Skywire® 4G LTE CAT 4 AWS IoT with TLS

NimbeLink Corp

Updated: July 2020

Table of Contents

1. Introduction

1.1 Overview

This document serves as a guide for Amazon AWS connections using the NimbeLink 4G LTE CAT4 TC4NAG Skywire. This tutorial will document the configuration of the modem and the Amazon AWS settings, and will demonstrate two different connection examples.

1.2 Orderable Parts

Orderable Device	Description	Carrier	Network Type
NL-SWDK	Skywire Development Kit	Any	Any
NL-SW-LTE-TC4NAG	4G LTE CAT 4	AT&T, Verizon	LTE

1.3 Disclaimer

Please note that the NL-SW-LTE-TC4NAG's built-in TLS stack does not support server name indication (SNI). SNI is a TLS extension where the client indicates the hostname it is trying to connect to as part of the TLS handshake. For more information, please see: https://tools.ietf.org/html/rfc6066

When using the modem's built-in TLS stack, your endpoint must not require SNI for a successful TLS connection. If your endpoint requires SNI, you will need to run your own SSL/TLS stack on your host processor and use the NL-SW-LTE-TC4NAG's TCP sockets. For example, mbed TLS (https://tls.mbed.org/) and wolfSSL (https://www.wolfssl.com/) are two external TLS stacks. See https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations for more alternatives. NimbeLink does not provide support for integrating these embedded libraries into your host platform.

If you are using an operating system such as Windows or Linux, you can use the operating system's SSL/TLS stack over a data connection like PPP or CDC-ECM. Please see the Linux Networking Guide for more information: https://nimbelink.com/Documentation/Skywire/4G_LTE_Cat_4_Telit/1002199_NL-SW-LTE-TC4NAG_Linux_Networking_Guide.pdf

Alternatively, a proxy server could be set up that does not require SNI, but can communicate with the server that does. NimbeLink does not provide support for setting this up.

2. AWS IoT Setup

2.1 Preliminary Setup

Before starting, it is important to note that this guide assumes that the reader already has a valid Amazon AWS account. If this is not the case, Amazon offers a free trial account that can be used to test this guide. For more information about the free account, please follow this link:

https://aws.amazon.com/free/

2.2 Create a Policy

The first step in the AWS connection process is to create a policy. Login to the AWS IoT console at the following link:

https://console.aws.amazon.com/iot/

and navigate to the 'Secure' > 'Policies' menu. Once there, press the “Create a policy" button located near the center of the screen.

In the next page, choose a name (1.1) for the policy. Also choose “iot:*" for the “Action" (1.2) and “*" for the “Resource ARN" (1.2) fields. Check the “Allow" box (1.2), and then click "Add Statement". Finally, click "Create" (1.3) to create the policy. Refer to the image below.

Note that this policy is very permissive, and AWS recommends following a least permissive approach to policies. Example policies that follow this practice can be found at the following resources if you’re interested in learning more:

HTTP:
https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-rest-api.html?icmpid=docs_iot_console

MQTT:
https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-mqtt.html?icmpid=docs_iot_console

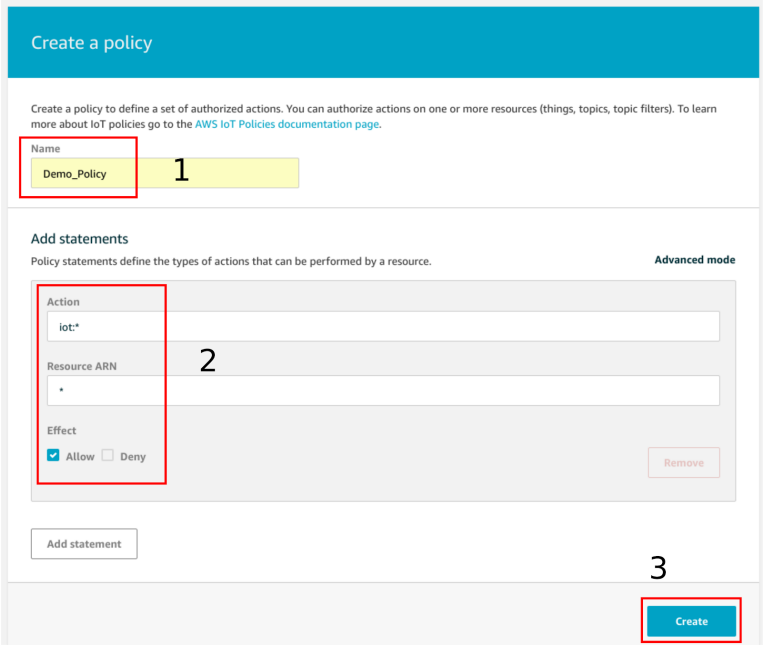


Figure 1. Create a Policy

2.3 Create a "Thing"

Next, navigate to 'Manage' > 'Things' (2.1) using the menu on the left-hand side of the dashboard. Next, select ‘Create’ (2.2) in the top right corner to make a new “thing". Refer to the image below for reference.

Figure 2. Thing Management Page

After pressing the “Create" button (2.2), select the “Create a single thing" option in the next page that loads. In the following page, enter a custom name in the appropriate box (3.1), and then press the “Next" button (3.2). The webpage should look something like this:

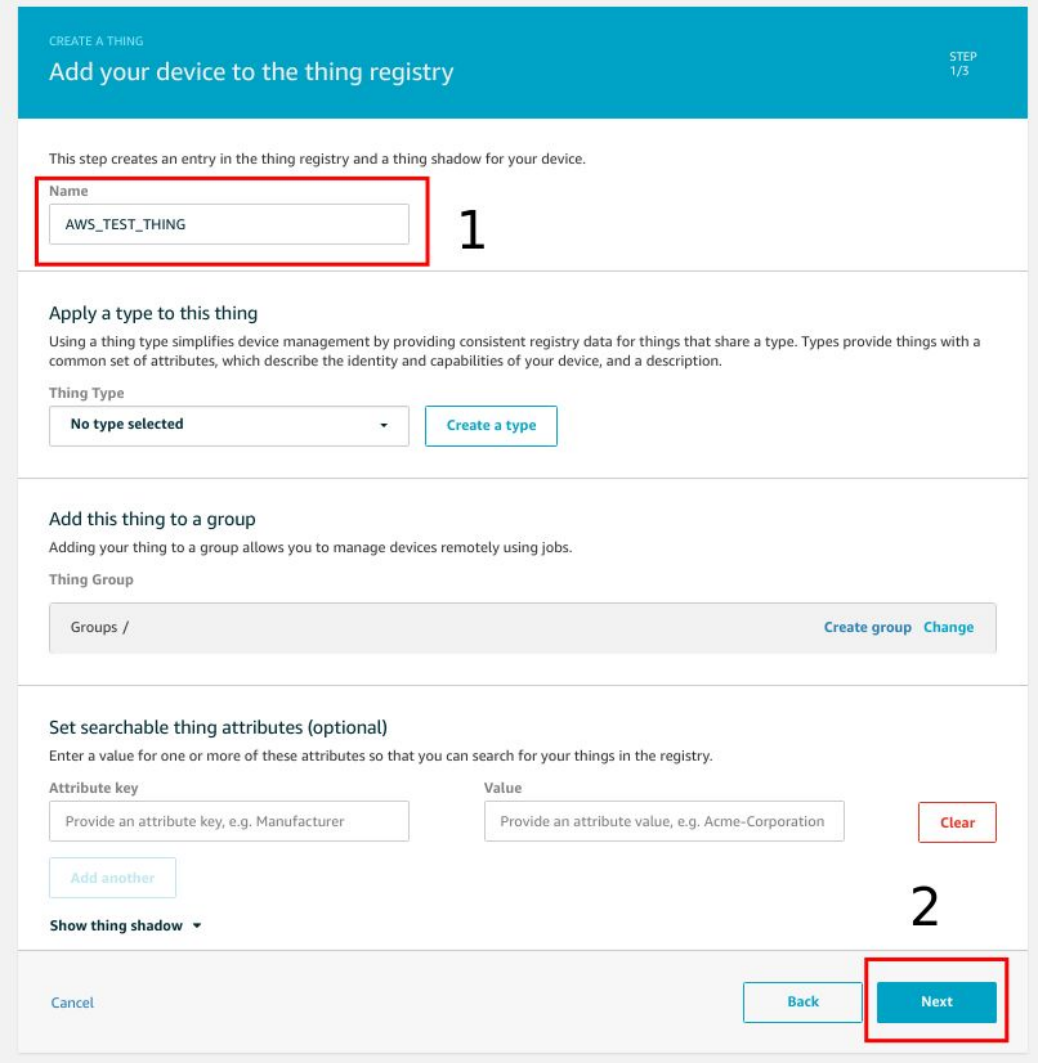


Figure 3. Creating a Thing

2.4 Generate Certificates

After pressing the “Next" button (3.2), select the “Create certificate" option in the next web page that loads. Amazon AWS will then generate a client certificate, private key, and a public key for the “thing" that was just created. Download these certificates (4.1) and save them in a convenient place. Also, be sure to download the Amazon AWS CA certificate (4.2) as this will be needed for the TLS connection.

Next, press the “Activate" button (4.3) to assign the generated certificates to the “thing". Finally, click “Attach a policy" (4.4) to proceed to the next step. Refer to the image below for reference.

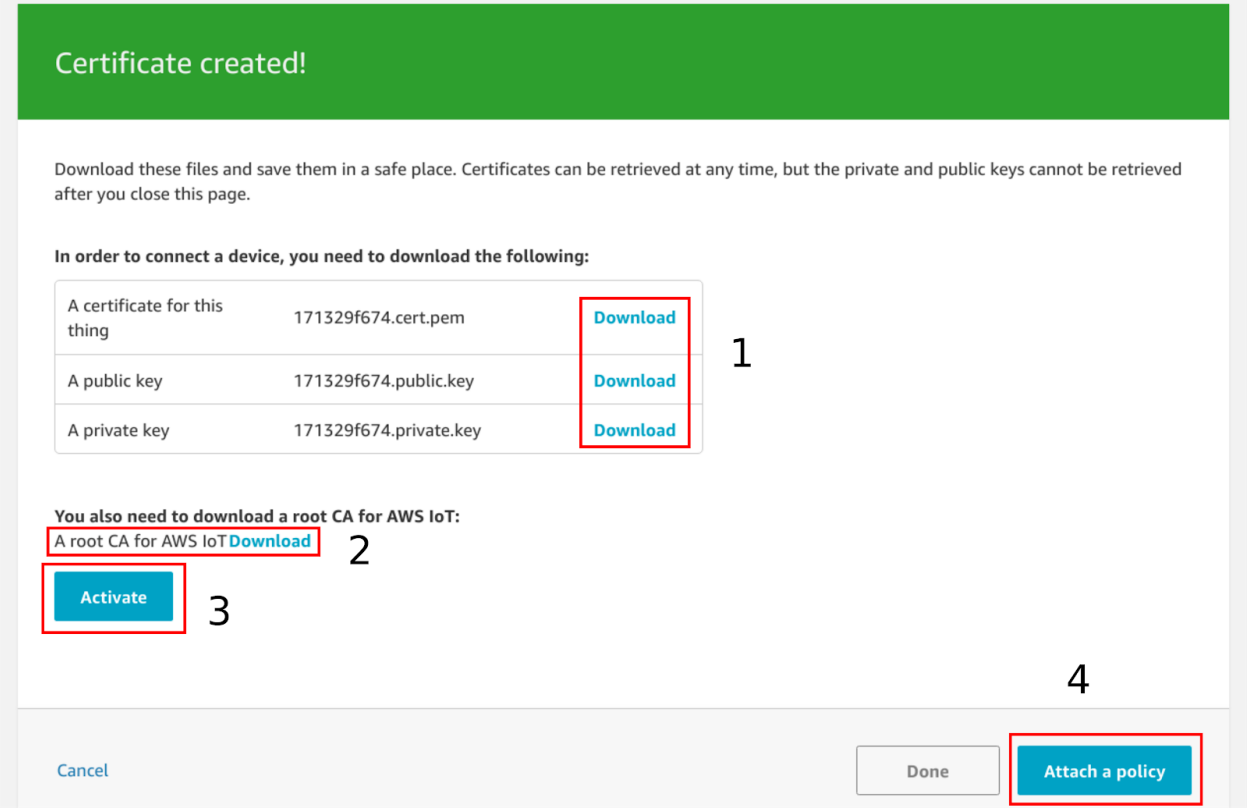


Figure 4. Creating Certificates

Note: The public and private key can only be downloaded from this page. Once this page is navigated from, these files will no longer be available for download.

2.5 Attach the Policy to the "Thing"

After advancing to the next page, attach the policy (5.1) created in Section 2.2 to the “thing" created in Section 2.3 and click “Register Thing” (5.2). Refer to the image below as an example.

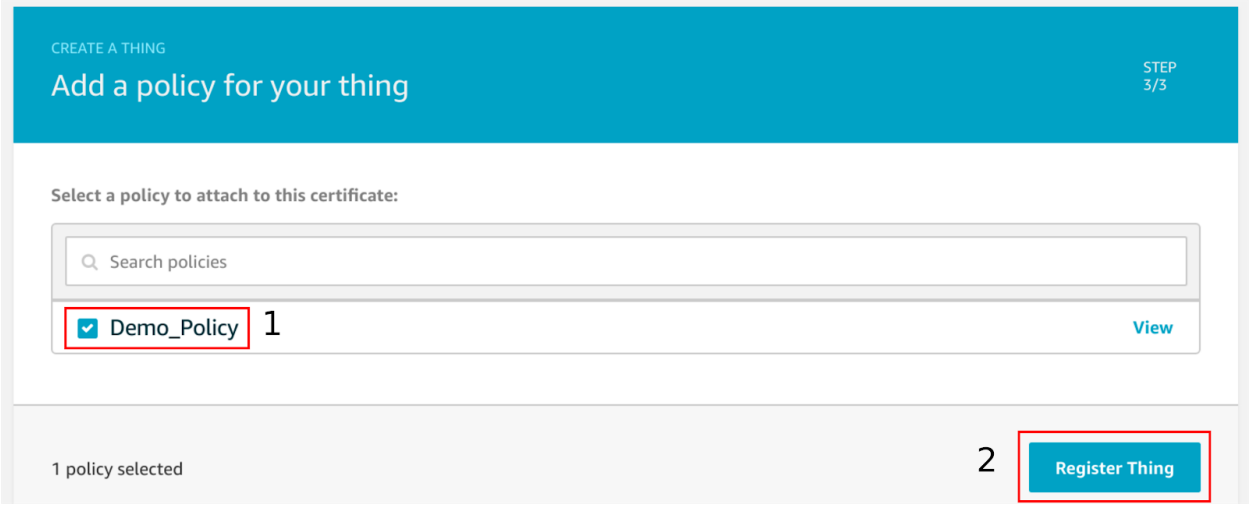


Figure 5. Attaching Policy to Thing

After each of the steps in Section 2 have been completed, proceed to Section 3 for the Skywire configuration instructions.

3. Skywire Configuration

3.1 Prerequisites

This section assumes that the three certificates from AWS IoT Console have been downloaded, the modem is set up and registered on the network, and the CME Errors are set to verbose (**AT+CMEE=2**)

3.2 Uploading Certificates

The first step in the configuration of the Skywire is to upload the certificates needed for the TLS connection. These three certificates are the private key, client certificate and the CA certificate.

The next two sections will lay out instructions for uploading the certificates in a Linux and a Windows environment. Section 3.1.1 details the Linux instructions, while Section 3.1.2 contains the Windows instructions.

3.2.1 Certificate Uploading Using a Linux Environment

First open a terminal and navigate to the directory that contains the certificates that were downloaded in Section 2.4. Type "**ls -l**" to list the contents of the directory on individual lines. Take note of the file sizes of each of the relevant certificates. This information will be needed shortly.

Next, establish a connection to the TC4NAG using a preferred serial console. Once the serial console has been set up properly, issue the following commands to activate SSL and read the contents of the TC4NAG's file system. Indices **0, 1,** and **2** in the following **SSLSECDATA** commands are for the device certificate, rootCA, and private key, respectively.

```
AT#SSLEN=1,1
AT#SSLSECDATA=1,2,0
AT#SSLSECDATA=1,2,1
AT#SSLSECDATA=1,2,2
```

If files exist, the terminal will output the contents, followed by an **OK.** If there isn't a file, the terminal will respond with **No data stored** followed by an **OK.**

If files exists, delete them with the following:

```
AT#SSLSECDATA=1,0,x
```

Where **x** is **0, 1,** or **2** from above.

After clearing out the file system, individually upload each of the three requisite certificates by following the process below:

1. Using the information returned by the "**ls -l**" command, determine the number of bytes for the certificate that is to be uploaded.
2. Issue the AT command below, where **x** is replaced with **0, 1, or 2** depending on if the certificate being uploaded (device certificate, rootCA, or private key, respectively) and **y** is the number of bytes for the certificate being uploaded.

   AT#SSLSECDATA=1,1,x,y

   - Note that the certificates **must** be uploaded in the correct order. For instance, the device certificate must be uploaded in the **0th** location using the command **AT#SSLSECDATA=1,1,0,y**

3. The serial terminal will respond with "**>**", and will wait for data to be entered. This data will be entered using the Linux terminal in step 4.
4. First, disconnect the serial terminal from the "**/dev/ttyUSB***" port. Then, issue the following command in the Linux terminal, where the name of the file is replaced with the name of the certificate in question, and the destination is replaced with the appropriate path to the serial line:

```
cat VeriSign-Class\3-Public-Primary-Certification-Authority-G5.pem > /dev/ttyUSB0
```

   This command will pipe the contents of the certificate to the serial line, which will then be stored in a file on the TC4NAG. Reopen the serial connection with the modem and press **CTRL+Z** to finish uploading.

   The serial line will respond with **OK**

Repeat the four steps listed on the previous page until each of the three files have been uploaded. Once the files are confirmed to have been uploaded successfully, proceed to Section 3.2.

3.2.2 Certificate Uploading Using a Windows Environment

First open the Windows command prompt and navigate to the directory that contains the certificates that were downloaded in Section 2.4. Type "**dir**" to list the contents of the directory on individual lines. Take note of the file sizes of each of the relevant certificates. This information will be needed shortly.

Next, establish a connection to the TC4NAG using a preferred serial console. Once the serial console has been set up properly, issue the following commands to activate SSL and read the contents of the TC4NAG's file system. Indices **0, 1,** and **2** in the following **SSLSECDATA** commands are for the device certificate, rootCA, and private key, respectively.

```
AT#SSLEN=1,1
#SSLSECDATA=1,2,0
AT#SSLSECDATA=1,2,1
AT#SSLSECDATA=1,2,2
```

If files exist, the terminal will output the contents, followed by an **OK.** If there isn't a file, the terminal will respond with **No data stored** followed by an **OK.**

If files exists, delete them with the following:

AT#SSLSECDATA=1,0,x

Where **x** is **0, 1,** or **2** from above.

After clearing out the file system, individually upload each of the three requisite certificates by following the process below:

1. Using the information returned by the "**dir**" command, determine the number of bytes for the certificate that is to be uploaded.
2. Issue the AT command below, where **x** is replaced with **0, 1, or 2** depending on if the certificate being uploaded (device certificate, rootCA, or private key, respectively) and **y** is the number of bytes for the certificate being uploaded.

   AT#SSLSECDATA=1,1,x,y

   - Note that the certificates **must** be uploaded in the correct order. For instance, the device certificate must be uploaded in the **0th** location using the command **AT#SSLSECDATA=1,1,0,y**

3. The serial terminal will respond with "**>**", and will wait for data to be entered. This data will be entered using the Linux terminal in step 4.
4. First, disconnect the serial terminal from the COM port that the modem is connected to. Only one program can have access to the serial line at a time, which is why the terminal emulator must be disconnected during the certificate upload phase.

   Next, issue the following command in the command prompt, where the name of the file is replaced with the name of the certificate being uploaded, and the "COM10" string is replaced with the proper COM port number.

   copy 8da6fe87f3-certificate.pem \\.\COM10

   This command will pipe the contents of the certificate to the serial line, which will then be stored in a file on the TC4NAG. Reopen the serial connection with the modem and press **CTRL+Z** to finish uploading.

   The serial line will respond with **OK.**

Repeat the four steps listed on the previous page until each of the three files have been uploaded. After the files have been successfully uploaded, reattach the serial console to the appropriate COM port. Once finished, proceed to Section 3.2.

3.3 Verifying the Certificate Uploads

To verify that the file uploads were successful, issue the following commands:

```
AT#SSLSECDATA=1,2,0
AT#SSLSECDATA=1,2,1
AT#SSLSECDATA=1,2,2
```

The terminal should respond with something similar to the following:

```
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
.                              .                              .
.                              .                              .
.                              .                              .
4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
OK
```

It is possible for the files to have different formatting and not print exactly like above.

After each of the three files have been uploaded and verified properly, proceed to Section 3.3.

3.4 SSL Profile Configuration

The next step is to configure the SSL connection parameters on the Skywire. To do this, issue the following set of commands:

1. Configure the cipher suite:

   ```
   AT#SSLSECCFG=1,3,2,1
   ```

   This command selects `3` (TLS_RSA_WITH_AES_128_CBC_SHA) as the cipher suite used for the connection, as denoted by the second argument. This can be exchanged with the following

   1 = TLS_RSA_WITH_RC4_128_MD5
   2 = TLS_RSA_WITH_RC4_128_SHA
   3 = TLS_RSA_WITH_AES_128_CBC_SHA
   4 = TLS_RSA_WITH_NULL_SHA
   5 = TLS_RSA_WITH_AES_256_CBC_SHA

   This command also sets auth mode to `2` (Server/Client authentication mode) and cert_format to `1`. The auth mode is why the CA Certificate (server), Certificate

(client), and Private Key (client) are needed, and the cert_format specifies that the certificates are PEM files and not DER files (set to **0** if the certs are DER files)

2. Specify the SSL protocol:

   AT#SSLCFG=1,1,1500,3600,50,1

   This command specifies that the SSL connection should use SSId **1**, PDP context **1** (checkable with **AT+CEREG?**), a packet size of **1500** bits, a timeout of **3600**, a default timeout of **50**, and a time interval of **1**, after which data is sent even if the packet size isn't reached.

   After the SSL configurations have been set properly, proceed to Section 3.4.

3. Activate the PDP Context

   AT&T Firmware:

   AT#SGACT=1,1

   VZW Firmware:

   AT#SGACT=3,1

   This command will open the context **1** for communication

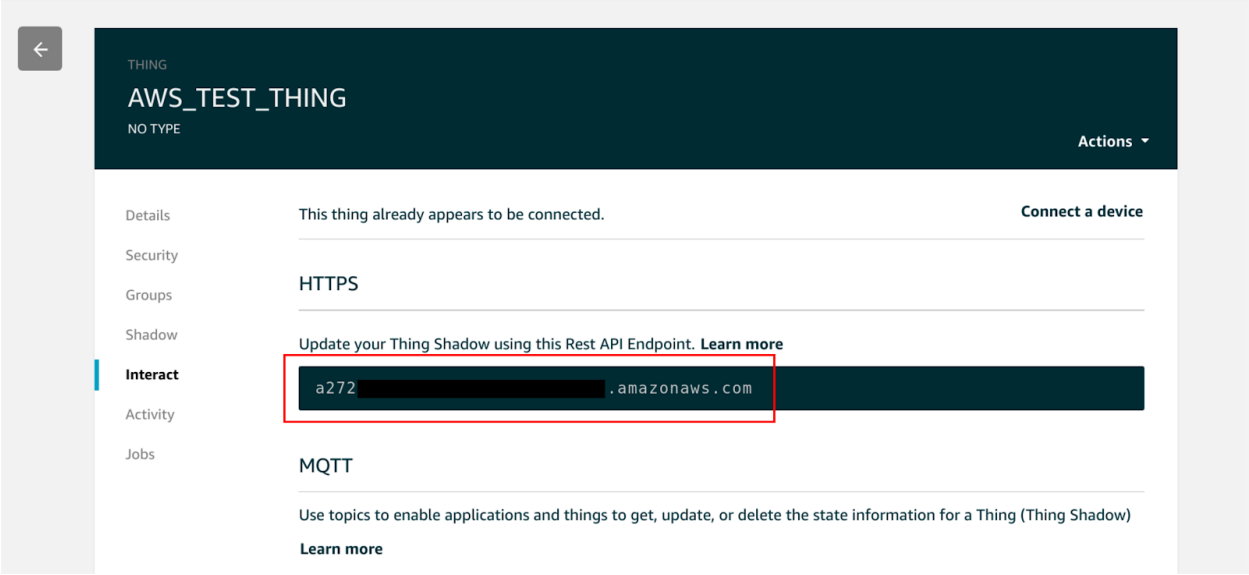
Once the above have been successfully completed, proceed to Section 4.

4. Connect to Amazon AWS

4.1 Opening an SSL Socket

After the Skywire has been configured properly, it is ready to establish a connection to the AWS server.

First, find the endpoint for the “thing” that was created on the AWS website. To do this, navigate to the “Things” page using the menu on the left-hand side of the AWS console page. Click on the “thing” and then navigate to the “Interact” menu. The correct menu should look something like this:



In the image above, the URL for the device endpoint has been enclosed in a red rectangle. Record whatever URL shows up in this page, as it will be needed in the SSL socket connection command.

To open the SSL socket, issue the command below. Make sure to replace the device endpoint in the AT command below with the endpoint that is unique to the AWS account being used for this example.

```
AT#SSLD=1,8443,"a272...amazonaws.com",0,1,60
```

Where **1** specifies the SSL socket to use and **8443** is the port to use, The endpoint URL is the unique URL associated with the AWS account, the **0** specifies the closure type (whether to allow fast restore later), the **1** is the connection mode (**1** = command, **0** = online. Command is easier for terminal use), and the **60** is the maximum allowed TCP inter-packet delay before timeout

After issuing this command, the modem will attempt to connect to the AWS endpoint. If the connection is successful, the modem will return the following:

```
OK
```

The above text indicates that the SSL handshake was successful and that the socket has been opened. If the modem responds with anything other than the above text, issue the following command to get more information about the error:

After the SSL socket has been successfully opened. Proceed to Section 4.2.

4.2 Sending an HTTP Request

After successfully opening an SSL socket, use the following command to send an HTTP request to the AWS endpoint:

```
AT#SSLSEND=1
```

Where "**1**" is the number of the socket used for the SSL connection.

After issuing this AT command, the modem will wait for text to be entered into the terminal. This text can either be pasted or typed into the terminal. After all of the text has been entered or pasted in, press "**CTRL-Z**" to finalize the transmission. See the text below for a sample HTTP POST request using the "**AT#SSLSEND**" command.

Also, take note of the text in red in the table on the next page. Any of the red text indicates key presses, and are not to be typed explicitly. These key press sequences are crucial in order to format the POST command properly. Finally, bold text signifies commands issued to the modem, and text pasted into the terminal.


```
AT#SSLSEND=1
> POST /things/AWS_TEST_THING/shadow HTTP/1.1 "CTRL+M, CTRL+J"
Host: a272...amazonaws.com:8443 "CTRL+M, CTRL+J"
Content-Type: application/json "CTRL+M, CTRL+J"
Content-Length: 116 "CTRL+M, CTRL+J, CTRL+M, CTRL+J"

{"state":{"desired":{"string1":"TLS Connect to AWS","string2":"Using the
built-in stack","string3":"of the TC4NAG"}}}
"CTRL-Z"
OK

SSLSRING: 1,313
```

As can be seen in the text above, the POST command was pasted into the terminal, and the modem responded with "**OK**" indicating that the transmission succeeded. Also note that the "**SSLSRING:1,313**" URC indicates that an HTTP response was received. Section 4.3 will detail how to read this response.

Another important item to note is the "**Content-Length: 116**" line. In this case, the value of "**116**" indicates that 116 bytes of data are being sent through the socket. This helps the endpoint know how many bytes to consider as data. It is crucial to ensure that this number is updated whenever the data JSON is edited.

Note: Properly formatting the POST command can be challenging. Below are a few pointers for correct POST command formatting and entry:

- It is usually impossible to paste the entire POST command all at once. Try pasting the command in line-by-line as opposed to all at once.
- Press "**CTRL+M, CTRL+J**" after each line of the POST command. This sequence enters in a newline and carriage return character after each line.
- Press "**CTRL+M, CTRL+J, CTRL+M, CTRL+J**" after the "**Content-Length: 116**" line. In other words, insert two newline and carriage return sequences after this line.
- Replace the "**AWS_TEST_THING**" identifier with the unique name assigned during the "thing" creation in Section 2.3.
- Replace the AWS endpoint "**a272...amazonaws.com:8443**" with the unique endpoint associated with the Amazon AWS account in use.
- If the contents of the JSON are changed for any reason, the integer argument of the "**Content-Length: 116**" line must be modified.
- In other words, if the JSON is made larger or smaller, the total number of bytes being sent must be recalculated, and the "**Content-Length: x**" line must be updated with the new length.

4.3 Reading an HTTP Response

To read an HTTP response, issue the following command:

AT#SSLRECV=1,313,10

Where "**1**" is replaced by the number of the socket used for the SSL connection, **313** is replaced by the number of bytes to read, and **10** is the timeout.

In the case of the sample POST command in Section 4.2, the HTTP response follows.

```
AT#SSLRECV=1,313,10
SSLRECV: 313
HTTP/1.1 200 OK
content-type: application/json
content-length: 72
date: Wed, 24 Jun 2020 19:59:48 GMT
x-amzn-RequestId: a45f7ef9-13f3-b4f7-04aa-78818b6d5f0c
connection: keep-alive
<message>

OK
```

Note that multiple **SSLRECV** statements can be sent. For instance, the output could look like the following:

```
AT#SSLRECV=1,50,10
SSLRECV: 50
HTTP/1.1 200 OK
content-type: application/

OK
SSLSRING: 1,263
```

```
AT#SSLRECV=1,263,10
json
content-length: 72
date: Wed, 24 Jun 2020 19:59:48 GMT
x-amzn-RequestId: a45f7ef9-13f3-b4f7-04aa-78818b6d5f0c
connection: keep-alive
<message>

OK
```

Note that the max length readable at one time is **1000** bytes.

4.4 Closing an SSL Socket

To close an SSL socket, issue the following command:

AT#SSLH=1

Where "**1**" is replaced with the number of the socket that was used for the SSL connection. The modem should respond with **OK**, unless the connection was already terminated by the server. In this case, the modem will respond with an **ERROR**

5. Working Example: AWS IoT

This section is split up into two subsections. The first shows the preliminary set up of the TSL/SSL socket and the second shows communication with AWS IoT. For AWS IoT, three certs are needed: the rootCA, client certificate, and client private key.

Comments are in // blue

Note: Minicom interprets CRLF differently, which is why the certs and HTTP requests are formatted in an odd manner. The certs and HTTP requests were written into the serial line using the **`cat`** method, described above in Section 3.2.

5.1 Set Up

```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Aug 13 2017, 15:25:34.
Port /dev/ttyUSB0, 09:48:48

Press CTRL-A Z for help on special keys

AT#SSLEN=1,1 // activate SSL
OK
AT#SSLSECDATA=1,2,0 // Check if there is a pre existing cert
#SSLSECDATA: 1,0
-----BEGIN CERTIFICATE-----

MIIDWTCCAkGgAwIBAgIUJ+1lL9xBBn64vmI4yU9JRHZKBGYwDQYJKoZIhvcNAQE
L

BQAwTTFLMEkGA1UECwxCQW1hem9uIFdlYiBTZXJ2aWNlcyBPPUFtYXpvbi5jb20g

SW5jLiBMPVNlYXR0bGUgU1Q9V2FzaGluZ3RvbiBDPVVTMB4XDTIwMDY1
...

-


OK
AT#SSLSECDATA=1,0,0 // delete pre existing cert
OK
AT#SSLSECDATA=1,2,0 // make sure cert was deleted
#SSLSECDATA: 1,0
```



```
No data stored

OK
AT#SSLSECDATA=1,2,1
#SSLSECDATA: 1,1
-----BEGIN CERTIFICATE-----

MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF

ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQ
DExBBbWF6

b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAL
...

-


OK
AT#SSLSECDATA=1,0,1
OK
AT#SSLSECDATA=1,2,1
#SSLSECDATA: 1,1
No data stored

OK
AT#SSLSECDATA=1,2,2
#SSLSECDATA: 1,2
-----BEGIN RSA PRIVATE KEY-----

MIIEpAIBAAKCAQEAxKNu5yyoBIXnnA6r8k2ifSY0zDuFa0RykAZBW1A+AbqHb+za

pFULKW7Qyvlkw5/ItN6RLYvYibWiTpXzSygbZpcuPxl8QTPGnKGxszbYo7sCRJs7

D0w1m15GUORPXRkovU38lIKPHZjY8EuX2UYhRDk5vxBhqRDi3o6U
...
=

-


OK
AT#SSLSECDATA=1,0,2
OK
AT#SSLSECDATA=1,2,2
```



```
#SSLSECDATA: 1,2
No data stored

OK
AT#SSLSECDATA=1,1,0,1220 // Write the client cert (1220 bytes long)
> -----BEGIN CERTIFICATE-----

MIIDWTCCAkGgAwIBAgIUJ+1lL9xBBn64vmI4yU9JRHZKBGYwDQYJKoZIhvcNAQE
L

BQAwTTFLMEkGA1UECwxCQW1hem9uIFdlYiBTZXJ2aWNlcyBPPUFtYXpvbi5jb20g

SW5jLiBMPVNlYXR0bGUgU1Q9V2FzaGluZ3RvbiBDPVVTMB4XDTIwM1
...

-

OK
AT#SSLSECDATA=1,1,1,1188 // Write the rootCA (1188 bytes long)
> -----BEGIN CERTIFICATE-----

MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF

ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQ
DExBBbWF6

b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwML
...
-

OK
AT#SSLSECDATA=1,1,2,1679 // Write the client private key (1679 bytes long)
> -----BEGIN RSA PRIVATE KEY-----

MIIEpAIBAAKCAQEAxKNu5yyoBIXnnA6r8k2ifSY0zDuFa0RykAZBW1A+AbqHb+za

pFULKW7Qyvlkw5/ltN6RLYvYibWiTpXzSygbZpcuPxl8QTPGnKGxszbYo7sCRJs7

D0w1m15GUORPXRkovU38lIKPHZjY8EuX2UYhRDk5vxBhqRDi3U
...
=

-

OK
```



```
AT#SSLSECCFG=1,3,2,1 // Configure connection negotiation settings
OK
AT#SSLCFG=1,1,1500,3600,50,1 // Configure SSL communication settings
OK
AT#SGACT=1,1 // Activate PDP context
#SGACT: 10.66.78.233

OK
```

5.2 Communication

```
// open SSL socket to unique endpoint
AT#SSLD=1,8443,"a1g...amazonaws.com",0,1,60
OK
AT#SSLSEND=1 // Send data over socket
> GET /things/TC4NAG/shadow HTTP/1.1 // HTTP request to get the current shadow
"CTRL+M, CTRL+J,
CTRL+M, CTRL+J, CTRL+Z"

OK

SSLSRING: 1,525 // received 525 bytes
AT#SSLRECV=1,525,10 // read 525 bytes from the socket
#SSLRECV: 525
HTTP/1.1 200 OK
content-type: application/json
content-length: 336
date: Thu, 25 Jun 2020 15:19:18 GMT
x-amzn-RequestId: 37747264-4faa-cf62-5c17-448c929c50c7
connection: keep-alive

{"state":{"desired":{"color":"blue","power":"off"},"reported":{"color":"red","power":"on"},"
delta":{"color":"blue","power":"off"}},"metadata":{"desired":{"color":{"timestamp":15930
96284},"power":{"timestamp":1593096284}},"reported":{"color":{"timestamp":1593032
139},"power":{"timestamp":1593032139}}},"version":3,"timestamp":1593098358}

OK
AT#SSLSEND=1 // send data over socket
// this was input with the cat method described in section 3.2
> POST /things/TC4NAG/shadow HTTP/1.1 // HTTP request to update the shadow
                    Host: a1g...amazonaws.com
```


Content-Type:
application/json

Content-Length: 53

{"state":{"reported":{"color":"blue","power":"off"}}} “CTRL+Z”

OK

SSLSRING: 1,369 // received 369 bytes
AT#SSLRECV=1,369,10 // read 369 bytes from the socket
#SSLRECV: 369
HTTP/1.1 200 OK
content-type: application/json
content-length: 180
date: Thu, 25 Jun 2020 15:21:23 GMT
x-amzn-RequestId: 90c1e0ca-2f23-7f9d-1c33-2de7bd080d39
connection: keep-alive

{"state":{"reported":{"color":"blue","power":"off"}},"metadata":{"reported":{"color":{"time
stamp":1593098483},"power":{"timestamp":1593098483}}},"version":4,"timestamp":15
93098483}

OK
AT#SSLSEND=1 // send data over the socket
> GET /things/TC4NAG/shadow HTTP/1.1 // HTTP request to get the current shadow

"CTRL+M, CTRL+J,

CTRL+M, CTRL+J, CTRL+Z"

OK

SSLSRING: 1,488 // received 488 bytes
AT#SSLRECV=1,488,10 // read 488 bytes from the socket
#SSLRECV: 488
HTTP/1.1 200 OK
content-type: application/json
content-length: 299
date: Thu, 25 Jun 2020 15:21:42 GMT
x-amzn-RequestId: 0e0a6186-b083-9ac0-4bec-97c3386e925c
connection: keep-alive

{"state":{"desired":{"color":"blue","power":"off"},"reported":{"color":"blue","power":"off"}}
,"metadata":{"desired":{"color":{"timestamp":1593096284},"power":{"timestamp":15930


```
96284}},"reported":{"color":{"timestamp":1593098483},"power":{"timestamp":1593098
483}}},"version":4,"timestamp":1593098502}

OK
AT#SSLH=1 // close the SSL socket
OK
AT#SSLEN=1,0 // deactivate SSL
OK
AT#SGACT=1,0 // deactivate PDP context
OK
```


6. Troubleshooting

6.1 HTTP Response Codes

6.1.1 403 Forbidden

If a connection can be established, but the AWS response to the "**GET**" command is "**403 Forbidden**", make sure that the current AWS policy is set to allow all IoT actions. This can be done through the AWS IoT Console.

6.1.2 400 Bad Request

If a connection can be established, but the AWS response to the "**GET**" or "**POST**" command is "**400 Bad Request**", make sure the syntax of the "**GET**" or "**POST**" command is correct.

6.1.3 404 Not Found

If a connection can be established, but the AWS response to the "**GET**" or "**POST**" command is "**404 Not Found**", the HTTP request cannot be fulfilled because the resource does not exist. For instance, there might be a typo, like

```
GET /things/TCNAG/shadow HTTP/1.1
```

instead of

```
GET /things/TC4NAG/shadow HTTP/1.1
```

6.2 Verify Credentials

If for some reason the credentials for the AWS connection do not work, OpenSSL can be used to check their validity. This process is helpful for narrowing down the source of the connection issue.

6.2.1 Testing AWS Credentials using OpenSSL

To test credentials with OpenSSL, first ensure that OpenSSL is properly installed on a Linux or Windows system. Next, navigate to the directory that contains the certificates that are being tested. Issue the following command to attempt a connection to AWS:

```
openssl s_client -servername a272...amazonaws.com -connect
a272...amazonaws.com:8443 -CAfile VeriSign-Class\
3-Public-Primary-Certification-Authority-G5.pem -cert
8da6fe87f3-certificate.pem.crt -key 8da6fe87f3-private.pem.key -certform PEM
-keyform PEM
```

Be sure to replace any of the text in bold with unique certificate names, and the unique AWS endpoint URL.

If the connection is successful, the terminal should respond with "CONNECTED" followed by some information about the connection, including the server certificate. To further test the connection, type the following command, followed by the sequence "**CTRL+M, CTRL+J**":

GET /things/AWS_TEST_THING/shadow HTTP/1.1

Remember to replace the "**AWS_TEST_THING**" string with the unique "thing" name assigned in Section 2.3.

The server should respond with something similar to:

```
HTTP/1.1 200 OK
content-type: application/json
content-length: 385
date: Tue, 24 Jul 2018 22:35:57 GMT
x-amzn-RequestId: 6ae1eb60-6910-9a7b-e8b0-6c6d2904e239
connection: keep-alive

{"state":{"desired":{"string1":"TLS Connect to AWS","string2":"Using
the built-in stack","string3":"of the
TC4NAG"},"delta":{"string1":"TLS Connect to AWS","string2":"Using the
built-in stack","string3":"of the
TC4NAG"}},"metadata":{"desired":{"string1":{"timestamp":1532463231},"
string2":{"timestamp":1532463231},"string3":{"timestamp":1532463231}}
},"version":19,"timestamp":1532471757}
```

If a valid connection can be established, then it is safe to say that the certificates are indeed valid, and thus are not the source of the problem.

