Assignment 3

Due Friday, Dec 2nd at 11:59pm

Assignments are to be completed individually. We generally expect you to make an honest effort searching around online before contacting course staff with technical questions (`stackexchange.com` and `crypto.stackexchange.com` are great resources btw).

Submission Instructions

Place each answer in a clearly named file (*e.g.,* `Q1.txt`). Answers may be in `txt`, `pdf` or `doc/docx`. To avoid problems and potential mark penalties, do not use other formats. Place these answers along with all accompanying files in a `.zip` and submit via OWL by the due date. As per the course late policy, assignments can be uploaded only up to 48 hours past the due date.

1. [30 marks] Part 1. Setting up a TLS Enabled Webserver

In this question you will setup a TLS enabled webserver with a goal of serving a basic webpage over a TLS connection. To do this we're going to have to fake a couple of things since we don't want to make you pay for an actual domain name or web hosting.
Use the following setup guide to get your basic TLS enabled server up and running:
http://essex.cc/teaching/Information-Security/assignments/server-setup.html

Deliverables. Places all files in a directory `Q1`.

(a) A screen capture of Chrome showing your website's URL, the default webpage, and the green padlock icon. Name this file Q1.png (or Q1.jpg),

(b) Download the testssl.sh script from https://testssl.sh/. Run it on your website and pipe the output as follows:

```
./testssl.sh [your website domain] > Q1.txt
```

Solution. Solutions will vary, but mainly we're looking for server being up and running and all the not ok's have been fixed

2. [30 marks] Part 2. Tune the TLS Configuration

Not all TLS configurations are created equal. In class we looked at good TLS configs (*e.g.,* nchangfong.com) and not-so-good configs (*e.g.,* uwo.ca). The Apache default is more in the not-so-good direction, so in this question you will improve your website's TLS configuration. In

Question 1 you may have noticed that the result of `testssl.sh` found several aspects of the configuration it described as "NOT ok."

Objective: Configure your TLS settings so `testssl.sh` no longer finds anything that it considers "NOT ok." In addition configure the settings to only offer ciphersuites that meet the following criteria (some of them may already be satisfied by the default config):

- Key exchange shall be only ECDHE or DHE
- Signature method shall be 2048-bit RSA
- Block cipher shall be AES-CBC or AES-GCM
- Hash shall be in the SHA family
- Does not offer any SSL versions

Deliverables. Places all files in a directory `Q2`.

(a) Re-run the testssl.sh script on your website meeting the above objectives and put the result into a file Q2.txt: `./testssl.sh [your website domain] > Q2.txt`

(b) A copy of your modified `ssl.conf` file.

Solution. Depending on which server (and version) you used, the default TLS settings may have a number of shortcomings. The mains ones to fix were:

(a) Disable SSL v3 (and earlier) support

(b) Disable HTTP compression to counter BREACH attack

(c) Disable all non high-grade cipher suites (including 3DES and RC4)

(d) Make sure signatures use SHA256 and 2048-bit (or greater) RSA

Tip: I would also suggest disabling ciphersuites with RSA kx, but we won't mark for that.

3. [20 marks] Negotiate This

There are two main ways a client and server can agree on a TLS ciphersuite. One is client preference: the client has an ordered lists of ciphersuite preferences, and the server picks the ciphersuite that is *most* preferred by the client, and that is supported by the server. The other method is server preference: the server has a list of ciphersuite preferences and the picks the ciphersuite that is *most* preferred by the server, and that is supported by the client.

Deliverables. Place a file Q3 in a directory `Q3`. Thinking about ciphersuite preferences of the following clients and servers, use SSL Labs Server Test and SSL Labs Client Test to help you answer the following questions. For each of the following which ciphersuite would be selected if:

(a) Chrome 45 connected to `uwo.ca` and the server picks

Solution. You can use the SSL test page `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` after.

(b) Chrome 45 connected to `uwo.ca` and the client picks

Solution. `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` before and after

(c) Chrome 45 connected to `nchangfong.com` and the client picks

Solution. `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`

(d) IE 6 / XP connected to `nchangfong.com` and the server picks

Solution. `TLS_RSA_WITH_3DES_EDE_CBC_SHA`

Solution. To see what ciphersuites Chrome 45 supports at https://www.ssllabs.com/ssltest/viewClient.html?name=Chrome&version=45

4. [20 marks] TLS Key Kraziness

As we've talked about in class a single TLS connection involves *a lot* of keys. In this question you will enumerate *all* of the keys used in a single TLS connection to *google.ca*. This includes all public, private, and secret keys exchanged or generated during a TLS handshake.

Deliverables. Place a file Q4 in a directory `Q4`. First list the ciphersuite your browser connected under. The file then should have 3 columns: a description of the key (*e.g.,* Google's public ECDHE key), the name of the entities who know the key (*e.g.,* everyone), and a sentence or two describing the purpose of the key (*e.g.,* used for public key agreement).

Solution. Well let's see. All together there should be sixteen (16) keys that come in to play:

(a) GeoTrust Global CA signing key, used to self-sign GeoTrust's certificate, and to sign Google Internet Authority G2's certificate. Only Geotrust knows it.

(b) GeoTrust Global CA verification key, used to verify the signatures on GeoTrust's certificate, and Google Internet Authority G2's certificate. Everyone knows it.

(c) Google Internet Authority G2's signing key, used to sign google.ca's certificate. Only Google Internet Authority G2 knows it.

(d) Google Internet Authority G2's verification key, used to verify signature on google.ca's certificate. Everyone knows it.

(e) google.ca's signing key, used to sign google.ca's ephemeral elliptic-curve Diffie-Hellman (ECDHE) public key. Only google.ca knows it.

(f) gogole.ca's verification key, used to verify the signature on google.ca's ECDHE public key. Everyone knows it.

(g) google.ca's ECDHE public key, sent to client for key agreement. Everyone knows it.

(h) google.ca's ECDHE private key, used to generate the corresponding public key, and to apply to client's public key to form shared secret. Only google.ca knows it.

(i) Client's ECDHE public key, sent to google.ca for key agreement. Everyone knows it.

(j) Client's ECDHE private key, used to generate the corresponding public key, and to apply to google.ca's public key to form shared secret. Only client knows it.

(k) Diffie-Hellman shared secret. Result of ECDHE key exhcange. Also called the pre-master secret. Used to derive master secret. Only client and server know it.

(l) Master secret. Derived from pre-master secret and client/server random values, and used to derive symmetric keys. Only client and server know it.

(m) Client-write symmetric encryption key. Used by the client to encrypt messages. Used by the server to decrypt messages. Only client and server know it.

(n) Server-write symmetric encryption key. Used by the server to encrypt messages. Used by the client to decrypt messages. Only client and server know it.

(o) Client-write MAC key. Used by client to generate a MAC tag. Used by server to verify MAC tag. Only client and server know it.

(p) Server-write MAC key. Used by server to generate a MAC tag. Used by client to verify aMAC tag. Only client and server know it.

TIP: If you ever wanted to setup TLS enabled web server for real

If you're ever interested in setting up a TLS enabled webserver for real (i.e., outside the scope of the course), I recommend the Digital Ocean micro hosting, private domain registration with Hover, and free certificates with Let's Encrypt via the wonderful Certbot TLS configuration utility. You can get an A grade on SSLTest pretty much out of the box. The whole setup costs around $6 USD/month, which is the cheapest option I'm currently aware of.