原创 rtoax 2019-08-10 20:46:44 1246 收藏 版权
分类专栏： 【计算机网络】

Table of Contents

HOWTO: Using Openssl C library

http://theshybulb.com/2015/10/10/use-openssl-c-library.html

Oct 10, 2015

For one of the Matasano crypto challenges, I had to decrypt the text which was encrypted using AES in ECB mode. Everything about AES is actually documented by the National Institute of Standards and Technology. You can get all the algorithms behind AES encryption. It is probably not a good idea to implement it from scratch. Openssl has a well tested and widely used library which works.

This Openssl library page gives a complete example of how to use them. There are a few preparatory steps before you can use the instructions though. These instructions are for Ubuntu like Linux distributions. These worked well on my Raspberry Pi too.

Installing Openssl library

Following command installs all the C libraries needed to use Openssl with your C code.

```
sudo apt-get install libssl-dev
```

For example, you will want to include the following header files:

```
#include <openssl/evp.h>
#include <openssl/ssl.h>
#include <openssl/rsa.h>
#include <openssl/x509.h>
```

Compiling your C program with the Openssl library

Next, you can follow the instructions from the Openssl crypto library page to create your C program. I have an example program in my Crytopals Github repository. While linking the program you need to provide the ssl and crypto library names. Following command should do it:

```
gcc yourprogram.c -lssl -lcrypto
```

A few pointers on the `do_crypt` function

- If you are going to use the `do_crypt` function for decrypting a text encrypted using electronic code book (ECB) mode, you should remove the following assert line since there is no Initialization Vector for ECB.

```
OPENSSL_assert(EVP_CIPHER_CTX_iv_length(&ctx) == 16);
```

- The example code operates on the raw data. So, if you are trying to decrypt the data which is base64 encoded, your first step should be to convert it into raw data.

Libssl API

https://wiki.openssl.org/index.php/Libssl_API

libssl is the portion of OpenSSL which supports TLS (SSL and TLS Protocols), and depends on libcrypto.

This is a **C** api. To use it you need to include (at least) **openssl/ssl.h** and to link your program with libssl library.

- Diffie-Hellman parameters
- Hostname validation

How to get libssl API to compile with it[edit]

on Debian base (debian, ubuntu, ...) you would need libssl-dev : apt-get install libssl-dev.

on Redhat base (RedHat, Fedora, ...) you would need openssl-devel : yum install openssl-devel

You can get sources directly too to compile statically over it.

例：Diffie-Hellman parameters

To use perfect forward secrecy cipher suites, you must set up Diffie-Hellman parameters (on the server side), or the PFS cipher suites will be silently ignored.

Contents

[hide]

Diffie-Hellman[edit]

`SSL_CTX_set_tmp_dh` is used to set the Diffie-Hellman parameters for a context. One of the easiest ways to get Diffie-Hellman parameters to use with this function is to generate random Diffie-Hellman parameters with the dhparam command-line program with the `-C` option, and embed the resulting code fragment in your program. For example, `openssl dhparam -C 2236` might result in:

```
#ifndef HEADER_DH_H
#include <openssl/dh.h>
#endif
DH *get_dh2236()
    {
    static unsigned char dh2236_p[]={
        0x0F,0x52,0xE5,0x24,0xF5,0xFA,0x9D,0xDC,0xC6,0xAB,0xE6,0x04,
        0xE4,0x20,0x89,0x8A,0xB4,0xBF,0x27,0xB5,0x4A,0x95,0x57,0xA1,
        0x06,0xE7,0x30,0x73,0x83,0x5E,0xC9,0x23,0x11,0xED,0x42,0x45,
        0xAC,0x49,0xD3,0xE3,0xF3,0x34,0x73,0xC5,0x7D,0x00,0x3C,0x86,
        0x63,0x74,0xE0,0x75,0x97,0x84,0x1D,0x0B,0x11,0xDA,0x04,0xD0,
        0xFE,0x4F,0xB0,0x37,0xDF,0x57,0x22,0x2E,0x96,0x42,0xE0,0x7C,
        0xD7,0x5E,0x46,0x29,0xAF,0xB1,0xF4,0x81,0xAF,0xFC,0x9A,0xEF,
        0xFA,0x89,0x9E,0x0A,0xFB,0x16,0xE3,0x8F,0x01,0xA2,0xC8,0xDD,
        0xB4,0x47,0x12,0xF8,0x29,0x09,0x13,0x6E,0x9D,0xA8,0xF9,0x5D,
        0x08,0x00,0x3A,0x8C,0xA7,0xFF,0x6C,0xCF,0xE3,0x7C,0x3B,0x6B,
        0xB4,0x26,0xCC,0xDA,0x89,0x93,0x01,0x73,0xA8,0x55,0x3E,0x5B,
        0x77,0x25,0x8F,0x27,0xA3,0xF1,0xBF,0x7A,0x73,0x1F,0x85,0x96,
        0x0C,0x45,0x14,0xC1,0x06,0xB7,0x1C,0x75,0xAA,0x10,0xBC,0x86,
        0x98,0x75,0x44,0x70,0xD1,0x0F,0x20,0xF4,0xAC,0x4C,0xB3,0x88,
        0x16,0x1C,0x7E,0xA3,0x27,0xE4,0xAD,0xE1,0xA1,0x85,0x4F,0x1A,
        0x22,0x0D,0x05,0x42,0x73,0x69,0x45,0xC9,0x2F,0xF7,0xC2,0x48,
        0xE3,0xCE,0x9D,0x74,0x58,0x53,0xE7,0xA7,0x82,0x18,0xD9,0x3D,
        0xAF,0xAB,0x40,0x9F,0xAA,0x4C,0x78,0x0A,0xC3,0x24,0x2D,0xDB,
        0x12,0xA9,0x54,0xE5,0x47,0x87,0xAC,0x52,0xFE,0xE8,0x3D,0x0B,
        0x56,0xED,0x9C,0x9F,0xFF,0x39,0xE5,0xE5,0xBF,0x62,0x32,0x42,
        0x08,0xAE,0x6A,0xED,0x88,0x0E,0xB3,0x1A,0x4C,0xD3,0x08,0xE4,
        0xC4,0xAA,0x2C,0xCC,0xB1,0x37,0xA5,0xC1,0xA9,0x64,0x7E,0xEB,
        0xF9,0xD3,0xF5,0x15,0x28,0xFE,0x2E,0xE2,0x7F,0xFE,0xD9,0xB9,
        0x38,0x42,0x57,0x03,
      } ;
    static unsigned char dh2236_g[]={
        0x02,
      } ;
 DH *dh;

    if ((dh=DH_new()) == NULL) return(NULL);
 dh->p=BN_bin2bn(dh2236_p,sizeof(dh2236_p),NULL);
 dh->g=BN_bin2bn(dh2236_g,sizeof(dh2236_g),NULL);
    if ((dh->p == NULL) || (dh->g == NULL))
      {   DH_free(dh); return(NULL); }
    return(dh);
    }
```

which can then be used like this:

```
DH *dh = get_dh2236 ();
if (1 != SSL_CTX_set_tmp_dh (ctx, dh))
  error ();
DH_free (dh);
```

Be sure to choose a bit length appropriate to the security level you want to achieve, although keep in mind that Diffie-Hellman parameters longer than 2236 bits may be incompatible with older versions of NSS. Even worse, it appears that versions of Java prior to 1.7 don't support Diffie-Hellman parameters longer than 1024 bits!

Validating Parameters[edit]

The Diffie-Hellman parameters should be validated after loading. To perform paramter validation, you call DH_check. DH_check returns 0 or a bitmask values of the following:

- DH_CHECK_P_NOT_PRIME (0x01)
- DH_CHECK_P_NOT_SAFE_PRIME (0x02)
- DH_UNABLE_TO_CHECK_GENERATOR (0x04)
- DH_NOT_SUITABLE_GENERATOR (0x08)

The validation code might look as follows (error checking omitted for clarity):

```
BIO* bio = ...;
DH* dh = PEM_read_bio_DHparams(bio, NULL, NULL, NULL);

int rc, codes = 0;
rc = DH_check(dh, &codes);
assert(rc == 1);

if(BN_is_word(dh->g, DH_GENERATOR_2))
{
    long residue = BN_mod_word(dh->p, 24);
    if(residue == 11 || residue == 23) {
        codes &= ~DH_NOT_SUITABLE_GENERATOR;
    }
}

if (codes & DH_UNABLE_TO_CHECK_GENERATOR)
    printf("DH_check: failed to test generator\n");

if (codes & DH_NOT_SUITABLE_GENERATOR)
    printf("DH_check: g is not a suitable generator\n");

if (codes & DH_CHECK_P_NOT_PRIME)
    printf("DH_check: p is not prime\n");

if (codes & DH_CHECK_P_NOT_SAFE_PRIME)
    printf("DH_check: p is not a safe prime\n");
```

The additional call to BN_mod_word(dh->p, 24) (and unmasking of DH_NOT_SUITABLE_GENERATOR) is performed to ensure your program accepts IETF group parameters. OpenSSL checks the prime is congruent to 11 when g = 2; while the IETF's primes are congruent to 23 when g = 2. Without the test, the IETF parameters would fail validation. For details, see Diffie-Hellman Parameter Check (when g = 2, must p mod 24 == 11?).

Elliptic curve Diffie-Hellman[edit]

For elliptic curve Diffie-Hellman, you can do something like this:

```
EC_KEY *ecdh = EC_KEY_new_by_curve_name (NID_X9_62_prime256v1);
if (! ecdh)
  error ();
if (1 != SSL_CTX_set_tmp_ecdh (ctx, ecdh))
  error ();
EC_KEY_free (ecdh);
```

Or, in OpenSSL 1.0.2 (not yet released, as of Feb 2013) and higher, you should be able to do:

```
SSL_CTX_set_ecdh_auto (ctx, 1)
```

For more information, see Elliptic Curve Diffie Hellman and Elliptic Curve Cryptography.

RFC 3526 PEM Encoded Groups[edit]

Below are three Diffie-Hellman MODP groups specified in RFC 3526, More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) (the 1024-bit parameter is from RFC 2409). They can be used with PEM_read_bio_DHparams and a memory BIO. RFC 3526 also offers 1536-bit, 6144-bit and 8192-bit primes.

```
static const char g_dh1024_sz[] =
    "-----BEGIN DH PARAMETERS-----\n"
    "MIGHAoGBAP//yQ/aoiFowjTExmKLgNwc0SkCTgiKZ8x0Agu+pjsTmyJR\n"
    "Sgh5jjQE3e+VGbPNOkMbMCsKbfJfFDdP4TVtbVHCReSFtXZiXn7G9ExC6aY37WsL\n"
    "/1y29Aa37e44a/taiZ+lrp8kEXxLH+ZJKGZR7OZTgf//AgEC\n"
    "-----END DH PARAMETERS-----";

static const char g_dh1536_sz[] = "-----BEGIN DH PARAMETERS-----\n"
    "MIHHAoHBAP//yQ/aoiFowjTExmKLgNwc0SkCTgiKZ8x0Agu+pjsTmyJR\n"
    "Sgh5jjQE3e+VGbPNOkMbMCsKbfJfFDdP4TVtbVHCReSFtXZiXn7G9ExC6aY37WsL\n"
    "/1y29Aa37e44a/taiZ+lrp8kEXxLH+ZJKGZR7ORbPcIAfLihY78FmNpINhxV05pp\n"
    "Fj+o/STPX4NlXSPco62WHGLzViCFUrue1SkHcJaWbWcMNU5KvJgE8XRsCMojcyf/\n"
    "/wIBAg==\n"
    "-----END DH PARAMETERS-----";

static const char g_dh2048_sz[] =
    "-----BEGIN DH PARAMETERS-----\n"
    "MIIBCAKCAQEA///JD9qiIWjCNMTGYouA3BzRKQJOCIpnzHQCC76mOxOb\n"
    "IlFKCHmONATd75UZs806QxswKwpt8l8UN0/hNW1tUcJF5IW1dmJefsb0TELppjft\n"
    "awv/XLb0Brft7jhr+1qJn6WunyQRfEsf5kkoZlHs5Fs9wgB8uKFjvwWY2kg2HFXT\n"
    "mmkWP6j9JM9fg2VdI9yjrZYcYvNWIIVSu57VKQdwlpZtZww1Tkq8mATxdGwIyhgh\n"
    "fDKQXkYuNs474553LBgOhgObJ4Oi7Aeij7XFXfBvTFLJ3ivL9pVYFxg5lUl86pVq\n"
    "5RXSJhiY+gUQFXKOWoqsqmj//wIBAg==\n"
    "-----END DH PARAMETERS-----";

static const char g_dh3072_sz[] =
    "-----BEGIN DH PARAMETERS-----\n"
    "MIIBiAKCAYEA///JD9qiIWjCNMTGYouA3BzRKQJOCIpnzHQCC76mOxOb\n"
    "IlFKCHmONATd75UZs806QxswKwpt8l8UN0/hNW1tUcJF5IW1dmJefsb0TELppjft\n"
    "awv/XLb0Brft7jhr+1qJn6WunyQRfEsf5kkoZlHs5Fs9wgB8uKFjvwWY2kg2HFXT\n"
    "mmkWP6j9JM9fg2VdI9yjrZYcYvNWIIVSu57VKQdwlpZtZww1Tkq8mATxdGwIyhgh\n"
    "fDKQXkYuNs474553LBgOhgObJ4Oi7Aeij7XFXfBvTFLJ3ivL9pVYFxg5lUl86pVq\n"
    "5RXSJhiY+gUQFXKOWoqqxC2tMxcNBFB6M6hVIavfHLpk7PuFBFjb7wqK6nFXXQYM\n"
    "fbOXD4Wm4eTHq/WujNsJM9cejJTgSiVhnc7j0iYa0u5r8S/6BtmKCGTYdgJzPshq\n"
```

```
    "ZFIfKxgXeyAMu+EXV3phXWx3CYjAutlG4gjiT6B05asxQ9tb/OD9EI5LgtEgqTrS\n"
    "yv//AgEC\n"
    "-----END DH PARAMETERS-----";

static const char g_dh4096_sz[] =
    "-----BEGIN DH PARAMETERS-----\n"
    "MIICCAKCAgEA///JD9qiIWjCNMTGYouA3BzRKQJOCIpnzHQCC76mOxOb\n"
    "IlFKCHmONATd75UZs806QxswKwpt8l8UN0/hNW1tUcJF5IW1dmJefsb0TELppjft\n"
    "awv/XLb0Brft7jhr+1qJn6WunyQRfEsf5kkoZlHs5Fs9wgB8uKFjvwWY2kg2HFXT\n"
    "mmkWP6j9JM9fg2VdI9yjrZYcYvNWIIVSu57VKQdwlpZtZww1Tkq8mATxdGwIyhgh\n"
    "fDKQXkYuNs474553LBgOhgObJ4Oi7Aeij7XFXfBvTFLJ3ivL9pVYFxg5lUl86pVq\n"
    "5RXSJhiY+gUQFXKOWoqqxC2tMxcNBFB6M6hVIavfHLpk7PuFBFjb7wqK6nFXXQYM\n"
    "fbOXD4Wm4eTHq/WujNsJM9cejJTgSiVhnc7j0iYa0u5r8S/6BtmKCGTYdgJzPshq\n"
    "ZFIfKxgXeyAMu+EXV3phXWx3CYjAutlG4gjiT6B05asxQ9tb/OD9EI5LgtEgqSEI\n"
    "ARpyPBKnh+bXiHGaEL26WyaZwycYavTiPBqUaDS2FQvaJYPpyirUTOjbu8LbBN6O\n"
    "+S6O/BQfvsqmKHxZR05rwF2ZspZPoJDDoiM7oYZRW+ftH2EpcM7i16+4G912IXBI\n"
    "HNAGkSfVsFqpk7TqmI2P3cGG/7fckKbAj030Nck0BjGZ//8CAQI=\n"
    "-----END DH PARAMETERS-----";
```

例：Hostname validation

OpenSSL 1.1.0 provides built-in functionality for hostname checking and validation. Viktor Dukhovni provided the implementation in January, 2015. Its been available in Master since that time. The code is beginning to see widespread testing as the release of OpenSSL 1.1.0 approaches.

One common mistake made by users of OpenSSL is to assume that OpenSSL will validate the hostname in the server's certificate. Versions prior to 1.0.2 did not perform hostname validation. Version 1.0.2 and up contain support for hostname validation, but they still require the user to call a few functions to set it up.

A man page on hostname validation has been available since 1.0.2. Also see the **X509_check_host()**.

Example Usage[edit]

The following is from Hostname validation and shows how you could use OpenSSL's built-in hostname validation.

```
const char servername[] = "www.example.com";
SSL *ssl = NULL;
X509_VERIFY_PARAM *param = NULL;
...

servername = "www.example.com";
ssl = SSL_new(...);
param = SSL_get0_param(ssl);

/* Enable automatic hostname checks */
X509_VERIFY_PARAM_set_hostflags(param, X509_CHECK_FLAG_NO_PARTIAL_WILDCARDS);
if (!X509_VERIFY_PARAM_set1_host(param, servername, sizeof(servername) - 1)) {
  // handle error
  return 0;
}

/* Enable peer verification, (with a non-null callback if desired) */
SSL_set_verify(ssl, SSL_VERIFY_PEER, NULL);

/*
 * Establish SSL connection, hostname should be checked
 * automatically test with a hostname that should not match,
 * the connection will fail (unless you specify a callback
```

```
 * that returns despite the verification failure.  In that
 * case SSL_get_verify_status() can expose the problem after
 * connection completion.
 */
...
```

The above works starting with OpenSSL 1.0.2. A simpler interface is available starting with OpenSSL 1.1.0:

```
...
 SSL_set_hostflags(ssl, X509_CHECK_FLAG_NO_PARTIAL_WILDCARDS);
 if (!SSL_set1_host(ssl, "www.example.com")) {
   /* handle error */
 }
 /* Enable peer verification (with a non-null callback if desired) */
 SSL_set_verify(ssl, SSL_VERIFY_PEER, NULL);
 ...
```

documentation at SSL_set1_host

Wildcard support is configured via the flags documented for X509_check_host(), the two most frequently useful are:

- **X509_CHECK_FLAG_NO_WILDCARDS**
- **X509_CHECK_FLAG_NO_PARTIAL_WILDCARDS**

populate the X509_VERIFY_PARAMS with the desired hostname, and let the OpenSSL code call X509_check_host automatically.

This makes it easier to some day enable DANE TLSA support, because with DANE, name checks need to be skipped for DANE-EE(3) TLSA records, as the DNSSEC TLSA records provides the requisite name binding instead.

Also with the X509_VERIFY_PARAM approach, name checks happen early, and for applications that don't continue handshakes with unauthenticated peers, terminate as early as possible.

There is an associated new X509 error code: **X509_V_ERR_HOSTNAME_MISMATCH**

SSL Conservatory and cURL code[edit]

This was the original information, might still be valid for < 1.0.2 openssl versions :

The ssl-conservatory repository shows how validating the hostname can be done. However, the ssl-conservatory code does not handle wildcard certificates, so borrowing some code from cURL might be one way to go instead. This commitshows how to graft the wildcard-matching code from cURL into the ssl-conservatory code.

Below is a copy of openssl_hostname_validation.c, although to compile it also needs the files hostcheck.h, hostcheck.c, and openssl_hostname_validation.h.

```
/* Obtained from: https://github.com/iSECPartners/ssl-conservatory */

/*
Copyright (C) 2012, iSEC Partners.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

```

```
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
 */

/*
 * Helper functions to perform basic hostname validation using OpenSSL.
 *
 * Please read "everything-you-wanted-to-know-about-openssl.pdf" before
 * attempting to use this code. This whitepaper describes how the code works,
 * how it should be used, and what its limitations are.
 *
 * Author:  Alban Diquet
 * License: See LICENSE
 *
 */

// Get rid of OSX 10.7 and greater deprecation warnings.
#if defined(__APPLE__) && defined(__clang__)
#pragma clang diagnostic ignored "-Wdeprecated-declarations"
#endif

#include <openssl/x509v3.h>
#include <openssl/ssl.h>

#include "openssl_hostname_validation.h"
#include "hostcheck.h"

#define HOSTNAME_MAX_SIZE 255

/**
* Tries to find a match for hostname in the certificate's Common Name field.
*
* Returns MatchFound if a match was found.
* Returns MatchNotFound if no matches were found.
* Returns MalformedCertificate if the Common Name had a NUL character embedded in it.
* Returns Error if the Common Name could not be extracted.
*/
static HostnameValidationResult matches_common_name(const char *hostname, const X509 *server_cert) {
        int common_name_loc = -1;
        X509_NAME_ENTRY *common_name_entry = NULL;
        ASN1_STRING *common_name_asn1 = NULL;
        char *common_name_str = NULL;

        // Find the position of the CN field in the Subject field of the certificate
        common_name_loc = X509_NAME_get_index_by_NID(X509_get_subject_name((X509 *) server_cert), NID_commonName,
        if (common_name_loc < 0) {
                return Error;
        }

        // Extract the CN field
        common_name_entry = X509_NAME_get_entry(X509_get_subject_name((X509 *) server_cert), common_name_loc);
        if (common_name_entry == NULL) {
                return Error;
```

```
        }

        // Convert the CN field to a C string
        common_name_asn1 = X509_NAME_ENTRY_get_data(common_name_entry);
        if (common_name_asn1 == NULL) {
                return Error;
        }
        common_name_str = (char *) ASN1_STRING_data(common_name_asn1);

        // Make sure there isn't an embedded NUL character in the CN
        if ((size_t)ASN1_STRING_length(common_name_asn1) != strlen(common_name_str)) {
                return MalformedCertificate;
        }

        // Compare expected hostname with the CN
        if (Curl_cert_hostcheck(common_name_str, hostname) == CURL_HOST_MATCH) {
                return MatchFound;
        }
        else {
                return MatchNotFound;
        }
}


/**
* Tries to find a match for hostname in the certificate's Subject Alternative Name extension.
*
* Returns MatchFound if a match was found.
* Returns MatchNotFound if no matches were found.
* Returns MalformedCertificate if any of the hostnames had a NUL character embedded in it.
* Returns NoSANPresent if the SAN extension was not present in the certificate.
*/
static HostnameValidationResult matches_subject_alternative_name(const char *hostname, const X509 *server_cert) {
        HostnameValidationResult result = MatchNotFound;
        int i;
        int san_names_nb = -1;
        STACK_OF(GENERAL_NAME) *san_names = NULL;

        // Try to extract the names within the SAN extension from the certificate
        san_names = X509_get_ext_d2i((X509 *) server_cert, NID_subject_alt_name, NULL, NULL);
        if (san_names == NULL) {
                return NoSANPresent;
        }
        san_names_nb = sk_GENERAL_NAME_num(san_names);

        // Check each name within the extension
        for (i=0; i<san_names_nb; i++) {
                const GENERAL_NAME *current_name = sk_GENERAL_NAME_value(san_names, i);

                if (current_name->type == GEN_DNS) {
                        // Current name is a DNS name, let's check it
                        char *dns_name = (char *) ASN1_STRING_data(current_name->d.dNSName);

                        // Make sure there isn't an embedded NUL character in the DNS name
                        if ((size_t)ASN1_STRING_length(current_name->d.dNSName) != strlen(dns_name)) {
                                result = MalformedCertificate;
                                break;
                        }
                        else { // Compare expected hostname with the DNS name
                                if (Curl_cert_hostcheck(dns_name, hostname)
                                    == CURL_HOST_MATCH) {
```

```
135                                     result = MatchFound;
136                                     break;
137                             }
138                     }
139             }
140     }
141     sk_GENERAL_NAME_pop_free(san_names, GENERAL_NAME_free);
142
143     return result;
144 }
145
146
147 /**
148  * Validates the server's identity by looking for the expected hostname in the
149  * server's certificate. As described in RFC 6125, it first tries to find a match
150  * in the Subject Alternative Name extension. If the extension is not present in
151  * the certificate, it checks the Common Name instead.
152  *
153  * Returns MatchFound if a match was found.
154  * Returns MatchNotFound if no matches were found.
155  * Returns MalformedCertificate if any of the hostnames had a NUL character embedded in it.
156  * Returns Error if there was an error.
157  */
158 HostnameValidationResult validate_hostname(const char *hostname, const X509 *server_cert) {
159     HostnameValidationResult result;
160
161     if((hostname == NULL) || (server_cert == NULL))
162             return Error;
163
164     // First try the Subject Alternative Names extension
165     result = matches_subject_alternative_name(hostname, server_cert);
166     if (result == NoSANPresent) {
167             // Extension was not found: try the Common Name
168             result = matches_common_name(hostname, server_cert);
169     }
170
171     return result;
172 }
```

相关博客文章

《OpenSSL API》https://blog.csdn.net/sunweixiang1002/article/details/83029232

《openssl api 进行c语言编程的问题》https://bbs.csdn.net/topics/392171145?page=1

《使用 OpenSSL API 进行安全编程》https://www.ibm.com/developerworks/cn/linux/l-openssl.html

《OpenSSL使用之API指南》http://www.360doc.com/content/16/1210/00/17203858_613428107.shtml

rtoax
码龄4年 暂无认证

1163	2774	335	175万+	
原创	周排名	总排名	访问	等级

2万+	1544	1379	2153	4540
积分	粉丝	获赞	评论	收藏

热门文章

最新评论

您愿意向朋友推荐“博客详情页”吗？

强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

2021

11月	10月	09月	08月
8篇	26篇	53篇	28篇

07月 49篇 | 06月 80篇 | 05月 70篇 | 04月 51篇

03月 103篇 | 02月 36篇 | 01月 19篇

2020年 709篇 | 2019年 507篇

2018年 222篇 | 2017年 94篇

pem格式证书编码 x509_SSL证书编码格式 — weixin_39999859的博客 276

Certificates and EncodingsAt its core an X.509 certificate is a digital document that has been encoded and/or digitally signed according to RFC 5280.I…

rtoax 关注

0 0 0

专栏目录

举报