# HOWTO: CFHTTP and self-signed 4k certs.

**tl;dr: skip to Here's How, 2048 bit keys, or troubleshooting.**

If you're trying to access an SSL site with a self-signed certificate (or any certificate without a chain it knows about) using your web browser or a program like cURL you get a prompt to accept it anyway and you're on your merry way.

ColdFusion uses the underlying Java libraries to handle handle networking calls like <CFHTTP> and there's no argument to tell it to ignore lack of a valid trust chain during a request to an HTTPS URL.

Java, like your browser, has a list of trusted root certificates which gets updated once-in-a-while when new signing authorities come about or existing certificates are invalidated or compromised.

**Sidebar on Certificate Signing**

Accessing a URI over HTTPS vs HTTP does two things for you:

a. Encrypts the information travelling from the server to your client computer or device.
   An SSL (or TLS) starts out using asymetric communication - the server sends the client its public key which the client uses to begin the encryption negotiation. The two machines then exchange private keys and the rest of the session uses symmetric encryption which is much faster and less CPU intensive. If it's actually the two machines at the end exchanging their keys, the communication is safe.
b. Authenticates the identity of the server.
   i. When you type https://google.com into your browser the communication will be encrypted, but how do you know you are actually interacting with Googl's servers and not some rouge government or ISP's? Well, the certificate the server sends you is "signed" by a Certificate Authority - a company entrusted with verifying that the certificate you receive for google.com was actually granted to Google and not someone else claiming to be "Google". It's like having a photo ID - the government has validated the connection between your face and your name, etc.
   ii. You could also have a student ID - the school has verified your face goes with your name, but they probably don't have as rigorous procedures, etc. This is like a certificate signed by an unknown CA. For example your company could sign certificates for their intranet sites.
   iii. The least secure is a self-signed certificate - "I am who *I* say I am." You

> have to take their word for it, that's all. If you trust them and are sure there is noone in between you and them playing "telephone" you can accept their certificate and your communications will be encrypted with "whoever" that is on the other end.

There are valid situations where you will need to communicate with and trust a site with a self-signed certificate.

So, the way you get Java and therefore ColdFusion to accept a self-signed certificate is to get the certificate and import it into its keystore using a command line Java tool called `keytool`. Once imported, Java will trust that certificate and you can connect to an HTTPS URL which uses it.

## Here's how:

Visit the HTTPS URL in question in your web browser and export the public key:
(Windows, but Mac and Linux are very similar as are different browsers)

1. Browse to the SSL website
2. Double click the Lock icon in the status bar
3. Click the Details Tab
4. Click the button Copy to File...
5. Click Next in the Export Wizard
6. Choose the first option (default) for DER encoded
7. Click Next
8. Choose the folder/directory to save the certificate
9. Type any filename for certificate such as mycert.cer
10. Click Next, Click Finish

Alternately, you can use openssl to retrieve the certificate like so:
`openssl s_client -connect servername:portNumber`

Copy the lines `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` and everything in between into a text file and name it with a .cer extension.

Copy the certificate to your /jre/lib/security directory. For Adobe ColdFusion it's located here:
Server Configuration:
*cf_root*/runtime/jre/lib/security/

Multiserver/J2EE on JRun 4 Configuration:

*jrun_root*/jre/lib/security/

And if you're using your own Sun JDK installation:
*jdk_root*/jre/lib/security/

Backup the file `cacerts` in that directory.

Import the certificate into the default Java certificate store.
Adobe's KB article explains. As do Steven Erat, australiansearchengine, Brandon Purcell, and Ramesh Sabeti

Here's where you use Java's `keytool` to do the fancy. Now, keytool lives at /jre/bin/keytool

The base command you need is:

```
keytool -import -v -keystore cacerts -alias someServer-cert -file
someServerCertFile.cer -storepass changeit
```

And here's how I did it: Navigate to the same /runtime/jre/lib/security/ directory where you copied the certificate (On Windows using my own JDK install that path was C:\Program Files\Java\jdk1.6.0_27\jre\lib\security\) in your terminal / command prompt and execute the following command:

```
..\..\bin\keytool -import -v -keystore cacerts -alias ServerName
-file ServerName.der -storepass changeit
```

And yes, the password is actually "changeit".

You'll get a prompt saying the certificate is not trusted and do you want to import it, type "yes" and hit return.

To confirm it imported successfully, run the following command and look for your alias in the resulting output.

```
..\..\bin\keytool -list -v -keystore cacerts -storepass changeit
```

Restart ColdFusion.
Now you can use CFHTTP to connect to the server that uses a self-signed certificate for its SSL connection.

## Concerning 2048 bit or longer SSL certificate keys.

Many SSL certificates are issued with lengths of 2048, 4096 or more bits. Your version of Java

probably does not support keys of 256 bits or greater. To enable stronger encryption on ColdFusion in general you need to install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6. Download the .zip file. *(If you are legally allowed to!)* And replace the existing `US_export_policy.jar` and `local_policy.jar` files in the same /jre/lib/security/ directory the `cacerts` file was in above. Dan Gaspar explains.

Restart ColdFusion.

Now in addition to allowing CFHTTP to use 2048 bit or greater SSL certificates, ColdFusion's encryption functions can now create and use keys greater than 128 bits. (Which you should be doing anyway!)

*And if you really want to dig into encryption in ColdFusion this Adobe Tech Note gets into some good detail: Strong encryption in ColdFusion MX 7. The LiveDoc on encrypt() for ColdFusion 9 is also worth reading.*

## Troubleshooting.

Things might not always go as planned. Here are some things to try.

Use `OpenSSL` from your client (ColdFusion server) to connect to the remote server and verify you can access it and that it is sending a certificate.

```
openssl s_client -connect servername:portNumber
```

(OpenSSL should  be included on your MacOS or Linux installation. You can get the source at the link above and precompiled Windows binaries here.)

Use `cURL` from your client (ColdFusion server) to connect to the remote server and verify you can make a successful request to the desired URL.

```
curl --insecure --user username:password https://hostname:port/path/
```

See the cURL documentation if you need to pass custom user agents or headers for authentication.

An example would be something like this:

```
curl --insecure --user admin:password
https://servername:443/api/products/ --header "X-APPID: 1234567890"
--user-agent "some special user agent like a client id."
```

Turn on SSL debugging in Java to see details of how the Java behind CFHTTP is attempting to

negotiate the connection.

Adobe Says: “Add the `-Djavax.net.debug=all` switch to the JVM arguments section of the Java and JVM section of the ColdFusion Administrator or add the switch directly to the jvm.config file and restart ColdFusion.”

Attempt the connection and look at the output in *cf_root*/runtime/logs/coldfusion-out.log

The part you’re looking for begins: `Starting HTTP request {URL='https://`*hostname*`:`*port*`/<etc>', method='<get/post/etc>'}`

The end.