CC32xx SSL Demo Application

Overview

SSL is the universally accepted means by which communication is authenticated and encrypted on the World Wide Web. SSL certificates are designed to provide two principles, privacy and authentication. Privacy is achieved by encryption/decryption and authentication is achieved by signature/verification. This wiki will introduce a user to SSL/TLS and its implementation on the CC3200 devices.

Application details

Protocol and Ciphers

The SSL Protocol supports the use of a variety of different encryption/decryption algorithms - also known as ciphers - for use in operations such as authenticating the connection between a server and client, transmitting certificates, and establishing session keys. Depending on the version of SSL supported, clients and servers may support different sets of ciphers. The following methods and ciphers are supported by CC32xx.

Method	Cipher
SSLv3	RSA_WITH_RC4_128_SHA
SSLv3	RSA_WITH_RC4_128_MD5
TLSv1	RSA_WITH_RC4_128_SHA
TLSv1	RSA_WITH_RC4_128_MD5
TLSv1	RSA_WITH_AES_256_CBC_SHA
TLSv1	DHE_RSA_WITH_AES_256_CBC_SHA
TLSv1	ECDHE_RSA_WITH_AES_256_CBC_SHA
TLSv1	ECDHE_RSA_WITH_RC4_128_SHA

API for SSL

The CC32xx has extended the BSD Socket API in order to support the SSL layer. At the application level, the basic socket flow when using a secured socket is kept the same; operations such as connect, accept, send, recv or select are supported.

When a client application 'connects' to a secure socket, the function will only return successfully if a secure session was established with the server successfully. An error is returned if the secure session is not established. After the connection is established, the data path is secure.

When a server application 'accepts' a connection over a secure opened socket, the function will only return successfully if a secure session was established with the client successfully. If the client is rejected by the secure session, the CC32xx will automatically prepare itself to accept a new connection from another client without application interference. In other words, if a secure client connection is not successful, there is no need to recall the accept() function via the user application. After the connection is established, the data path is secure.

If the remote side decides to downgrade the connection to an unsecured socket, recv() will return with the ESECCLOSED error. How the socket is handled (whether to close the socket or continue unsecured) will need to be

decided by the application.

Note: The CC32xx cannot initiate a downgrade to an unsecured socket, nor can it dynamically upgrade from an insecure to secure socket.

How-to/Program Flow

Below is the application program flow used to establish and configure a secure socket:

Set Current Time in the Device (required)

To begin, Current time must be set in the device. This time is used to validate the certificate. If the Time is beyond the validity period of Certificate, sl_connect will return error.

```
g_time.tm_day = DATE;
g_time.tm_mon = MONTH;
g_time.tm_year = YEAR;
g_time.tm_sec = HOUR;
g_time.tm_hour = MINUTE;
g_time.tm_min = SECOND;

retVal = sl_DeviceSet(SL_DEVICE_GENERAL_CONFIGURATION,
                          SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME,
                          sizeof(SlDateTime),(unsigned char *)(&g_time));
```

Open Secure Socket (required)

A secure socket must be opened by the CC32xx device. The sl_Socket() function may be used with the "Protocol" parameter set to SL_SEC_SOCKET (value=100).

```
SockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, SL_SEC_SOCKET);
```

Force specific method (optional)

The CC32xx supports the SSL 3.0, TLS 1.0, TLS 1.1, and TLS 1.2 protocols/methods. By default, SSL 3.0 and TLS 1.2 are enabled. A specific method can be forced by using the sl_SetSockOpt() function.

```
char method = SL_SO_SEC_METHOD_SSLV3;

Status = sl_SetSockOpt(SockID, SL_SOL_SOCKET, SL_SO_SEC_METHOD, &method, sizeof(method));
```

Where *method* can be chosen from the following list:

- SL_SO_SEC_METHOD_SSLV3
- SL_SO_SEC_METHOD_TLSV1
- SL_SO_SEC_METHOD_TLSV1_1
- SL_SO_SEC_METHOD_TLSV1_2
- SL_SO_SEC_METHOD_SSLv3_TLSV1_2

Force specific cipher (optional)

By default, the CC32xx will pick the most secure cipher suite that both sides of the connection can support. A specific cipher can be forced by using the sl_SetSockOpt() function.

```
long cipher = SL_SEC_MASK_SSL_RSA_WITH_RC4_128_SHA;


Status = sl_SetSockOpt(SockID, SL_SOL_SOCKET, SL_SO_SEC_MASK, &cipher,  sizeof(cipher));
```

Where *cipher* can be chosen from the following list:

- SL_SEC_MASK_SSL_RSA_WITH_RC4_128_SHA
- SL_SEC_MASK_SSL_RSA_WITH_RC4_128_MD5
- SL_SEC_MASK_TLS_RSA_WITH_AES_256_CBC_SHA
- SL_SEC_MASK_TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- SL_SEC_MASK_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- SL_SEC_MASK_TLS_ECDHE_RSA_WITH_RC4_128_SHA

Files/Variables related to TLS/SSL

The CC32xx uses files specific to TLS/SSL that may be defined by the user at the application level. The files needed are listed below based on the connection type, and must be expressed in the DER format.

Client Files

1. **Private Key** - used when server verifies the client. If file mapped to 0, the connection may be refused by the server if the server wishes to verify client.
2. **Certificate** - used when server verifies the client. If file mapped to 0, the connection may be refused by the server if the server wishes to verify client.
3. **CA, Certificate Authority** - used when verifying the certificate provided by the server. There is the option to disable server verification by mapping file to id 0. In that case, if the secure session established successfully, connect will return with a specific error (ESECSNOVERIFY). It`s application decision to close socket, or to ignore it and continue with secured data.
4. **DH, Diffie-Hellman key** - this file is not needed in the client case, map to id 0.

Server Files

1. **Private Key** - this file is always needed by a server.
2. **Certificate** - the file is always needed by a server.
3. **CA, Certificate Authority** - used when server verifies the client. By mapping file to id 0, server will not try to verify the certificate of the client.
4. **DH, Diffie-Hellman key** - This key is only needed for the following cipher suites: DHE_RSA_WITH_AES_256_CBC_SHA, ECDHE_RSA_WITH_AES_256_CBC_SHA, or ECDHE_RSA_WITH_RC4_128_SHA.

Mapping the TLS/SSL Files/Variables

The TLS/SSL Files/Variables can be defined and mapped to a socket using the following code.

```
typedef struct
{
  unsigned char PrivateKey;
  unsigned char Certificate;
  unsigned char CA;
  unsigned char DH;
}SlSockSecureFiles_t;


SlSockSecureFiles_t SecureFiles;
sockSecureFiles.secureFiles[0] = 0;  // mapping private key, 0 file not exist
sockSecureFiles.secureFiles[1] = 0;  // mapping certificate, 0 file not exist
sockSecureFiles.secureFiles[2] = SL_SSL_CA_CERT/*129*/; // mapping CA, 0 file not exist
sockSecureFiles.secureFiles[3] = 0;  // mapping certificate, 0 file not exist


Status = sl_SetSockOpt(SockID, SL_SOL_SOCKET, SL_SO_SEC_FILES, & SecureFiles, sizeof(SlSockSecureFiles));
```

Source Files briefly explained

- **main.c** - The main file that explains how certificate can be used with SSL.

Supporting Files

- **pinmux.c** - Generated by the PinMUX utility.
- **startup_ccs.c** - CCS related functions
- **startup_ewarm.c** - IAR related functions
- **gpio_if.c** - GPIO interface APIs

Usage

1. Preload valid CA Certificate in DER format into SFLASH using Uniflash.For detailed instructions about using Uniflash, refer Uniflash User Guide.
   - Generate Google CA Certificate file. Look into CA certificate section below for more details
   - In Uniflash, click the "add file" option.
   - Name the file to /cert/129.der
   - In the Url field mention the path to the Google CA certificate
   - Select the Erase and Update check boxes and program
2. Run the reference application
   - Flash the bin
   - Open the Project in IAR/CCS. Build and download the application to the board
3. **sl_Connect** API should return with a non-negative value indicating successful connection with the server.
4. On the Board Led Red will be on If some error occurs. On successful execution Led Green Will be on.

CA Certificate

CA Certificate can be downloaded using various methods. For Example, On Windows 7 machine, procedure to Download CA Certificate of www.google.com is:

1. Press Start button
2. Typing certmgr.msc into the Search box, and then pressing ENTER.
3. Double Click Trusted Root Certificate Authorities
4. Double Click Certificate
5. Look for "Equifax Secure CA"
6. Double Click on it. It will open the Certificate.
7. Select Details Tab
8. Click on “Copy to File” button and export the certificate as .cer Format

For Firefox, the procedure is:

1. Open FireFox
2. Go to Tools->Options
3. Click the Advanced tab
4. In the Advanced, click the Certificates tab
5. Click the "View Certificates" button
6. Look for "Equifax Secure CA"
7. Mark it and export it as a .der format

Limitations/Known Issues

- SSL certificates must be preloaded to the serial flash. If CA certificate is not present Application throws an error at sl_Connect.
- SSL certificates are not encrypted
- SSL Certificate may change over time. Always Download the Valid CA Certificate
- Certificate Time is checked for its validity, Update the current time in the Source Code

Article Sources and Contributors

CC32xx SSL Demo Application *Source*: http://processors.wiki.ti.com/index.php?oldid=178632 *Contributors*: A0221015, Codycooke, Jitgupta, Malokyle

Image Sources, Licenses and Contributors

File:Cc31xx cc32xx return home.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Cc31xx_cc32xx_return_home.png *License*: unknown *Contributors*: A0221015
File:Cc32xx return sample apps.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Cc32xx_return_sample_apps.png *License*: unknown *Contributors*: A0221015