Bab 20

Aplikasi - Pengamanan Sesi

Yang dimaksud dengan sesi (*session*) disini adalah sesi komunikasi. Ditingkat dasar seperti *layer* IP, sesi komunikasi biasanya bersifat umum. Pada *layer* atas, sesi komunikasi biasanya bersifat lebih khusus, misalnya untuk berkomunikasi dengan *shell*[1]. Kita akan bahas 3 macam pengamanan sesi yaitu:

- SSL/TLS untuk pengamanan sesi komunikasi antar proses.
- SSH untuk pengamanan sesi *shell*, yaitu sesi dimana antarmuka pengguna dengan sistem menggunakan *command line interface*.
- IPsec untuk pengamanan sesi koneksi internet umum (*layer* IP).

20.1 SSL/TLS

SSL adalah singkatan dari *Secure Socket Layer*, suatu *defacto standard* yang dibuat oleh Netscape, perusahaan pembuat *web browser* terpopuler tahun 1995. Ada dua versi SSL yang digunakan secara umum, yaitu SSL2 (lihat [hic95]) dan SSL3 (lihat [fri96]). Tahun 1999, standard SSL diambil alih oleh Internet Engineering Task Force (IETF) dan namanya diubah menjadi TLS (lihat [die99]), yang merupakan singkatan dari *Transport Layer Security* (SSL3 menjadi TLS versi 1). Perubahan nama ini mungkin agar nama menjadi lebih netral karena *socket* adalah istilah Unix. Versi terbaru dari TLS adalah versi 1.2 (lihat [die08]). Karena nama SSL sudah sangat melekat, meskipun nama sudah berganti, disini kita menyebutnya sebagai SSL/TLS.

[1] Ini adalah istilah Unix. Dalam dunia Microsoft Windows, yang mirip dengan *shell* adalah *command prompt*.

SSL/TLS dimaksudkan untuk mengamankan sesi komunikasi antar proses. Dalam suatu *operating system* yang *multi-tasking* seperti Unix, Microsoft Windows, Linux atau MacOS, berbagai proses berjalan secara bersamaan. (Untuk melihat berbagai proses yang aktif, jika menggunakan Microsoft Windows, pembaca dapat menggunakan Windows Task Manager. Jika menggunakan Unix, Linux atau MacOS 10, pembaca dapat menggunakan *command line program* `ps` untuk melihat berbagai proses yang aktif.) Pengamanan komunikasi tidak terbatas pada pengamanan komunikasi antar proses dalam satu komputer, akan tetapi juga pengamanan komunikasi antar proses di dua komputer yang berbeda. Bahkan pengamanan komunikasi antar proses di dua komputer yang berbeda jauh lebih penting karena terdapat lebih banyak ancaman. Sebagai contoh, penggunaan terbesar dari SSL/TLS adalah untuk "sesi aman" antara proses *web browser* dengan proses *web server* yang biasanya berada pada dua komputer yang berbeda.

20.1.1 Standard SSL/TLS

Dalam sesi menggunakan SSL/TLS, proses yang disebut *client* berkomunikasi dengan proses yang disebut *server*. Secara garis besar, sesi antara *client* dan *server* diamankan dengan, pertama melakukan *handshake*, lalu mengenkripsi komunikasi antara *client* dan *server* selama sesi berlangsung. Tujuan dari *handshake* adalah:

- *Server authentication* (*optional*).
- Menentukan parameter enkripsi.
- *Client authentication* (*optional*).

Bagian *handshake* mungkin merupakan yang terpenting dalam sesi SSL/TLS. Yang jelas *handshake* merupakan bagian paling rumit dari segi protokol. Secara garis besar, protokol *handshake* adalah sebagai berikut:

1. *Client* mengirim ke *server*: nomor versi SSL/TLS yang digunakan *client*, parameter enkripsi, data yang dibuat secara acak, dan informasi lain yang dibutuhkan oleh *server* untuk berkomunikasi dengan *client*. Jika dibutuhkan, *client* juga meminta *server certificate*.
2. *Server* mengirim ke *client*: nomor versi SSL/TLS yang digunakan *server*, parameter enkripsi, data yang dibuat secara acak, dan informasi lain yang dibutuhkan oleh *client* untuk berkomunikasi dengan *server*. Jika diminta dalam langkah 1, *server* juga mengirim *server certificate*. Jika dibutuhkan, *server* juga meminta *client* untuk mengirim *client certificate*.
3. Jika *client* meminta *server certificate* dalam langkah 1, *client* melakukan *server authentication* (akan dijelaskan secara lebih rinci) menggunakan

server certificate dan informasi lain yang didapat. Jika *authentication* sukses, *server certificate* tidak diminta, atau pengguna mengizinkan, *client* meneruskan ke langkah 4. Jika tidak, sesi dihentikan.

4. Menggunakan data yang telah didapat, *client* membuat suatu *premaster secret* untuk sesi. Tergantung jenis enkripsi yang digunakan, ini dapat dilakukan dengan partisipasi *server*. *Premaster secret* dienkripsi menggunakan kunci publik *server* (diambil dari *server certificate*), lalu dikirim ke *server*.

5. Jika *server* meminta *client certificate* pada langkah 2, *client* menandatangan secara *digital* data yang unik untuk sesi yang diketahui oleh *client* dan *server*. Data berikut *digital signature* dan *client certificate* dikirim oleh *client* ke *server*.

6. Jika *server* meminta *client certificate* pada langkah 2, *server* melakukan *client authentication*. Jika *authentication* diminta dan gagal, maka sesi dihentikan.

7. *Client* dan *server* membuat *master secret* menggunakan *premaster secret*. *Master secret* digunakan oleh *client* dan *server* untuk membuat kunci sesi yang merupakan kunci enkripsi simetris.

8. *Client* memberi tahu *server* bahwa kunci sesi akan digunakan untuk mengenkripsi komunikasi lebih lanjut. *Client* kemudian mengirim pesan yang dienkripsi ke *server* yang mengatakan bahwa ia selesai dengan *handshake*.

9. *Server* memberi tahu *client* bahwa kunci sesi akan digunakan untuk mengenkripsi komunikasi lebih lanjut. *Server* kemudian mengirim pesan yang dienkripsi ke *client* yang mengatakan bahwa ia selesai dengan *handshake*.

10. *Handshake* selesai.

Server authentication dilakukan dengan memeriksa *server certificate*. Dalam *server certificate* terdapat informasi antara lain:

- kunci publik *server*,
- masa berlaku *certificate*,
- *domain name* untuk *server*, dan
- *domain name* untuk pembuat *certificate* (biasanya *certificate* dibuat oleh suatu *certificate authority*).

Server certificate ditanda-tangan secara *digital* oleh pembuatnya. Pemeriksaan *certificate* dilakukan dengan menjawab berbagai pertanyaan sebagai berikut:

- Apakah tanggal pemeriksaan berada dalam masa berlaku *certificate*?
- Apakah pembuat *certificate* (teridentifikasi dengan *domain name* pembuat) dapat kita percayai? Pembuat bisa merupakan suatu *certificate authority* atau sumber lain.
- Apakah *certificate* ditanda-tangan secara *digital* dengan benar oleh pembuatnya?
- Apakah *domain name* untuk *server* yang tertera dalam *certificate* sesuai dengan *domain name* untuk *server* yang sebenarnya?

Jika semua pertanyaan terjawab secara memuaskan, maka *authentication* sukses. Jika ada pertanyaan yang jawabannya tidak memuaskan, maka *authentication* gagal. *Client authentication* juga dilakukan serupa. Setelah *handshake* selesai, komunikasi antara *client* dan *server* dienkripsi menggunakan kunci sesi.

Standard SSL/TLS tentunya juga menentukan format yang harus digunakan dalam komunikasi antara *client* dan *server*. Standard juga menentukan *cipher suite* yang dapat digunakan. *Cipher suite* adalah kombinasi metode enkripsi yang terdiri dari:

- Metode enkripsi untuk *key exchange* atau *key agreement*.
- Metode enkripsi untuk *certificate*.
- Metode enkripsi menggunakan kunci sesi.
- Metode enkripsi untuk *message authentication code* (MAC).

Untuk *key exchange* atau *key agreement*, metode enkripsi yang dapat digunakan antara lain:

- RSA, dan
- Diffie-Hellman (DH).

Untuk *certificate*, metode enkripsi yang dapat digunakan antara lain:

- RSA, dan
- DSA/DSS.

Untuk enkripsi menggunakan kunci sesi, metode enkripsi yang dapat digunakan antara lain:

- RC4,
- 3DES,

- AES 128 bit, dan
- AES 256 bit.

Jika menggunakan *block cipher* seperti 3DES atau AES, maka metode enkripsi harus menggunakan mode *cipher block chaining* (CBC). Untuk *message authentication code*, metode enkripsi yang dapat digunakan antara lain:

- HMAC-MD5,
- HMAC-SHA-1,
- HMAC-SHA256.

Standard TLS versi 1.2 mengharuskan implementasi sedikitnya bisa mendukung *cipher suite* yang terdiri dari RSA untuk *key exchange* dan *certificate*, AES 128 bit mode CBC untuk enkripsi menggunakan kunci sesi, dan HMAC-SHA-1 untuk *message authentication code*. Untuk mendapatkan informasi teknis secara rinci mengenai TLS versi 1.2, silahkan membaca [die08].

20.1.2 Penggunaan SSL/TLS

Seperti dikatakan sebelumnya, penggunaan terbesar SSL/TLS adalah untuk *secure web browsing*. Semua *web browser* yang populer mendukung *secure web browsing* menggunakan SSL/TLS. Biasanya pengguna *web browser* tidak perlu mengetahui SSL/TLS. Saat *web browser* akan menampilkan *web page* dengan *prefix* `https` (jadi bukan `http`), maka *web browser* secara otomatis akan memulai sesi SSL/TLS. Dalam melakukan *handshake* SSL/TLS, *web browser* akan melakukan *server authentication* dengan memeriksa *certificate* untuk *web server*. *Web browser* biasanya sudah memiliki daftar *certificate authority* yang dapat dipercaya, dan ada *web browser* yang memperbolehkan pengguna untuk menambah pembuat *certificate* yang dipercaya kedalam daftar. Jika ada masalah dalam *authentication* maka *web browser* biasanya memberi tahu pengguna dan menanyakan pengguna apakah sesi diteruskan atau tidak. Biasanya ada juga opsi untuk menambah *certificate* yang bermasalah ke daftar pengecualian dimana *certificate* yang ada dalam daftar tidak perlu diperiksa. Jika *handshake* SSL/TLS berhasil maka *web page* dapat ditampilkan setelah terlebih dahulu di*download* dengan proteksi sesi SSL/TLS.

Selain *web browser*, perangkat lunak untuk *web server* seperti Apache juga sudah mendukung SSL/TLS. Jadi pembaca dapat membuat *web server* yang dapat melayani pengguna internet dengan *secure web page*. Apache memberi opsi untuk menggunakan *server certificate* yang dibuat sendiri atau dibuat oleh suatu *certificate authority*. Tentunya ada kemungkinan *web browser* pengguna akan berpendapat bahwa *certificate* bermasalah, akan tetapi ini dapat diatasi seperti dibahas di paragraf sebelum ini.

Karena *web browser* dan *web server* mendukung SSL/TLS, maka semua sistem informasi *client-server* yang berbasis *web* dapat menggunakan fasilitas SSL/TLS dengan mudah. Buku ini sangat merekomendasikan pendekatan *client-server* yang berbasis *web* untuk suatu sistem informasi. Penggunaan sistem informasi akan sangat fleksibel karena pengguna dapat berada dimana saja asalkan ada koneksi TCP/IP dengan *server*, contohnya:

- di komputer yang sama dengan *server*,
- di komputer lain yang terhubung dengen *server* melalui *local area network*, atau
- di lokasi lain, bahkan di negara yang berbeda waktu 12 jam dengan *server*, asalkan ada koneksi internet ke *server*.

Jika sistem informasi dibuat menggunakan sesuatu yang *platform independent* seperti Apache-MySQL-PHP, maka sistem informasi akan lebih fleksibel lagi karena *server* dapat ditempatkan di komputer dengan *operating system* apa saja, baik Microsoft Windows, Linux, MacOS 10, FreeBSD, OpenBSD, atau varian Unix lainnya, dan dapat dengan mudah dipindahkan.

Tentunya penggunaan SSL/TLS tidak harus melalui *web*. Pada prinsipnya apa saja yang memerlukan pengamanan komunikasi antar proses dapat menggunakan SSL/TLS. Ada beberapa alat yang dapat membantu pembuatan sistem yang menggunakan SSL/TLS. Contohnya adalah *cryptographic library* seperti RSA BSafe (lihat bagian 24.2) atau Cryptlib (lihat bagian 24.3). Contoh lain adalah OpenSSL yang juga memberikan fasilitas *cryptographic library* selain memberikan fasilitas SSL/TLS. Cryptlib dan OpenSSH keduanya menggunakan OpenSSL. OpenSSL adalah implementasi *open source* dari SSL/TLS yang didasarkan pada SSLeay, implementasi SSL oleh Eric Young. Meskipun *open source*, OpenSSL menggunakan *licensing* yang tidak kompatibel dengan GPL (*GNU Public License*). Untuk pembaca yang menginginkan implementasi SSL yang lebih kompatibel dengan GPL, GnuTLS adalah implementasi serupa yang menggunakan LGPL (*Lesser GNU Public License*).

20.2 SSH

Asal mula dari *secure shell* (SSH) adalah *software* yang dibuat oleh Tatu Ylönen, ssh, yang dimaksudkan sebagai program Unix untuk *secure remote shell access*, menggantikan telnet dan program *remote shell access* lainnya yang tidak aman. Protokol SSH kemudian dijadikan standard IETF.

Secara tradisional, dalam mengakses sistem jenis Unix, *shell* kerap digunakan sebagai antarmuka. Jika sistem yang diakses berada di komputer lain, maka program Unix untuk *remote shell access* seperti telnet atau rlogin perlu

digunakan. Namun penggunaan telnet atau rlogin tidak aman karena dapat disadap. Lebih parah dari itu, *password* untuk mengakses sistem dapat dicuri, misalnya menggunakan *password sniffer*. SSH berfungsi seperti telnet dan rlogin, tetapi komunikasi antara komputer pengguna dan komputer yang diakses diamankan. Pengamanan bukan hanya untuk saat yang penting seperti *login*, tetapi juga untuk komunikasi selanjutnya. SSH menggunakan konsep *client-server* dimana *client* terdapat di komputer pengguna dan *server* terdapat di komputer dimana *shell* berjalan.

Ada perbedaan pendapat antara Ylönen dan para pembuat OpenSSH mengenai status nama ssh. Ylönen berpendapat bahwa ssh adalah *trademark* yang menjadi hak miliknya, sedangkan pembuat OpenSSH berpendapat bahwa nama tersebut sudah menjadi nama generik sehingga tidak dapat digunakan sebagai *trademark*. Buku ini tidak berpihak dalam hal tersebut, tetapi sekedar mengungkapkan adanya perbedaan pendapat.

20.2.1 Standard SSH

Standard IETF untuk SSH (ada yang menyebutnya SSH-2) dipublikasikan pada tahun 2006 dalam bentuk sekumpulan RFC ([leh06], [ylo06a], [ylo06b], [ylo06c], [ylo06d], [sch06] dan [cus06]). Standard mendefinisikan arsitektur dari protokol SSH terdiri dari berbagai *layer* sebagai berikut:

- *Transport layer* (lihat [ylo06c]), *layer* terbawah untuk SSH. *Layer* ini fungsinya mirip dengan SSL/TLS. Jadi *handshake* dilakukan di *layer* ini, dan juga komunikasi pada tingkat paket, yang diamankan menggunakan enkripsi. *Authentication* yang dilakukan pada *handshake* adalah *server authentication*. *Layer* ini juga melakukan pergantian kunci sesi, misalnya setiap 1 GB data dikomunikasikan atau setiap 1 jam. *Layer* ini memberikan layanan ke *layer* atas melalui antarmuka yang telah ditetapkan.

- *User authentication layer* (lihat [ylo06b]), *layer* untuk *login* ke *server*. SSH memberikan fleksibilitas kepada *server* dan *client* mengenai cara *user authentication*. *Server* dapat menawarkan beberapa alternatif untuk cara *user authentication*. *Client* kemudian mencoba satu per satu satu diantaranya dengan urutan yang sembarang. Berikut adalah beberapa cara yang dapat digunakan untuk *user authentication*:

  - *Public key infrastructure*. Hanya cara ini yang harus didukung suatu implementasi untuk *user authentication*. Dengan cara ini, *server* mengetahui kunci publik dari suatu pasangan kunci. Pengguna harus memiliki kunci privat dari pasangan kunci, yang digunakan untuk membuat *digital signature* sesuatu yang dikirim *server*. *Server* mengecek *digital signature* tersebut untuk menentukan apakah benar

pengguna memiliki kunci privat dari pasangan kunci. Algoritma *digital signature* yang dapat digunakan antara lain RSA dan DSA/DSS.

- *Password*. Ini adalah cara yang sederhana untuk *user authentication*. *Layer* ini juga memberi fasilitas untuk pergantian *password*.
- *Host-based*. Dengan cara ini siapapun dapat mengakses *shell* asalkan *client* berada pada komputer tertentu berdasarkan *domain name* atau nomor IP. *Authentication* dilakukan dengan mengecek *certificate* untuk komputer dimana *client* berada.

Ada mekanisme untuk menambah cara *user authentication* tanpa merubah standard yaitu *generic message exchange authentication* [cus06].

- *Connection layer* (lihat [ylo06d]). Di *layer* ini konsep *global request*, *channel*, sesi interaktif, dan TCP/IP *port forwarding* didefinisikan. *Global request* adalah permintaan yang dapat merubah *state* di sisi *server* independen dari semua *channel*. Setiap sesi dan *forwarded port connection* merupakan *channel*. Jenis *channel* yang standard termasuk:
  - *shell*,
  - *direct* TCP/IP, untuk *forwarded connection* yang tidak diminta, jadi *port forwarding* atas inisiatif sisi pengirim sendiri (pengirim bisa *client* bisa *server*), dan
  - *forwarded* TCP/IP, untuk *forwarded connection* yang diminta oleh penerima (penerima bisa *client* bisa *server*).

  Satu *connection* dapat terdiri dari beberapa *channel*.

Untuk informasi teknis yang rinci mengenai arsitektur protokol SSH, silahkan membaca [ylo06a].

Kadang, sewaktu melakukan *server authentication*, *client* tidak mengenal kunci publik *server*. Jika demikian, *fingerprint* dari kunci publik dapat diperlihatkan ke pengguna. Jika pengguna cukup yakin bahwa kunci publik memang milik *server* yang ia ingin akses, maka sesi dapat diteruskan. Ada metode alternatif untuk mengecek *fingerprint* yang ditawarkan oleh standard SSH dengan menggunakan fasilitas DNSSEC (lihat [sch06]). *Fingerprint* dari *server* dapat dipublikasikan secara aman sebagai bagian dari DNS *record*.

20.2.2 Penggunaan SSH

Penggunaan SSH secara tradisional memang untuk mengakses *shell* di komputer lain (*server*). Biasanya *server* menggunakan *operating system* jenis Unix. Beberapa *operating system* jenis Unix yang populer saat ini termasuk:

- Linux,

- Mac OS 10,
- Solaris,
- FreeBSD, dan
- OpenBSD.

Dua implementasi SSH yang cukup dikenal adalah ssh (suatu produk komersial) dan OpenSSH (*open source*). Untuk OpenSSH, *client* dapat juga ditempatkan di komputer dengan *operating system* Microsoft Windows.

Penggunaan lain SSH adalah sebagai jalur aman bagi suatu program *client-server*. Istilah penggunaan ini adalah *tunneling*. Dua program yang termasuk dalam paket OpenSSH:

- SFTP dan
- SCP,

menggunakan SSH sebagai *tunnel*. SFTP berfungsi seperti program ftp, meskipun fiturnya diluar pengamanan jauh lebih sedikit dibandingkan ftp. SCP adalah program untuk *secure remote copy*. Menggunakan SCP *files* bisa di*copy* antar komputer secara aman, biasanya antara komputer *client* dan komputer *server*. Beberapa aplikasi dari pihak ketiga juga menggunakan OpenSSH untuk *tunneling*.

Sejak versi 4.3, OpenSSH dapat digunakan untuk membuat suatu *virtual private network* berdasarkan OSI *layer* 2/3 TUN. Suatu *virtual private network* (VPN) adalah jaringan komputer didalam jaringan yang lebih besar dimana koneksi antar komputer dalam VPN diamankan sehingga VPN seolah membuat jaringan sendiri. Menggunakan OpenSSH, VPN *tunnel* antara dua komputer dapat dibuat dimana di setiap komputer ujungnya adalah suatu *virtual device*, sebut saja `tun0`. Tentunya `tun0` harus dikonfigurasi untuk memiliki IP *address*, contohnya menggunakan `ipconfig` untuk Microsoft Windows atau `ifconfig` untuk Linux atau jenis Unix lainnya. Meskipun OpenSSH dapat digunakan untuk membuat VPN, karena *overhead* yang tinggi, solusi ini tidak se-efisien solusi menggunakan IPsec (lihat bagian 20.3). Kecuali untuk penggunaan darurat atau sementara, penggunaan OpenSSH untuk implementasi VPN tidak direkomendasikan. Perlu ditekankan disini bahwa VPN *tunneling* yang dilakukan oleh OpenSSH tidak ada dalam standard SSH.

20.3 IPsec

IPsec adalah protokol pengamanan komunikasi pada *layer* IP (Internet Protocol). Secara garis besar, IPsec mengamankan komunikasi dengan memberikan fasilitas *authentication* dan enkripsi untuk setiap paket IP. Dalam protokol

IPsec ditentukan juga cara untuk *mutual authentication* di permulaan sesi dan cara untuk negosiasi kunci sesi.

20.3.1 Standard IPsec

Ada 3 tempat dimana standard IPsec dapat dimplementasi:

- terintegrasi dalam IP *stack*,
- diluar komputer (*bump in the wire*), atau
- dibawah IP *stack* diatas *device driver* (*bump in the stack*).

IP *stack* untuk berbagai *operating system* sudah mengintegrasikan *standard* IPsec. Solusi kedua secara tradisional merupakan solusi militer menggunakan perangkat keras khusus. Solusi ketiga bisa digunakan jika IP *stack* belum mengintegrasikan *standard* IPsec.

Dari segi arsitektur protokol (lihat [ken05a]), IPSec terdiri dari 4 komponen:

- protokol untuk *authentication header* (AH),
- protokol untuk *encapsulating security payload* (ESP),
- konsep *security association*, terdiri dari beberapa parameter pengamanan yang menentukan bagaimana suatu sambungan satu arah menggunakan AH atau ESP diamankan, dan
- protokol untuk manajemen *security association*, termasuk *key management*, secara otomatis menggunakan *internet key exchange* (IKE atau IKEv2).

Authentication header (AH) memberi jaminan integritas data, *authentication* terhadap asal dari paket IP, dan pencegahan *replay attack*. AH mengamankan integritas IP *payload* dan setiap *field* dalam *header* kecuali jika *field* adalah *mutable field* (*field* yang dapat berubah dalam perjalanan paket, contohnya TTL). Integritas IP *payload* diamankan menggunakan *integrity check value* (ICV). *Authentication* juga dijamin oleh ICV secara tidak langsung dengan penggunaan kunci rahasia bersama dalam kalkulasi ICV. Ada dua mode penggunaan AH:

- *transport mode* untuk mengamankan *layer* atas termasuk TCP, dan
- *tunnel mode* untuk mengamankan "*inner*" IP.

Tabel 20.1 memperlihatkan format paket AH.

- *Next header* adalah *field* sebesar 8 bit yang mengidentifikasi jenis *payload*. Isinya merupakan IP *protocol number* sesuai dengan yang telah ditentukan oleh *Internet Assigned Numbers Authority* (IANA). Sebagai contoh, *protocol number* 4 mengidentifikasi IPv4, sedangkan 6 mengidentifikasi TCP.

Bit 0-7	Bit 8-15	Bit 16-23	Bit 24-31
Next header	Payload length	Reserved	
Security parameters index (SPI)			
Sequence number field			
Integrity check value - ICV (variable)			

Tabel 20.1: Format paket AH

- *Payload length* adalah *field* sebesar 8 bit yang memberi tahu besarnya AH. Nilai *field* ini ditambah 2 lalu dikalikan dengan 32 bit merupakan besarnya AH.

- *Reserved* adalah *field* sebesar 16 bit untuk penggunaan masa depan. Pengirim paket harus mengisi *field* ini dengan 0. Penerima paket menggunakan nilai *field* ini dalam komputasi ICV, dan setelah itu dapat mengabaikannya.

- SPI adalah *field* sebesar 32 bit yang digunakan oleh penerima paket untuk menentukan *security association* yang berlaku.

- *Sequence number* adalah field sebesar 32 bit yang digunakan untuk mencegah *replay attack*. Untuk setiap *security association*, paket pertama yang dikirim mempunyai *sequence number* 1, paket kedua mempunyai *sequence number* 2 dan seterusnya. Penerima mencegah *replay attack* menggunakan *sliding window*. AH dengan *sequence number* diluar *sliding window* otomatis ditolak. Duplikat AH juga ditolak. Jika AH dengan *sequence number* sama dengan nilai terkecil *sliding window* sudah diterima, maka *sliding window* dapat digeser. Besar *sliding window* minimal 32, direkomendasikan 64, tetapi implementasi dapat menggunakan *sliding window* yang lebih besar.

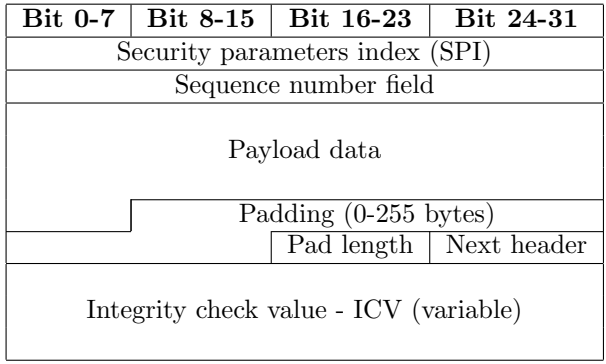
- ICV adalah *field* yang besarnya adalah kelipatan dari 32 bit. Pengirim paket mengkalkulasi ICV dan menempatkannya di *field* ini. Penerima paket juga mengkalkulasi ICV dan membandingkannya dengan *field* ini. Jika cocok maka penerima menganggap integritas paket tidak terganggu.

Untuk mendapatkan informasi teknis yang lebih rinci mengenai *authentication header*, termasuk *transport mode* dan *tunnel mode*, penggunaan SPI untuk menentukan *security association*, kalkulasi ICV, dan *packet fragmentation* dan *reassembly*, silahkan membaca [ken05b].

Encapsulating security payload (ESP) memberi fasilitas pengamanan kerahasiaan data, *authentication* terhadap asal data, dan integritas. ESP dapat digunakan sendiri, bersama dengan AH, atau secara berlapis. Pengamanan kerahasiaan data dilakukan menggunakan enkripsi, sedangkan integritas dan *authentication* terhadap asal data dilakukan menggunakan ICV seperti halnya dengan AH. Juga seperti halnya dengan AH, ada dua mode penggunaan ESP:

- *transport mode* untuk mengamankan *layer* atas termasuk TCP, dan
- *tunnel mode* untuk mengamankan "*inner*" IP.

Tabel 20.2 memperlihatkan format paket ESP.



Tabel 20.2: Format paket ESP

- *Payload data* adalah data yang bisa dienkripsi bisa tidak. Jika dienkripsi maka *payload data* termasuk data sinkronisasi enkripsi seperti *initialization vector*.
- *Padding* diperlukan untuk dua macam keperluan yaitu agar besar naskah yang dienkripsi merupakan kelipatan dari ukuran blok dan agar ruang antara ahir dari naskah acak dan permulaan dari *pad length* terisi. Oleh sebab itu bisa jadi sebagian dari *padding* dienkripsi dan sebagian tidak dienkripsi.
- *Pad length* adalah *field* yang memberi tahu besarnya *padding* dalam byte.

Security parameter index, *sequence number*, *next header* dan *integrity check value*, isinya dan fungsinya sama seperti dalam AH. Meskipun enkripsi dan integritas (termasuk *authentication*) dapat digunakan secara independen, ESP biasanya digunakan untuk keduanya. Jadi 3 kombinasi enkripsi dan integritas yang dapat digunakan adalah:

- *confidentiality only* (BOLEH didukung implementasi),
- *integrity only* (HARUS didukung implementasi), dan
- *confidentiality and integrity* (HARUS didukung implementasi).

Untuk mendapatkan informasi teknis yang lebih rinci mengenai *encapsulating security payload*, termasuk *transport mode* dan *tunnel mode*, penggunaan SPI untuk menentukan *security association*, enkripsi, *padding*, kalkulasi ICV, dan *packet fragmentation* dan *reassembly*, silahkan membaca [ken05c].

Security association (SA) terdiri dari beberapa parameter yang menentukan bagaimana komunikasi paket satu arah menggunakan AH atau ESP diamankan. Jadi untuk komunikasi dua arah diperlukan sepasang SA, satu untuk setiap arah. SA yang digunakan ditentukan oleh SPI dan jenis pengamanan (AH atau ESP). Komputer yang terlibat dalam komunikasi AH atau ESP menyimpan setiap SA dalam suatu *database*. Parameter yang ada dalam SA termasuk:

- *security parameter index* (SPI),
- *counter* untuk *sequence number* (untuk paket keluar),
- *sequence counter overflow* yaitu *flag* yang mengindikasikan terjadinya *overflow* pada *counter*,
- *anti-replay window* (untuk paket yang diterima),
- algoritma dan kunci untuk *authentication* (untuk AH dan ESP),
- algoritma, kunci dan IV untuk enkripsi (untuk ESP),
- *lifetime* dari SA, bisa berupa jangka waktu atau jumlah byte,
- *mode*: *transport* atau *tunnel*,
- kebijakan untuk IP *filtering*,
- dan lainnya.

Setiap *device* TCP/IP yang menggunakan IPsec mempunyai *security policy database* (SPD) dan *security association database* (SAD). Suatu *security policy* dalam SPD adalah kebijakan bagaimana komunikasi TCP/IP lewat *device* tersebut harus diproses, termasuk apakah paket tertentu harus diproses menggunakan IPsec. Jadi *security policy* bersifat lebih umum, sedangkan *security association* bersifat lebih khusus.

Manajemen *security association* antar komputer dilakukan menggunakan protokol *internet key exchange* (versi kini IKEv2), terutama untuk *key management*. Komunikasi menggunakan IKEv2 selalu terdiri dari pasangan *request*

dan *response*. Pasangan *request* dan *response* disebut *exchange*. Komunikasi menggunakan IKEv2 selalu dimulai dengan *initial exchanges*, antara *initiator* dan *responder*, terdiri dari dua *exchange* yaitu IKE-SA-INIT dan IKE-AUTH. IKE-SA-INIT fungsinya adalah negosiasi algoritma kriptografi dan melakukan Diffie-Hellman *key agreement*. Untuk IKE-SA-INIT, pesan pertama adalah dari *initiator* ke *responder* sebagai berikut:

$$\begin{array}{ll} \textit{Initiator} & \textit{Responder} \\ HDR, SA_{i1}, KE_i, N_i & \rightarrow \end{array}$$

dimana HDR adalah *header* IKE terdiri dari SPI, berbagai nomor versi dan berbagai *flag*, SA_{i1} mendeklarasikan berbagai alternatif algoritma kriptografi yang didukung oleh *initiator*, KE_i adalah nilai Diffie-Hellman dari *initiator*, dan N_i adalah *nonce*[2] dari *initiator*. Pesan kedua dalam IKE-SA-INIT adalah dari *responder* ke *initiator* sebagai berikut:

$$\begin{array}{lll} \textit{Initiator} & & \textit{Responder} \\ & \leftarrow & HDR, SA_{r1}, KE_r, N_r[, CERTREQ] \end{array}$$

dimana HDR adalah *header* IKE, SA_{r1} menyatakan pilihan *cryptographic suite* berdasarkan apa yang ditawarkan SA_{i1}, KE_r adalah nilai Diffie-Hellman dari *responder*, dan N_r adalah *nonce* dari *responder*. *Responder* dapat meminta *certificate* dari *initiator* menggunakan CERTREQ (*optional*). Setelah pesan ini, kedua pihak (*initiator* dan *responder*) dapat menyepakati suatu *seed* yang dinamakan SKEYSEED untuk membuat berbagai kunci (*seed* disepakati menggunakan Diffie-Hellman *key agreement*), termasuk SK_a yaitu kunci untuk *authentication* dan SK_e yaitu kunci untuk enkripsi. Setiap pesan setelah IKE-SA-INIT diamankan menggunakan SK_a dan SK_e (kecuali *field* HDR). *Exchange* kedua dalam IKEv2 adalah IKE-AUTH, yang fungsinya adalah *authentication* pesan-pesan dalam IKE-SA-INIT dan negosiasi CHILD-SA. Pesan pertama dalam IKE-AUTH adalah dari *initiator* ke *responder* sebagai berikut:

$$\begin{array}{ll} \textit{Initiator} & \textit{Responder} \\ HDR, ID_i, [CERT,][CERTREQ,][ID_r,] & \rightarrow \\ AUTH, SA_{i2}, TS_i, TS_r & \end{array}$$

dimana pesan (kecuali *field* HDR) diamankan menggunakan SK_a dan SK_e. ID_i adalah identitas dari *initiator*. Jika *certificate* diminta oleh *responder* maka $CERT$ berisi *certificate*. *Initiator* juga dapat meminta *certificate responder* menggunakan $CERTREQ$ (*optional*). ID_r bersifat *optional* dan berisi pilihan *initiator* dari identitas *responder*. $AUTH$ adalah data *authentication* untuk pesan pertama dalam IKE-SA-INIT. SA_{i2}, TS_i dan TS_r digunakan untuk negosiasi CHILD-SA, dimana SA_{i2} adalah SA yang ditawarkan, TS_i dan

[2] *Nonce* adalah suatu nilai acak yang setiap kali dibuat, diharapkan berbeda dari *nonce* lainnya. Fungsi dari *nonce* adalah untuk mencegah *replay attack*.

TS_r adalah *traffic selectors* yang ditawarkan. Pesan kedua dalam IKE-AUTH adalah dari *responder* ke *initiator* sebagai berikut:

$$\begin{array}{lll} \textit{Initiator} & & \textit{Responder} \\ & \leftarrow & HDR, ID_r, [CERT,] \\ & & AUTH, SA_{r2}, TS_i, TS_r. \end{array}$$

Seperti halnya dengan pesan pertama, pesan kecuali *field* HDR diamankan menggunakan SK_a dan SK_e. ID_r adalah identitas *responder*. Jika *certificate* diminta oleh *initiator*, maka $CERT$ berisi *certificate*. $AUTH$ adalah data *authentication* untuk pesan kedua dalam IKE-SA-INIT, SA_{r2} adalah SA yang disetujui *responder* untuk CHILD-SA dan TS_i dan TS_r adalah *traffic selectors* yang disetujui. Setelah IKE-AUTH, CHILD-SA dapat dibuat atau CHILD-SA yang sudah ada dapat diganti kuncinya menggunakan *exchange* CREATE-CHILD-SA. Kedua pihak dapat memulai *exchange* ini, jadi *initiator* untuk CREATE-CHILD-SA tidak harus *initiator* untuk IKE-SA-INIT dan IKE-AUTH. Pesan pertama dalam CREATE-CHILD-SA adalah sebagai berikut:

$$\begin{array}{lll} \textit{Initiator} & & \textit{Responder} \\ HDR, [N,]SA, Ni[, KE_i][, TS_i, TS_r] & \rightarrow & \end{array}$$

Jika CREATE-CHILD-SA sedang digunakan untuk mengganti kunci, maka N mengidentifikasi SA yang hendak diganti kuncinya. SA adalah tawaran berbagai parameter untuk SA, dan N_i adalah *nonce*. KE_i (*optional*) adalah nilai Diffie-Hellman yang digunakan untuk memperkuat kunci SA. Jika SA menawarkan beberapa grup untuk Diffie-Hellman, maka KE_i merupakan elemen dari grup yang seharusnya dapat disetujui *responder* (jika tidak disetujui, maka *exchange* gagal dan harus diulang dengan nilai KE_i yang lain). Jika CREATE-CHILD-SA sedang membuat CHILD-SA yang baru (jadi bukan mengganti kunci), maka TS_i dan TS_r berisi *traffic selectors* yang ditawarkan. Jawaban untuk pesan pertama adalah pesan berikut:

$$\begin{array}{lll} \textit{Initiator} & & \textit{Responder} \\ & \leftarrow & HDR, SA, N_r, [KE_r,] \\ & & [TS_i, TS_r] \end{array}$$

dimana SA adalah berbagai parameter SA yang dipilih oleh *responder* dari yang ditawarkan *initiator*, dan N_r adalah *nonce*. Jika pesan pertama memuat KE_i dan berasal dari grup yang disetujui dalam SA, maka KE_r merupakan nilai Diffie-Hellman dari *responder*. Jika KE_i berasal dari grup yang tidak ada dalam SA, maka *exchange* harus ditolak, dan *initiator* harus mengulang *exchange* menggunakan KE_i yang berada dalam grup yang disetujui SA. Jika CREATE-CHILD-SA bukan sedang digunakan untuk mengganti kunci, maka TS_i dan TS_r adalah *traffic selectors* yang dipilih oleh *responder* dari yang

ditawarkan oleh *initiator*. Selain *exchange* jenis IKE-SA-INIT, IKE-AUTH dan CREATE-CHILD-SA, ada satu lagi jenis *exchange* dalam IKEv2 yaitu INFORMATIONAL. *Exchange* INFORMATIONAL digunakan untuk pesan *control* dan dapat berisi notifikasi, penghapusan dan konfigurasi. Untuk mendapatkan informasi yang rinci mengenai *exchange* INFORMATIONAL dan informasi lainnya mengenai IKEv2, silahkan membaca [kau05].

20.3.2 Penggunaan IPsec

Karena IPsec melakukan pengamanan komunikasi pada *level* IP, IPsec banyak digunakan untuk implementasi *virtual private network* (VPN), yang memang secara umum beroperasi di *level* IP. Ini jauh lebih efisien dibandingkan implementasi menggunakan OpenSSH karena IPsec lebih terintegrasi dengan IP dibandingkan OpenSSH.

Ada sedikit komplikasi jika IPsec digunakan melewati NAT (*network address translation*), yang digunakan oleh berbagai perangkat seperti:

- *consumer broadband modem/router*,
- *firewall*,

dimana *address* IP internet di *interface* eksternal perangkat diterjemahkan menjadi *address* IP privat yang digunakan jaringan internal seperti 192.168.1.5. (Situasinya sebetulnya lebih rumit lagi karena biasanya terdapat lebih dari satu *address* IP privat di jaringan internal, kadang paket dari luar harus diteruskan ke salah satu dari sekian *address* IP privat yang ada berdasarkan *port address*.) Protokol AH tidak kompatibel dengan NAT karena *address* IP dalam paket diganti, sedangkan *address* IP merupakan *field* yang diamankan AH dan bukan merupakan *mutable field*. Beruntung untuk ESP, *address* IP bukan merupakan *field* yang diamankan, jadi ESP dapat digunakan bersamaan dengan NAT. Mungkin itulah sebabnya ESP juga mendukung *authentication* disamping enkripsi, karena kita tidak dapat menggunakan AH untuk *authentication* yang melewati NAT. Akan tetapi meskipun pergantian *address* IP bukan merupakan masalah bagi ESP, persoalan *mapping* antara *address* IP eksternal dengan *address* IP privat perlu dipecahkan. Salah satu solusi untuk ini adalah menggunakan NAT *traversal* dalam IKEv2 dan UDP *encapsulation* untuk ESP. Untuk informasi yang lebih rinci mengenai hal ini, silahkan membaca [kau05].

20.4 Ringkasan

Di bab ini telah dibahas tiga standard pengamanan sesi yaitu SSL/TLS, SSH dan IPsec. SSL/TLS dimaksudkan untuk pengamanan komunikasi antar proses, dan banyak digunakan untuk *secure web pages*. SSH dimaksudkan untuk

pengamanan penggunaan *remote shell*. Satu implementasi SSH yaitu OpenSSH mendukung penggunaannya sebagai *virtual private network* (VPN), namun penggunaan ini tidak efisien dan hanya direkomendasikan untuk penggunaan darurat atau sementara. IPsec adalah standard pengamanan pada *level* IP, oleh sebab itu penggunaan IPsec dalam implementasi VPN sangat direkomendasikan, meskipun penggunaannya dengan *network address translation* (NAT) sedikit rumit.