yang dibuat merupakan aplikasi konsol. Berikut ini langkah-langkah uji coba yang dilakukan.

Jalankan terlebih dulu program server, yaitu dengan mengetikkan perintah sebagai berikut:

```
D:\Socket>php -q server.php
```

Jika program server sudah berjalan, maka akan ditampilkan pesan sebagai berikut:

```
D:\Socket>php -q server.php
Server started, accepting connections
...
```

Kemudian jalankan program klien lengkap dengan argumen pesan yang akan disampaikan. Contohnya untuk mengirimkan pesan “Algoritma RSA” dan “socket programming”:

```
D:\Socket>php client.php "Algoritma
RSA"
Socket successfully created

Attempting to connect to
127.0.0.1:10000 ...
Successfully connected to
127.0.0.1:10000

Enciphering message ...
Message successfully enciphered

----- Plain message -----
Algoritma RSA
----- End of plain message -----

----- Cipher message -----
541 1795 101 2237 2560 193 1031 1864
957 1379 274 2251 541
----- End of cipher message -----

Sending message to server ...
Message sent

Response from server:
Thanks. Your connection # is 1.
```

```
Closing socket ...
Socket closed
D:\Socket>php client.php "socket
programming"
Socket successfully created

Attempting to connect to
127.0.0.1:10000 ...
Successfully connected to
127.0.0.1:10000

Enciphering message ...
Message successfully enciphered

----- Plain message -----
socket programming
----- End of plain message -----

----- Cipher message -----
732 2237 2197 374 1113 1031 1379 165
2560 2237 101 2560 957 1864 1864 193
2668 101
----- End of cipher message -----

Sending message to server ...
Message sent

Response from server:
Thanks. Your connection # is 2.

Closing socket ...
Socket closed
```

Dari dua kali pengiriman pesan di atas, pesan asli dan pesan terenkripsi yang dikirimkan adalah:

1. Pesan asli:
   `Algoritma RSA`
   Pesan terenkripsi:
   `541 1795 101 2237 2560 193 1031 1864 957 1379 274 2251 541`
2. Pesan asli:
   `socket programming`
   Pesan terenkripsi:
   `732 2237 2197 374 1113 1031 1379 165 2560 2237 101 2560 957 1864 1864 193 2668 101`

Setelah program klien dijalankan dua kali seperti contoh di atas, maka program server akan tertampil sebagai berikut:

```
D:\Socket>php -q server.php
Server started, accepting connections
...

===== Connection # 1 =====

Message from client:

----- Cipher message -----
541 1795 101 2237 2560 193 1031 1864
957 1379 274 2251 541
----- End of cipher message -----
----- Plain message -----
Algoritma RSA
----- End of plain message -----

Sending response message to client ...
Message sent

===== End of connection # 1 =====

===== Connection # 2 =====

Message from client:

----- Cipher message -----
732 2237 2197 374 1113 1031 1379 165
2560 2237 101 2560 957 1864 1864 193
2668 101
----- End of cipher message -----

----- Plain message -----
socket programming
----- End of plain message -----

Sending response message to client ...
Message sent

===== End of connection # 2 =====
```

Dari dua kali uji coba tersebut, pesan terenkripsi yang diterima server bisa dikembalikan ke pesan aslinya sehingga maksud pesan tersampaikan.

3 PENUTUP

Dalam sistem kriptografi kunci publik, terdapat kunci yang tidak bersifat rahasia, yaitu kunci enkripsi/publik. Algoritma RSA termasuk dalam sistem kriptografi kunci publik. Kunci enkripsi/publik dalam algoritma RSA adalah *e* dan *n*, sedangkan kunci dekripsinya adalah *d* dan *n*.

Kekuatan algoritma RSA terletak pada tingkat kesulitan dalam memfaktorkan bilangan non prima menjadi faktor primanya, yang dalam hal ini $n = a \times b$ [2]. Meskipun nilai *n* bersifat publik, faktor *a* dan *b* akan disembuyikan dengan aman dari orang lain karena sangat sulitnya memfaktorkan nilai *n* [4]. Agar lebih sulit difaktorkan, bilangan *a* dan *b* seharusnya merupakan bilangan prima yang sangat besar, misalnya hingga mencapai 100 digit.

Dalam makalah ini dibahas penerapan algoritma RSA dalam pengiriman data melalui *socket* berbasis TCP/IP. Program yang diuji memang sangat sederhana, tetapi ini adalah dasar komunikasi data melalui *socket*. *Socket* sebenarnya merupakan teknik pengiriman data yang cukup mumpuni. *Socket* sering digunakan dalam aplikasi yang memerlukan komunikasi data yang tidak bersifat standar, sehingga aplikasi akan bersifat unik. Contoh aplikasi populer yang menggunakan *socket* adalah aplikasi *instant messenger* dan *proxy server*.

DAFTAR REFERENSI

[1] Group, The PHP Documentation. 2007. *PHP Manual*.

[2] Munir, Rinaldi. 2006. *Matematika Diskrit, Edisi Keempat*. Bandung: Program Studi Teknik Informatika ITB.

[3] Radhakrishnan, Mani & Solworth, Jon. 2004. *Socket Programming in C/C++*. Chicago.

[4] Rivest, R. L., Shamir, A., & Adleman, L. 1977. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*.

[5] *Socket Programming with PHP*. http://www.devshed.com/c/a/PHP/Socket-Programming-With-PHP/. Diakses pada tanggal 19 Desember 2008 pukul 19.30 WIB.

[6] Tanenbaum, Andrew S. 2003. *Computer Networks, Fourth Edition*. Upper Saddle River, NJ: Prentice Hall.