Bab 24

Aplikasi - Cryptographic Library

Cryptographic library sangat membantu untuk membuat kriptografi terintegrasi dengan aplikasi. Cara integrasi bisa *embedded* (di*compile* bersama aplikasi) atau melalui *module* yang dapat di*load* oleh aplikasi, contohnya

- *dynamically linked library* (DLL) untuk Microsoft Windows, atau
- `jar` untuk Java.

Penggunaan *cryptographic library* oleh aplikasi biasanya dilakukan melalui *application program interface* (API). Yang diberikan oleh *cryptographic library* antara lain:

- berbagai algoritma kriptografi, contohnya AES, RSA, dan DSA,
- kemampuan *public key infrastructure* (PKI), dan
- *secure sessions*, contohnya SSL/TLS.

Di bab ini kita akan bahas 3 *cryptographic library* yang populer yaitu

- OpenSSL,
- RSA BSafe, dan
- Cryptlib.

OpenSSL merupakan produk *open source*, RSA BSafe merupakan produk komersial (tetapi ada versi yang disumbangkan oleh RSA sebagai *open source*), dan Cryptlib adalah *open source* dengan dua macam lisensi (*dual license*) yaitu GPL dan komersial.

24.1 OpenSSL

OpenSSL merupakan produk *open source* dengan lisensi yang berbeda dan tidak kompatibel dengan GPL. OpenSSL dapat di*download* dari *website*

`http://www.openssl.org`.

OpenSSL terdiri dari 3 komponen utama yaitu:

- SSL/TLS *library*,
- Crypto *library*, dan
- `openssl` *command line tool*.

SSL/TLS *library* memberikan fasilitas untuk melakukan sesi SSL/TLS dalam bentuk *loadable module* (DLL untuk Microsoft Windows). Fungsi API untuk SSL/TLS *library* dibagi menjadi 5 golongan:

- *Protocol methods*, untuk menentukan versi SSL/TLS dan jenis layanan (*client*, *server*, atau *client-server*).
- *Ciphers*, untuk menentukan berbagai jenis enkripsi yang dapat digunakan.
- *Protocol contexts*, untuk menentukan *context* secara global selama program berjalan.
- *Sessions*, untuk menentukan berbagai parameter per sesi.
- *Connections*, untuk koneksi. Ini adalah fungsi utama SSL/TLS *library*, dan selama program berjalan fungsi dari golongan ini yang banyak digunakan.

Crypto library adalah implementasi dari berbagai algoritma enkripsi dan *hashing*. Selain digunakan oleh SSL/TLS *library* dan openssl *command line tool*, *crypto library* juga digunakan oleh OpenSSH, GnuPG dan implementasi standard kriptografi lainnya.

Command line tool `openssl` adalah program yang berorientasi Unix yang dapat digunakan untuk:

- Pembuatan dan manajemen kunci privat/publik.
- Operasi kriptografi *public key* termasuk *key management*, *digital signing* dan *digital signature checking*.
- Pembuatan dan manajemen *certificate* X.509.
- Kalkulasi *digest*.

- Enkripsi dan dekripsi.
- Testing SSL/TLS *client* dan *server*.
- S/MIME.
- Membuat *time-stamp request*, membuat *time-stamp* dan melakukan verifikasi *time-stamp*.

24.2 RSA BSafe

RSA BSafe adalah produk komersial dari RSA Security Inc., suatu divisi dari EMC Corporation. Selain versi komersial, RSA Security juga membuat versi *open source* untuk *platform* Microsoft Windows, Solaris dan Linux:

- RSA BSafe Share for C++, dan
- RSA BSafe Share for Java.

Perbedaan utama versi komersial dan versi *share* adalah versi komersial termasuk sertifikasi FIPS-140, mendukung PKCS#11 (perangkat kripto) dan mendukung *platform* HP-UX, AIX, *mainframe* dan *embedded* disamping mendukung Microsoft Windows, Solaris dan Linux. Versi *share* tidak termasuk sertifikasi, tidak mendukung PKCS#11, dan hanya mendukung *platform* Microsoft Windows, Solaris dan Linux. Operasi yang didukung oleh RSA BSafe Share antara lain:

- Operasi kriptografi, termasuk enkripsi/dekripsi, pembuatan *message digests*, pembuatan *message authentication codes*, pembuatan dan verifikasi *digital signatures* dan pembuatan *pseudo-random numbers*.
- Operasi kunci, termasuk pembuatan pasangan kunci privat dan kunci publik, melakukan Diffie-Hellman *key agreement*, dan *encoding/decoding asymmetric keys*.
- Operasi *certificate*, termasuk pembuatan *self-signed certificates*, pembuatan *certificate requests*, verifikasi *certificate chains*, pembuatan *certificate revocation list*, melakukan operasi protokol OCSP, dan melakukan operasi *certificate stores*.
- Operasi PKCS#7 (*cryptographic message syntax*).
- Operasi SSL/TLS.

RSA BSafe Share for C++ bersifat *toolkit* yang menyediakan:

- *header files*,

- *object file libraries*, dan
- *code samples*,

untuk digunakan dengan Microsoft Visual Studio C/C++ (khusus untuk RSA BSafe Share for C++ versi Microsoft Windows). RSA BSafe for Java juga memberikan fasilitas serupa untuk Java dengan 2 macam API:

- implementasi standard Java JCE API dan
- implementasi standard Java JSSE API.

RSA BSafe Share dapat di*download* lengkap dengan dokumentasi di:

`https://community.emc.com/`.

24.3 Cryptlib

Cryptlib adalah suatu sistem yang dikembangkan oleh Peter Gutmann berdasarkan disertasinya mengenai *cryptographic security architecture*. Disertasi tersebut diselesaikannya tahun 2000, dan telah direvisi dan dijadikan buku yang diterbitkan tahun 2004 (lihat [gut04]). Disertasi asli juga dapat di*download* dari *website* University of Auckland.

Selain mempunyai *security architecture* yang dirancang menurut prinsip keamanan tingkat tinggi, Cryptlib juga mempunyai *software architecture* yang sangat baik. Alhasil penggunaannya relatif mudah, lebih mudah dibandingkan OpenSSL maupun RSA BSafe. Oleh sebab itu, buku ini sangat merekomendasikan penggunaan Cryptlib dan akan membahas Cryptlib secara lebih rinci dibandingkan OpenSSL dan RSA BSafe.

Cryptlib diprogram dalam C/C++, namun, menggunakan *language binding*, dapat diakses dari program yang ditulis dalam:

- C/C++,
- C#/.NET,
- Delphi,
- Java,
- Python,
- Tcl, atau
- Visual Basic.

Khusus untuk Python atau Tcl, *language binding* harus dibuat menggunakan fasilitas *extension* dari sistem Python atau Tcl. Sebagai contoh, menggunakan platform Microsoft Windows, Cryptlib dapat di*build* menggunakan Microsoft Visual Studio 2008, dan menggunakan ActiveState ActivePython 2.6, membuat extension dengan melakukan

```
python setup.py install
```

dari *command prompt*, setelah terlebih dahulu melakukan

```
cd cl333/bindings
```

dimana `cl333` adalah *main folder* untuk Cryptlib. Yang perlu diperhatikan adalah versi *compiler* C/C++ yang digunakan untuk *build* Python harus sama dengan versi yang digunakan untuk *build* Cryptlib (ActiveState ActivePython 2.6 untuk Microsoft Windows di*build* menggunakan Microsoft Visual Studio 2008). Program yang menggunakan Cryptlib harus melakukan inisialisasi sebelum memanggil fungsi Cryptlib lainnya dan harus melakukan finalisasi setelah selesai dengan Cryptlib. Program C/C++ yang menggunakan Cryptlib mempunyai format sebagai berikut:

```
#include "cryptlib.h"

cryptInit();

/* Isi program yang menggunakan Cryptlib */

cryptEnd();
```

Untuk Python, format adalah sebagai berikut:

```
from cryptlib_py import *

cryptInit()

# Isi program yang menggunakan Cryptlib

cryptEnd()
```

Cryptlib diimplementasi dengan cara yang sangat *object-oriented*. API untuk Cryptlib memberikan fasilitas untuk menggunakan Cryptlib pada 3 tingkatan antarmuka:

- *high-level interface*, yang memanipulasi *container objects* berupa *sessions*, *envelopes*, dan *certificates*,
- *mid-level interface*, yang memanipulasi *action objects* berupa *encryption contexts* dan *container objects* berupa *keysets*, dan
- *low-level interface* untuk kustomisasi layanan pendukung.

Pada tingkat *high-level interface*, Cryptlib memberikan layanan untuk *security services* yang lengkap. Untuk *secure enveloping*, antarmuka ini memberikan fasilitas untuk:

- *secure* CMS *enveloping*,
- *secure* S/MIME *enveloping*, dan
- *secure* PGP/OpenPGP *enveloping*.

Untuk *secure session*, *high-level interface* memberikan layanan yang mudah digunakan, baik sebagai *client* maupun sebagai *server*, untuk jenis sesi:

- SSL/TLS, dan
- SSH.

Antarmuka untuk tingkat ini juga memberikan layanan yang mudah digunakan untuk CA *services*, termasuk sebagai *client* atau *server* untuk berbagai protokol berikut:

- CMP,
- SCEP,
- OCSP, dan
- RTCS.

Cryptlib bahkan dapat digunakan sebagai *plug-and-play* PKI. Selain untuk CA *services*, antarmuka tingkat ini juga mendukung protokol untuk *time-stamps* yaitu TSP. Mayoritas pengguna akan menggunakan *high-level interface*. Untuk yang tidak terlalu paham kriptografi secara rinci disarankan untuk hanya menggunakan *high-level interface*. Sebagai contoh penggunaan *high-level interface*, berikut adalah *code snippet* untuk melakukan S/MIME *encrypted enveloping* menggunakan kunci yang terdapat dalam suatu X.509 *certificate*:

```
CRYPT_ENVELOPE cryptEnvelope;
int bytesCopied;
```

```
cryptCreateEnvelope(&cryptEnvelope, cryptUser,
  CRYPT_FORMAT_SMIME);

/* Tambahkan certificate ke envelope */
cryptSetAttribute(cryptEnvelope, CRYPT_ENVINFO_PUBLICKEY,
  certificate);

/* Tambahkan informasi mengenai besarnya data ke envelope */
cryptSetAttribute(cryptEnvelope, CRYPT_ENVINFO_DATASIZE,
  messageLength);

/* Masukkan data ke envelope, lakukan proses (enkripsi),
   lalu keluarkan data yang telah diproses. */
cryptPushData(cryptEnvelope, message, messageLength,
  &bytesCopied);
cryptFlushData(cryptEnvelope);
cryptPopData(cryptEnvelope, envelopedData,
  envelopedDataBufferSize, &bytesCopied);

cryptDestroyEnvelope(cryptEnvelope);
```

Sedikit penjelasan mengenai *code snippet*:

- Data dimasukkan kedalam *envelope* menggunakan `cryptPushData`. Hasil untuk parameter `&bytesCopied` biasanya sama dengan `messageLength`, tetapi ada kalanya beda (misalnya tidak semua data dapat masuk karena sudah penuh).
- `cryptFlushData` digunakan untuk *processing*. Cryptlib mengetahui apa yang harus dikerjakan dalam *processing* berdasarkan nilai berbagai atribut pada *envelope*.
- `cryptPopData` digunakan untuk mengeluarkan data hasil *processing* dari *envelope*.

Sebagai contoh penggunaan *high-level interface* untuk *secure session*, berikut adalah *code snippet* untuk memulai sesi SSL/TLS untuk *client*:

```
CRYPT_SESSION cryptSession;

cryptCreateSession(&cryptSession, cryptUser,
  CRYPT_SESSION_SSL);
```

```
cryptSetAttributeString(cryptSession,
  CRYPT_SESSINFO_SERVER_NAME, serverName, serverNameLength);
cryptSetAttribute(cryptSession, CRYPT_SESSINFO_ACTIVE,1);
```

Berikut adalah *code snippet* yang menggunakan kemampuan *plug-and-play* PKI dari *high-level interface*:

```
CRYPT_SESSION cryptSession;

/* Buat sesi CMP dan tambahkan name/address server */
cryptCreateSession(&cryptSession, cryptUser,
  CRYPT_SESSION_CMP);
cryptSetAttributeString(cryptSession, CRYPT_SESSINFO_SERVER,
  server, serverLength);

/* Tambahkan username, password dan smartcard */
cryptSetAttributeString(cryptSession, CRYPT_SESSINFO_USERNAME,
  userName, userNameLength);
cryptSetAttributeString(cryptSession, CRYPT_SESSINFO_PASSWORD,
  password, passwordLength);
cryptSetAttribute(cryptSession, CRYPT_SESSINFO_CMP_PRIVKEYSET,
  cryptDevice);

/* Aktivasikan sesi */
cryptSetAttribute(cryptSession, CRYPT_SESSINFO_ACTIVE, TRUE);
```

Yang dilakukan oleh *code snippet* diatas adalah:

- Menggunakan *smart card* untuk membuat kunci untuk *signing* dan kunci untuk enkripsi (dua pasang kunci).
- Meminta *certificate* untuk kunci *signing* dari CA.
- Menggunakan *certificate* kunci *signing* untuk meminta *certificate* kunci enkripsi dan *certificate* lainnya dari CA.
- Menyimpan semua kunci dan *certificate* yang dihasilkan dalam *smartcard*.

Code snippet diatas menunjukkan bahwa pada tingkat *high-level interface* hanya dibutuhkan beberapa instruksi untuk melakukan banyak tugas.

Pada tingkat *mid-level interface*, pengguna dapat melakukan operasi agak lebih rinci seperti:

- *key generation*,
- *key management*,
- operasi enkripsi dan *digest*,
- *key exchange*, dan
- operasi *digital signature*.

Semua operasi pada tingkat *mid-level interface* melibatkan *encryption context*. Operasi *key management* juga dapat melibatkan *container object* jenis *keyset*. Berikut adalah *code snippet* yang memberi contoh *key generation* pasangan kunci RSA 2048 bit ke *encryption context*, dilanjutkan dengan penyimpanan kunci privat ke *keyset* berupa *file* dengan format PKCS#15:

```
CRYPT_CONTEXT privKeyContext;
CRYPT_KEYSET cryptKeyset;

cryptCreateContext(&privKeyContext, cryptUser, CRYPT_ALGO_RSA);
cryptSetAttributeString(privKeyContext, CRYPT_CTXINFO_LABEL,
  label, labelLength);
cryptSetAttribute(privKeyContext, CRYPT_CTXINFO_KEYSIZE,
  2048/8);

cryptGenerateKey(privKeyContext);

cryptKeysetOpen(&cyptKeyset, cryptUser, CRYPT_KEYSET_FILE,
  "/home/kelsey/keys.p15", CRYPT_KEYOPT_NONE);
cryptAddPrivateKey(cryptKeyset, privKeyContext, password);
```

Dalam *code snippet* diatas, `label` digunakan karena diperlukan saat *retrieval* dari *keyset* untuk identifikasi. Jenis *keyset* adalah *file*, yang diindikasikan menggunakan konstan `CRYPT_KEYSET_FILE`. Untuk skala besar, penggunaan *database* (RDBMS atau RDBMS dengan ODBC) disarankan, terutama untuk keperluan CA. Berikut adalah berbagai jenis *keyset* dalam Cryptlib:

Jenis Keyset	Penjelasan
CRYPT_KEYSET_FILE	*File* PKCS#15 atau PGP *ring*.
CRYPT_KEYSET_HTTP	URL untuk lokasi *certificate*/CRL.
CRYPT_KEYSET_LDAP	Direktori LDAP.
CRYPT_KEYSET_DATABASE	RDBMS.
CRYPT_KEYSET_ODBC	ODBC RDBMS.
CRYPT_KEYSET_PLUGIN	RDBMS lewat *database network plugin interface*.
CRYPT_KEYSET_DATABASE_STORE	RDBMS untuk CA.
CRYPT_KEYSET_ODBC_STORE	ODBC RDBMS untuk CA.
CRYPT_KEYSET_PLUGIN_STORE	RDBMS lewat *database network plugin interface* untuk CA.

Jika pada tingkat *high-level interface* enkripsi dilakukan pada data dalam *container object*, pada tingkat *mid-level interface* enkripsi dilakukan langsung pada buffer *in place*, contohnya seperti dalam *code snippet* berikut:

```
cryptEncrypt(cryptContext, buffer, length);
```

Untuk *key exchange* kunci simetris, `cryptExportKey` dan `cryptImportKey` digunakan. Jika kunci simetris dibuat oleh satu pihak, pembuat kunci simetris melakukan `cryptExportKey` dan penerima melakukan `cryptImportKey`. *Code snippet* berikut adalah contoh untuk pembuat kunci simetris, dimana kunci simetris dienkripsi menggunakan kunci publik penerima (hasilnya berada dalam *buffer* dengan *pointer* `encryptedKey`):

```
CRYPT_CONTEXT pubKeyContext, cryptContext;
void *encryptedKey;
int encryptedKeyLength;

cryptCreateContext(&cryptContext, cryptUser, CRYPT_ALGO_AES);
```

```
cryptGenerateKey(cryptContext);

encryptedKey = malloc(encryptedKeyMaxLength);

cryptExportKey(encryptedKey, encryptedKeyMaxLength,
  &encryptedKeyLength, pubKeyContext, cryptContext);
```

Digital signing pada tingkat *mid-level interface* dilakukan menggunakan fungsi `cryptCreateSignature` sedangkan fungsi `cryptCheckSignature` digunakan untuk verifikasi. Namun fungsi `cryptCreateSignature` hanya melakukan bagian enkripsi, jadi *hashing* harus dilakukan terlebih dahulu. Berikut adalah *code snippet* untuk *digital signing*:

```
CRYPT_CONTEXT sigKeyContext, hashContext;
void *signature;
int signatureLength;

cryptCreateContext(&hashContext, cryptUser, CRYPT_ALGO_SHA);

cryptEncrypt(hashContext, data, dataLength);
cryptEncrypt(hashContext, data, 0);

signature = malloc(signatureMaxLength);

cryptCreateSignature(signature, signatureMaxLength,
  &signatureLength, sigKeyContext, hashContext);
cryptDestroyContext(hashContext);
```

Dan berikut adalah *code snippet* untuk verifikasi *digital signature*, dimana jika verifikasi gagal, akan menghasilkan *error* `CRYPT_ERROR_SIGNATURE`:

```
CRYPT_CONTEXT sigCheckContext, hashContext;

cryptCreateContext(&hashContext, cryptUser, CRYPT_ALGO_SHA);

cryptEncrypt(hashContext, data, dataLength);
cryptEncrypt(hashContext, data, 0);

cryptCheckSignature(signature, signatureLength,
  sigCheckContext, hashContext);
```

```
cryptDestroyContext(hashContext);
```

Pada tingkat *low-level interface*, berbagai implementasi algoritma enkripsi dan *authentication* dapat digunakan, misalnya untuk implementasi protokol kriptografi yang tidak standard. Namun untuk itu pengguna harus menguasai berbagai konsep dan algoritma kriptografi secara rinci. Tingkat *low-level interface* juga memberikan fasilitas untuk:

- kustomisasi *database plugin*,
- kustomisasi *network plugin*, atau
- kustomisasi *crypto plugin*.

Untuk dapat bertransaksi dengan berbagai *database* yang digunakan sebagai *certificate store*, Cryptlib memberi fasilitas *database plugin interface* yang dapat dikustomisasi sesuai dengan jenis *database* yang digunakan. Ada 5 fungsi yang harus dibuat untuk suatu *database plugin* yaitu:

- `openDatabase`, untuk memulai sesi dengan *database*,
- `closeDatabase`, untuk mengahiri sesi dengan *database*,
- `performUpdate`, untuk mengirim data ke *database* menggunakan instruksi SQL,
- `performQuery`, untuk mendapatkan data dari *database* menggunakan instruksi SQL, dan
- `performErrorQuery`, untuk mendapatkan informasi mengenai *error* yang terjadi dari *database*.

Untuk dapat menggunakan berbagai protokol komunikasi, Cryptlib memberi fasilitas *network plugin interface* yang dapat dikustomisasi sesuai dengan protokol. Ada 5 fungsi yang harus dibuat untuk suatu *network plugin* yaitu:

- `transportOKFunction`, untuk mengecek status dari *transport layer*,
- `transportConnectFunction`, untuk memulai koneksi dengan *server* atau *remote client*,
- `transportDisconnectFunction`, untuk mengahiri koneksi dengan *server* atau *remote client*,
- `transportReadFunction`, untuk mendapatkan data dari *server* atau *remote client*, dan

- `transportWriteFunction`, untuk mengirim data ke *server* atau *remote client*.

Untuk dapat menambahkan atau mengganti berbagai kapabilitas enkripsi dengan implementasi baru, Cryptlib memberi fasilitas *crypto plugin interface*. Berbeda dengan *database* dan *network plugin interface*, *crypto plugin interface* memberi akses ke antarmuka kapabilitas enkripsi internal dari Cryptlib. Implementasi baru dapat menjadi bagian dari program, atau dapat berupa implementasi eksternal misalnya menggunakan perangkat kripto. Jika implementasi baru menggantikan implementasi Cryptlib, implementasi lama harus di*unplug* dan implementasi baru di*plugin*.

Cryptlib dan dokumentasi dapat di*download* dari

`http://www.cs.auckland.ac.nz/~pgut001/cryptlib/`

atau

`http://www.cryptlib.com`.

24.4 Ringkasan

Di bab ini telah dibahas 3 *cryptographic library* yang populer yaitu OpenSSL, RSA Bsafe dan Cryptlib. OpenSSL bersifat *open source*, RSA BSafe bersifat komersial meskipun ada versi *open source*, sedangkan Cryptlib adalah *open source* dengan pilihan lisensi GPL atau komersial. Karena Cryptlib merupakan *library* yang terlengkap dan mudah digunakan, buku ini merekomendasikan Cryptlib dan membahasnya lebih rinci dibandingkan OpenSSL dan RSA BSafe.