# [openssl证书制作及编程](http://blog.csdn.net/sealyao/article/details/5774744)

标签： [编程](http://www.csdn.net/tag/%e7%bc%96%e7%a8%8b)[ssl](http://www.csdn.net/tag/ssl)[server](http://www.csdn.net/tag/server)[callback](http://www.csdn.net/tag/callback)[file](http://www.csdn.net/tag/file)[socket](http://www.csdn.net/tag/socket)

2010-07-29 19:56 4769人阅读 [评论](http://blog.csdn.net/sealyao/article/details/5774744#comments)(4) [收藏](http://blog.csdn.net/sealyao/article/details/javascript:void(0);) [举报](http://blog.csdn.net/sealyao/article/details/5774744#report)

IMG_256 分类：

网络协议（9） IMG_257 linux 编程（18） IMG_258

版权声明：本文为博主原创文章，未经博主允许不得转载。

一、openssl证书制作：

1、创建目录./demoCA/ ./demoCA/newcerts/，创建文件 ./demoCA/index.txt ./demoCA/serial。

2、执行echo 01 > ./demoCA/serial

3、制作自己的CA证书

$openssl req -new -x509 -keyout ca.key -out ca.crt

4、生成服务端的私钥(key文件)及csr文件

$openssl genrsa -des3 -out server.key 1024

$openssl req -new -key server.key -out server.csr

5、生成客户端的私钥(key文件)及csr文件

$openssl genrsa -des3 -out client.key 1024

$openssl req -new -key client.key -out client.csr

6、用生成的CA的证书为刚才生成的server.csr,client.csr文件签名

$openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key

$openssl ca -in client.csr -out client.crt -cert ca.crt -keyfile ca.key

7、生成pem格式证书

有时需要用到pem格式的证书，可以用以下方式合并证书文件（crt）和私钥文件（key）来生成

$cat client.crt client.key> client.pem

$cat server.crt server.key > server.pem

二、openssl编程

      openssl的编程有两种风格：原生态的和层层封装的。

2.1原生态openssl编程

      需要使用BSD socket和openssl中的较底层的API。原生态openssl有两个比较重要的结构SSL和SSL\_CTX。

      SSL\_CTX[**数据结构**](http://lib.csdn.net/base/datastructure)是用与建立SSL/TSL连接的结构，可用于指定通讯协议（SSLv23/TSLv1）、证书、密钥等相关信息。

      SSL数据结构是和具体会话相关联的结构。

常用函数：

      SSL\_library\_init（）：初始化一个openssl库。

SSL\_CTX\_new（const SSL\_METHOD \*method）：创建一个CTX结构。

SSL\_new（SSL\_CTX \*ctx）：创建一个SSL结构。

SSL\_set\_fd（SSL \*ssl, int fd）：把ssl结构和具体会话相关联。

SSL\_accept(SSL \*ssl)、SSL\_connect(SSL \*ssl)：用于SSL会话握手。

SSL\_read（SSL \*ssl, void \*buf, int num）和SSL\_write（SSL \*ssl, void \*buf, int num）用于SSL读写。

SSL\_shutdown（SSL \*ssl）用于关闭一个SSL/TSL会话。

使用这些函数就可以进行最基本的openssl编程了。Openssl编程框架如下。

代码示例：HTTPS服务器和客户端

服务器端:

**[cpp]** [view plain](http://blog.csdn.net/sealyao/article/details/5774744) [copy](http://blog.csdn.net/sealyao/article/details/5774744)

1. #include <sys/types.h>
2. #include <sys/socket.h>
3. #include <sys/un.h>
4. #include <unistd.h>
5. #include <stdio.h>
6. #include <arpa/inet.h>
8. #include <openssl/bio.h>
9. #include <openssl/ssl.h>
10. #include <openssl/err.h>
12. #define SERVER\_PEM "../digitCert/server.pem"
13. #define SERVER\_KRY "../digitCert/server.key"
15. **int** password\_callback(**char** \*buf, **int** size, **int** rwflag, **void** \*userdata)
16. {
17. /\* For the purposes of this demonstration, the password is "ibmdw" \*/
18. printf("\*\*\* Callback function called/n");
19. strcpy(buf,"123456");
20. **return** strlen(buf);
21. }
23. **int** main()
24. {
25. **int** serv\_sock,cli\_sock;
26. socklen\_t client\_len;
27. **struct** sockaddr\_in server\_address;
28. **struct** sockaddr\_in client\_address;
29. SSL\_CTX \* ctx;
30. SSL \*ssl;
32. **int** (\*callback)(**char** \*, **int**, **int**, **void** \*) = &password\_callback;
34. printf("Serving it up in a secure manner/n/n");
35. SSL\_library\_init();
36. SSL\_load\_error\_strings();
37. ERR\_load\_BIO\_strings();
38. ERR\_load\_SSL\_strings();
39. OpenSSL\_add\_all\_algorithms();
41. printf("Attempting to create SSL context... ");
42. ctx = SSL\_CTX\_new(SSLv23\_server\_method());
43. **if**(ctx == NULL)
44. {
45. printf("Failed. Aborting./n");
46. **return** 0;
47. }
49. printf("/nLoading certificates.../n");
50. SSL\_CTX\_set\_default\_passwd\_cb(ctx, callback);
51. **if**(!SSL\_CTX\_use\_certificate\_file(ctx, SERVER\_PEM, SSL\_FILETYPE\_PEM))
52. {
53. ERR\_print\_errors\_fp(stdout);
54. SSL\_CTX\_free(ctx);
55. **return** 0;
56. }
57. **else** printf("load server.csr successful!/n");
58. **if**((SSL\_CTX\_use\_PrivateKey\_file(ctx, SERVER\_KRY, SSL\_FILETYPE\_PEM))<=0)
59. {
60. printf("use private key failed!/n/n");
61. ERR\_print\_errors\_fp(stdout);
62. SSL\_CTX\_free(ctx);
63. **return** 0;
64. }
66. serv\_sock = socket(AF\_INET,SOCK\_STREAM,0);
67. **if**(-1 == serv\_sock){
68. perror("socket");
69. }
70. server\_address.sin\_family = AF\_INET;
71. server\_address.sin\_port = htons(443);
72. server\_address.sin\_addr.s\_addr = htonl(INADDR\_ANY);
74. **int** ret = bind(serv\_sock,(**struct** sockaddr\*)&server\_address,**sizeof**(**struct** sockaddr));
75. **if**(-1 == ret){
76. perror("bind");
77. }
78. listen(serv\_sock,5);
79. **while**(1){
80. cli\_sock = accept(serv\_sock,(**struct** sockaddr \*)&client\_address,(socklen\_t \*)&client\_len);
81. ssl = SSL\_new(ctx);
82. SSL\_set\_fd (ssl, cli\_sock);
84. SSL\_accept (ssl);
86. //读取
87. **char** line[4096];
88. SSL\_read(ssl, line, **sizeof** (line));
89. //正常处理HTTP协议
90. //写入，返回报文。
91. SSL\_write(ssl,"HTTP/1.0 200 OK/r/n/r/n",19);
92. SSL\_write(ssl,"<html><header></header><body><p1>Hello World<p1></body></html>",62);
93. close(cli\_sock);
94. SSL\_shutdown(ssl);
95. SSL\_free(ssl);
96. }
97. SSL\_CTX\_free (ctx);
98. }

客户端：

**[cpp]** [view plain](http://blog.csdn.net/sealyao/article/details/5774744) [copy](http://blog.csdn.net/sealyao/article/details/5774744)

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <sys/types.h>
5. #include <sys/socket.h>
6. #include <errno.h>
7. #include <unistd.h>
8. #include <netinet/in.h>
9. #include <limits.h>
10. #include <netdb.h>
11. #include <arpa/inet.h>
12. #include <ctype.h>
13. #include <openssl/crypto.h>
14. #include <openssl/ssl.h>
15. #include <openssl/err.h>
16. #include <openssl/rand.h>
18. #define DEBUG 1
20. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
21. 功能：搜索字符串右边起的第一个匹配字符
22. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
23. **char** \*Rstrchr(**char** \*s, **char** x)
24. {
25. **int** i = strlen(s);
26. **if** (!(\*s))
27. **return** 0;
28. **while** (s[i - 1])
29. **if** (strchr(s + (i - 1), x))
30. **return** (s + (i - 1));
31. **else**
32. i--;
33. **return** 0;
34. }
36. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
37. 功能：从字符串src中分析出网站地址和端口，并得到用户要下载的文件
38. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
39. **void** GetHost(**char** \*src, **char** \*web, **char** \*file, **int** \*port)
40. {
41. **char** \*pA;
42. **char** \*pB;
43. memset(web, 0, **sizeof**(web));
44. memset(file, 0, **sizeof**(file));
45. \*port = 0;
46. **if** (!(\*src))
47. **return**;
48. pA = src;
49. **if** (!strncmp(pA, "http://", strlen("http://")))
50. pA = src + strlen("http://");
51. **else** **if** (!strncmp(pA, "https://", strlen("https://")))
52. pA = src + strlen("https://");
53. pB = strchr(pA, '/');
54. **if** (pB) {
55. memcpy(web, pA, strlen(pA) - strlen(pB));
56. **if** (pB + 1) {
57. memcpy(file, pB + 1, strlen(pB) - 1);
58. file[strlen(pB) - 1] = 0;
59. }
60. } **else**
61. memcpy(web, pA, strlen(pA));
62. **if** (pB)
63. web[strlen(pA) - strlen(pB)] = 0;
64. **else**
65. web[strlen(pA)] = 0;
66. pA = strchr(web, ':');
67. **if** (pA)
68. \*port = atoi(pA + 1);
69. **else**
70. \*port = 443;
71. }
73. /\*\*\*\*\*\*\*\*\*\*\*\*关于本文档\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
74. \*filename: https-client.c
75. \*purpose: 演示HTTPS客户端编程方法
76. \*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
77. Linux爱好者 Linux知识传播者 SOHO族 开发者 最擅长C语言
78. \*date time:2007-01-30 20:06
79. \*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
80. \* 但请遵循GPL
81. \*Thanks to:Google
82. \*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
83. \* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
84. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
86. **int** main(**int** argc, **char** \*argv[])
87. {
88. **int** sockfd, ret;
89. **char** buffer[1024];
90. **struct** sockaddr\_in server\_addr;
91. **struct** hostent \*host;
92. **int** portnumber, nbytes;
93. **char** host\_addr[256];
94. **char** host\_file[1024];
95. **char** local\_file[256];
96. **FILE** \*fp;
97. **char** request[1024];
98. **int** send, totalsend;
99. **int** i;
100. **char** \*pt;
101. SSL \*ssl;
102. SSL\_CTX \*ctx;
104. **if** (argc != 2) {
105. **if** (DEBUG)
106. fprintf(stderr, "Usage:%s webpage-address/a/n", argv[0]);
107. exit(1);
108. }
109. **if** (DEBUG)
110. printf("parameter.1 is: %s/n", argv[1]);
112. GetHost(argv[1], host\_addr, host\_file, &portnumber);        /\*分析网址、端口、文件名等 \*/
113. **if** (DEBUG)
114. printf("webhost:%s/n", host\_addr);
115. **if** (DEBUG)
116. printf("hostfile:%s/n", host\_file);
117. **if** (DEBUG)
118. printf("portnumber:%d/n/n", portnumber);
120. **if** ((host = gethostbyname(host\_addr)) == NULL) {        /\*取得主机IP地址 \*/
121. **if** (DEBUG)
122. fprintf(stderr, "Gethostname error, %s/n", strerror(errno));
123. exit(1);
124. }
126. /\* 客户程序开始建立 sockfd描述符 \*/
127. **if** ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {        /\*建立SOCKET连接 \*/
128. **if** (DEBUG)
129. fprintf(stderr, "Socket Error:%s/a/n", strerror(errno));
130. exit(1);
131. }
133. /\* 客户程序填充服务端的资料 \*/
134. bzero(&server\_addr, **sizeof**(server\_addr));
135. server\_addr.sin\_family = AF\_INET;
136. server\_addr.sin\_port = htons(portnumber);
137. server\_addr.sin\_addr = \*((**struct** in\_addr \*) host->h\_addr);
139. /\* 客户程序发起连接请求 \*/
140. **if** (connect(sockfd, (**struct** sockaddr \*) (&server\_addr), **sizeof**(**struct** sockaddr)) == -1) {        /\*连接网站 \*/
141. **if** (DEBUG)
142. fprintf(stderr, "Connect Error:%s/a/n", strerror(errno));
143. exit(1);
144. }
146. /\* SSL初始化 \*/
147. SSL\_library\_init();
148. SSL\_load\_error\_strings();
149. ctx = SSL\_CTX\_new(SSLv23\_client\_method());
150. **if** (ctx == NULL) {
151. ERR\_print\_errors\_fp(stderr);
152. exit(1);
153. }
155. ssl = SSL\_new(ctx);
156. **if** (ssl == NULL) {
157. ERR\_print\_errors\_fp(stderr);
158. exit(1);
159. }
161. /\* 把socket和SSL关联 \*/
162. ret = SSL\_set\_fd(ssl, sockfd);
163. **if** (ret == 0) {
164. ERR\_print\_errors\_fp(stderr);
165. exit(1);
166. }
168. RAND\_poll();
169. **while** (RAND\_status() == 0) {
170. unsigned **short** rand\_ret = rand() % 65536;
171. RAND\_seed(&rand\_ret, **sizeof**(rand\_ret));
172. }
174. ret = SSL\_connect(ssl);
175. **if** (ret != 1) {
176. ERR\_print\_errors\_fp(stderr);
177. exit(1);
178. }
180. sprintf(request, "GET /%s HTTP/1.1/r/nAccept: \*/\*/r/nAccept-Language: zh-cn/r/n/
181. User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)/r/n/
182. Host: %s:%d/r/nConnection: Close/r/n/r/n", host\_file, host\_addr,
183. portnumber);
184. if (DEBUG)
185. printf("%s", request);        /\*准备request，将要发送给主机 \*/
187. /\*取得真实的文件名 \*/
188. **if** (host\_file && \*host\_file)
189. pt = Rstrchr(host\_file, '/');
190. **else**
191. pt = 0;
193. memset(local\_file, 0, **sizeof**(local\_file));
194. **if** (pt && \*pt) {
195. **if** ((pt + 1) && \*(pt + 1))
196. strcpy(local\_file, pt + 1);
197. **else**
198. memcpy(local\_file, host\_file, strlen(host\_file) - 1);
199. } **else** **if** (host\_file && \*host\_file)
200. strcpy(local\_file, host\_file);
201. **else**
202. strcpy(local\_file, "index.html");
203. **if** (DEBUG)
204. printf("local filename to write:%s/n/n", local\_file);
206. /\*发送https请求request \*/
207. send = 0;
208. totalsend = 0;
209. nbytes = strlen(request);
210. **while** (totalsend < nbytes) {
211. send = SSL\_write(ssl, request + totalsend, nbytes - totalsend);
212. **if** (send == -1) {
213. **if** (DEBUG)
214. ERR\_print\_errors\_fp(stderr);
215. exit(0);
216. }
217. totalsend += send;
218. **if** (DEBUG)
219. printf("%d bytes send OK!/n", totalsend);
220. }
222. fp = fopen("index.html", "a");
223. **if** (!fp) {
224. **if** (DEBUG)
225. printf("create file error! %s/n", strerror(errno));
226. **return** 0;
227. }
228. **if** (DEBUG)
229. printf("/nThe following is the response header:/n");
230. i = 0;
231. /\* 连接成功了，接收https响应，response \*/
232. **while** ((nbytes = SSL\_read(ssl, buffer, 1)) == 1) {
233. **if** (i < 4) {
234. **if** (buffer[0] == '/r' || buffer[0] == '/n')
235. i++;
236. **else**
237. i = 0;
238. **if** (DEBUG)
239. printf("%c", buffer[0]);        /\*把https头信息打印在屏幕上 \*/
240. } **else** {
241. fwrite(buffer, 1, 1, fp);        /\*将https主体信息写入文件 \*/
242. i++;
243. **if** (i % 1024 == 0)
244. fflush(fp);        /\*每1K时存盘一次 \*/
245. }
246. }
247. fclose(fp);
248. /\* 结束通讯 \*/
249. ret = SSL\_shutdown(ssl);
250. **if** (ret != 1) {
251. ERR\_print\_errors\_fp(stderr);
252. exit(1);
253. }
254. close(sockfd);
255. SSL\_free(ssl);
256. SSL\_CTX\_free(ctx);
257. ERR\_free\_strings();
258. exit(0);
259. }

2.2层层封装的openssl编程。

     这里BSD socket的基本操作，对上层是不可见的。而且openssl自己有对自己的底层操作进行了封装。

     除此之外还需要了解BIO的概念。

     BIO是openssl 对于io 类型的抽象封装，包括：内存、文件、日志、标准输入输出、socket、加/解密、摘要和ssl 通道等。

     BIO分为两种：source/sink类型的BIO是数据源或输入数据源（我不知道sink该怎么翻译），例如，sokect BIO和文件BIO。而filter BIO就是把数据从一个BIO转换到另外一个BIO或应用接口，在转换过程中，这些数据可以不修改（如信息摘要BIO），也可以进行转换。

     BIO链，可以预先设定好多个BIO之间的链接关系，当数据从source类型的BIO进来之后，数据自动再BIO链之间传递。这应该是BIO有吸引力的地方。比如三个BIO，mbio（内存类型），sbio（ssl类型），abio（accept类型），连成BIO链：mbio->sbio->abio。

     那么对mbio的读操作实际上就包含了从accept接收数据、对加密数据ssl解密，最终把数据读到内存中的操作；对mbio的写操作实际上包含了对数据进行ssl加密，并把加密数据从socket发出去的操作。

BIO常用函数：

     BIO \* BIO\_new(BIO\_METHOD \*type)：生成BIO

     BIO\_set(BIO \*a,BIO\_METHOD \*type)：设置BIO

     BIO\_free(BIO \*a)：删除BIO

     BIO\_free\_all(BIO \*a)：删除BIO链

     BIO\_read(BIO \*b, void \*buf, int len)

     BIO\_gets(BIO \*b,char \*buf, int size)

     BIO\_write(BIO \*b, const void \*buf, int len)

     BIO\_puts(BIO \*b,const char \*buf)

     BIO \* BIO\_push(BIO \*b,BIO \*append)：该函数把参数中名为append的BIO附加到名为b的BIO上，并返回b。

     BIO \* BIO\_pop(BIO \*b)：为b的BIO从一个BIO链中移除并返回下一个BIO。

  代码示例（echo）：

服务器端：

**[cpp]** [view plain](http://blog.csdn.net/sealyao/article/details/5774744) [copy](http://blog.csdn.net/sealyao/article/details/5774744)

1. #include <stdio.h>
2. #include <string.h>
4. #include <openssl/bio.h>
5. #include <openssl/ssl.h>
6. #include <openssl/err.h>
7. #include <unistd.h>
9. #define SERVER\_PEM "../digitCert/server.pem"
10. #define SERVER\_KRY "../digitCert/server.key"
12. **int** password\_callback(**char** \*buf, **int** size, **int** rwflag, **void** \*userdata)
13. {
14. /\* For the purposes of this demonstration, the password is "ibmdw" \*/
16. printf("\*\*\* Callback function called/n");
17. strcpy(buf,"123456");
18. **return** strlen(buf);
19. }
21. **int** main()
22. {
23. SSL\_CTX \*ctx;
24. SSL \*ssl;
25. BIO \*sslbio, \*acptbio, \*out;
26. pid\_t pid;
27. **int** len;
28. **char** buf[1024];
30. **int** (\*callback)(**char** \*, **int**, **int**, **void** \*) = &password\_callback;
32. printf("Secure Programming with the OpenSSL API, Part 4:/n");
33. printf("Serving it up in a secure manner/n/n");
35. SSL\_library\_init();
36. SSL\_load\_error\_strings();
37. ERR\_load\_BIO\_strings();
38. ERR\_load\_SSL\_strings();
39. OpenSSL\_add\_all\_algorithms();
41. printf("Attempting to create SSL context... ");
42. ctx = SSL\_CTX\_new(SSLv23\_server\_method());
43. **if**(ctx == NULL)
44. {
45. printf("Failed. Aborting./n");
46. **return** 0;
47. }
49. printf("/nLoading certificates.../n");
50. SSL\_CTX\_set\_default\_passwd\_cb(ctx, callback);
51. **if**(!SSL\_CTX\_use\_certificate\_file(ctx, SERVER\_PEM, SSL\_FILETYPE\_PEM))
52. {
53. ERR\_print\_errors\_fp(stdout);
54. SSL\_CTX\_free(ctx);
55. **return** 0;
56. }
57. **else** printf("load server.csr successful!/n");
58. **if**((SSL\_CTX\_use\_PrivateKey\_file(ctx, SERVER\_KRY, SSL\_FILETYPE\_PEM))<=0)
59. {
60. printf("use private key failed!/n/n");
61. ERR\_print\_errors\_fp(stdout);
62. SSL\_CTX\_free(ctx);
63. **return** 0;
64. }
66. printf("Attempting to create BIO object... ");
67. sslbio = BIO\_new\_ssl(ctx, 0);//0 indicate using server mode
68. **if**(sslbio == NULL)
69. {
70. printf("Failed. Aborting./n");
71. ERR\_print\_errors\_fp(stdout);
72. SSL\_CTX\_free(ctx);
73. **return** 0;
74. }
76. printf("/nAttempting to set up BIO for SSL.../n");
77. BIO\_get\_ssl(sslbio, &ssl);
79. printf("Waiting for incoming connection.../n");
80. /\*Begin to listen the port\*/
81. **if**(BIO\_do\_accept(acptbio) <= 0)
82. {
83. ERR\_print\_errors\_fp(stdout);
84. SSL\_CTX\_free(ctx);
85. BIO\_free\_all(sslbio);
86. BIO\_free\_all(acptbio);
87. **return** 1;
88. }
89. **while**(1)
90. {
91. /\*Waiting for a new connection to establish\*/
92. **if**(BIO\_do\_accept(acptbio) <= 0)
93. {
94. ERR\_print\_errors\_fp(stdout);
95. SSL\_CTX\_free(ctx);
96. BIO\_free\_all(sslbio);
97. BIO\_free\_all(acptbio);
98. **return** 1;
99. }
100. out = BIO\_pop(acptbio);
102. **if**((pid=fork()))//parent process
103. {
104. BIO\_free(out);
105. }
106. **else**
107. {
108. **if**(BIO\_do\_handshake(out) <= 0)
109. {
110. printf("Handshake failed./n");
111. ERR\_print\_errors\_fp(stdout);
112. SSL\_CTX\_free(ctx);
113. BIO\_free\_all(sslbio);
114. BIO\_free\_all(acptbio);
115. **return** 1;
116. }
117. **for**(;;)
118. {
119. memset(buf,0,1023);
120. len = BIO\_read(out,buf,1023);
121. **switch**(SSL\_get\_error(ssl,len))
122. {
123. **case** SSL\_ERROR\_NONE:
124. **break**;
125. **default**:
126. printf("Read Problem!/n");
127. exit(0);
128. }
129. **if**(!strcmp(buf,"/r/n")||!strcmp(buf,"/n"))
130. **break**;
131. **if**(buf[0]=='q')
132. **break**;
133. BIO\_write(out,buf,len);
134. printf("%s/n",buf);
135. }
136. BIO\_free(out);
137. BIO\_ssl\_shutdown(sslbio);
138. exit(0);
139. }//end else
140. }
141. BIO\_ssl\_shutdown(sslbio);
142. BIO\_free\_all(sslbio);
143. BIO\_free\_all(acptbio);
144. SSL\_CTX\_free(ctx);
145. }

客户端:

**[cpp]** [view plain](http://blog.csdn.net/sealyao/article/details/5774744) [copy](http://blog.csdn.net/sealyao/article/details/5774744)

1. #include <openssl/ssl.h>
2. #include <errno.h>
3. #include <openssl/bio.h>
4. #include <openssl/err.h>
6. #include <stdio.h>
7. #include <string.h>
9. **int** main()
10. {
11. BIO \*sslbio;
12. SSL \* ssl;
13. SSL\_CTX \* ctx;
15. **int** p;

18. /\* Set up the library \*/
19. SSL\_library\_init();//一定要有，初始化ssl库
20. ERR\_load\_BIO\_strings();
21. SSL\_load\_error\_strings();
22. OpenSSL\_add\_all\_algorithms();
24. /\* Set up the SSL context \*/
26. ctx = SSL\_CTX\_new(SSLv23\_method());
28. /\* Load the trust store \*/
30. **if**(!SSL\_CTX\_load\_verify\_locations(ctx, "../digitCert/ca.crt",0))//读取CA根证书，用这个证书来验证对方的证书是否可信
31. {
32. fprintf(stderr, "Error loading trust store/n");
33. SSL\_CTX\_free(ctx);
34. **return** 0;
35. }
37. /\* Setup the connection \*/
39. sslbio = BIO\_new\_ssl\_connect(ctx);//建立ssl类型的bio
41. /\* Set the SSL\_MODE\_AUTO\_RETRY flag \*/
43. BIO\_get\_ssl(sslbio, & ssl);//从已建立的ssl类型的bio sslbio中得到ssl变量
44. SSL\_set\_mode(ssl, SSL\_MODE\_AUTO\_RETRY);//设置ssl的模式为SSL\_MODE\_AUTO\_RETRY,使用这个选项进行设置，如果服务器突然希望进行一次新的握手，那么OpenSSL 可以在后台处理它。

47. /\* Create and setup the connection \*/
49. BIO\_set\_conn\_hostname(sslbio, "127.0.0.1:4433");
51. **if**(BIO\_do\_connect(sslbio) <= 0)//发起握手请求
52. {
53. fprintf(stderr, "Error attempting to connect/n");
54. ERR\_print\_errors\_fp(stderr);
55. BIO\_free\_all(sslbio);
56. SSL\_CTX\_free(ctx);
57. **return** 0;
58. }
59. **else** printf("connent to server successful!/n");
61. /\* Check the certificate \*/
63. **if**(SSL\_get\_verify\_result(ssl) != X509\_V\_OK)//验证对方的证书是否是合法的（时间未过期，等等。。。）
64. {
65. fprintf(stderr, "Certificate verification error: %ld/n", SSL\_get\_verify\_result(ssl));
66. BIO\_free\_all(sslbio);
67. SSL\_CTX\_free(ctx);
68. **return** 0;
69. }
70. **else** printf("verify server cert successful/n");
72. **char** buf[1024];
73. **for**(;;)
74. {
75. printf("/ninput:");
76. scanf("%s",&buf[0]);
77. BIO\_write(sslbio,buf,strlen(buf));
78. p = BIO\_read(sslbio, buf, 1023);
79. **if**(p <= 0) **break**;
80. buf[p] = 0;
81. printf("%s/n", buf);
82. memset(buf,0,1024);
83. }
85. /\* Close the connection and free the context \*/
86. BIO\_ssl\_shutdown(sslbio);
87. BIO\_free\_all(sslbio);
88. SSL\_CTX\_free(ctx);
89. **return** 0;
90. }

近来研究如何使用openssl进行编程，下面是一些概要。

1.使用相同的ca生成两个证书，一个是server.cer，一个是client.cer，注意生成server.cer的时候必须指明证书可以用于服务端的。

服务器代码：

1. #include "openssl/bio.h"
2. #include "openssl/ssl.h"
3. #include "openssl/err.h"
5. #include <cutil.h>
7. #define EXIT\_IF\_TRUE(x) if (x)                              \
8. do {                                                    \
9. fprintf(stderr, "Check '%s' is true\n", #x);    \
10. ERR\_print\_errors\_fp(stderr);                    \
11. exit(2);                                        \
12. }while(0)
14. int main(int argc, char \*\*argv)
15. {
16. SSL\_CTX     \*ctx;
17. SSL         \*ssl;
18. X509        \*client\_cert;
20. char szBuffer[1024];
21. int nLen;
23. struct sockaddr\_in addr;
24. int len;
25. int nListenFd, nAcceptFd;
27. // 初始化
28. cutil\_init();
29. cutil\_log\_set\_level(LOG\_ALL);
30. cutil\_log\_set\_stderr(1);
31. SSLeay\_add\_ssl\_algorithms();
32. OpenSSL\_add\_all\_algorithms();
33. SSL\_load\_error\_strings();
34. ERR\_load\_BIO\_strings();
36. // 我们使用SSL V3,V2
37. EXIT\_IF\_TRUE((ctx = SSL\_CTX\_new (SSLv23\_method())) == NULL);
39. // 要求校验对方证书
40. SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER, NULL);
42. // 加载CA的证书
43. EXIT\_IF\_TRUE (!SSL\_CTX\_load\_verify\_locations(ctx, "cacert.cer", NULL));
45. // 加载自己的证书
46. EXIT\_IF\_TRUE (SSL\_CTX\_use\_certificate\_file(ctx, "server.cer", SSL\_FILETYPE\_PEM) <= 0) ;
48. // 加载自己的私钥
49. EXIT\_IF\_TRUE (SSL\_CTX\_use\_PrivateKey\_file(ctx, "server.key", SSL\_FILETYPE\_PEM) <= 0) ;
51. // 判定私钥是否正确
52. EXIT\_IF\_TRUE (!SSL\_CTX\_check\_private\_key(ctx));
54. // 创建并等待连接
55. nListenFd = cutil\_socket\_new(SOCK\_STREAM);
56. cutil\_socket\_bind(nListenFd, NULL, 8812, 1);
58. memset(&addr, 0, sizeof(addr));
59. len = sizeof(addr);
60. nAcceptFd = accept(nListenFd, (struct sockaddr \*)&addr, (size\_t \*)&len);
61. cutil\_log\_debug("Accept a connect from [%s:%d]\n",
62. inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));
64. // 将连接付给SSL
65. EXIT\_IF\_TRUE( (ssl = SSL\_new (ctx)) == NULL);
66. SSL\_set\_fd (ssl, nAcceptFd);
67. EXIT\_IF\_TRUE( SSL\_accept (ssl) != 1);
69. // 进行操作
70. memset(szBuffer, 0, sizeof(szBuffer));
71. nLen = SSL\_read(ssl,szBuffer, sizeof(szBuffer));
72. fprintf(stderr, "Get Len %d %s ok\n", nLen, szBuffer);
73. strcat(szBuffer, " this is from server");
74. SSL\_write(ssl, szBuffer, strlen(szBuffer));
76. // 释放资源
77. SSL\_free (ssl);
78. SSL\_CTX\_free (ctx);
79. close(nAcceptFd);
80. }

客户端代码

1. #include "openssl/bio.h"
2. #include "openssl/ssl.h"
3. #include "openssl/err.h"
5. #include <cutil.h>
7. #define EXIT\_IF\_TRUE(x) if (x)                              \
8. do {                                                    \
9. fprintf(stderr, "Check '%s' is true\n", #x);    \
10. ERR\_print\_errors\_fp(stderr);                    \
11. exit(2);                                        \
12. }while(0)
14. int main(int argc, char \*\*argv)
15. {
16. SSL\_METHOD  \*meth;
17. SSL\_CTX     \*ctx;
18. SSL         \*ssl;
20. int nFd;
21. int nLen;
22. char szBuffer[1024];
24. // 初始化
25. cutil\_init();
26. cutil\_log\_set\_level(LOG\_ALL);
27. cutil\_log\_set\_stderr(1);
28. SSLeay\_add\_ssl\_algorithms();
29. OpenSSL\_add\_all\_algorithms();
30. SSL\_load\_error\_strings();
31. ERR\_load\_BIO\_strings();
33. // 我们使用SSL V3,V2
34. EXIT\_IF\_TRUE((ctx = SSL\_CTX\_new (SSLv23\_method())) == NULL);
36. // 要求校验对方证书
37. SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER, NULL);
39. // 加载CA的证书
40. EXIT\_IF\_TRUE (!SSL\_CTX\_load\_verify\_locations(ctx, "cacert.cer", NULL));
42. // 加载自己的证书
43. EXIT\_IF\_TRUE (SSL\_CTX\_use\_certificate\_file(ctx, "client.cer", SSL\_FILETYPE\_PEM) <= 0) ;
45. // 加载自己的私钥
46. EXIT\_IF\_TRUE (SSL\_CTX\_use\_PrivateKey\_file(ctx, "client.key", SSL\_FILETYPE\_PEM) <= 0) ;
48. // 判定私钥是否正确
49. EXIT\_IF\_TRUE (!SSL\_CTX\_check\_private\_key(ctx));
51. // 创建连接
52. nFd = cutil\_socket\_new(SOCK\_STREAM);
53. if(cutil\_socket\_connect(nFd, "127.0.0.1", 8812, 30) < 0)
54. {
55. cutil\_log\_error("连接服务器失败\n");
56. return -1;
57. }
59. // 将连接付给SSL
60. EXIT\_IF\_TRUE( (ssl = SSL\_new (ctx)) == NULL);
61. SSL\_set\_fd (ssl, nFd);
62. EXIT\_IF\_TRUE( SSL\_connect (ssl) != 1);
64. // 进行操作
65. sprintf(szBuffer, "this is from client %d", getpid());
66. SSL\_write(ssl, szBuffer, strlen(szBuffer));
68. // 释放资源
69. memset(szBuffer, 0, sizeof(szBuffer));
70. nLen = SSL\_read(ssl,szBuffer, sizeof(szBuffer));
71. fprintf(stderr, "Get Len %d %s ok\n", nLen, szBuffer);
73. SSL\_free (ssl);
74. SSL\_CTX\_free (ctx);
75. close(nFd);
76. }