Making progress

OK after success with our first real world TLS1.3 website tls13.cloudfare.com, we ran into problems with our next website www.miracl.com, due to fragmentation issues.

These were relatively easily fixed. The server response was fragmented into 4 fragments, each of which was encrypted in AES-GCM mode using the server handshake key and IV negotiated earlier. The same key would be used for each fragment, but what about the IV? It is I know a huge cryptographic mistake to re-use the same key and IV for more than one plaintext. So I deduced the IV must be modified for each subsequent fragment in some way. But how? I tried a few things that didn't work before I went back and read the RFC.

And there it is, section 5.3 of RFC8446. You basically XOR the fragment number with the IV. Wasted a lot of time by not simply checking with the RFC. Duh.

So that is now two TLS1.3 supporting websites our client works with. Lets try another one, www.bbc.co.uk.

The first attempt failed, but it was easy to see that this was because we were offering support for cipher suites that used AES-256 and SHA384, although it wasn't our first choice in our clientHello message – we were suggesting AES-128 and SHA256. But nonetheless the BBC website in its serverHello picked the more secure option, and we hadn't fully implemented this mode. Relatively easily fixed. Next issue was that one of the certificates in the certificate chain used the secp384r1 elliptic curve, and we hadn't provided support for this. Again easily fixed.

So the first website was hard, the second was easier, and the third we got working quite quickly. So I sense that our implementation is becoming more robust. The big break-through will be when I try a new website and it works first time without any modification!

Next up I tried mail.protonmail.com. Now they should value security! And this time it worked almost immediately without further modification. OK I had to increase the sizes of a few arrays, and I had to provide support for 4096 bit RSA keys, but basically it worked first time. It was interesting to note that although the server certificate used a hyper-secure 4096-bit RSA key, and so did the Swiss root CA, the intermediate certificate used only a 2048-bit RSA key. Potentially a weak link?

Certificate Chains

So far all of the certificate chains we have encountered that have been provided by the servers, have all been of length 3, which seems to have become a bit of a standard. The root Certificate Authority, an intermediate Certificate and then the actual Server certificate. The idea of the Intermediate certificate is to limit the exposure of the root CA, and this is definitely a good idea. However a chain of length greater than 3 seems pointless, and may involve weak links with different lifetimes, and anyway large certificate chains are a problem for small clients.

In the abstract of a recent paper I read "*Certificates ensure the authenticity of user's public keys, however their overhead (e.g. certificate chain) might be too costly for some IOT systems like aerial drones*". The article goes on to look into lightweight non-PKI solutions. However the point is well made: Large certificate chains are a problem.

Another cause of bulky certificate chains is the paranoid use of bulky 4096-bit RSA keys. A 521-bit ECC key is more secure and 8 times smaller.

Therefore it may be an idea to (a) limit our support to certificate chains of total length 3, and (b) insist on an upper limit on the certificate chain size. This could start with choosing a lean root CA. Or establishing our own Certificate Authority that guarantees that certificate chains are of length 3, and that their size has a fixed upper limit (maybe 4096 bytes?). In fact CAs can already be configured to implement constraints which limit chain lengths.

Simpler Testing

The client program has now been made a bit easier to use. Simply

```
client mail.protonmail.com
```

to force a TLS 1.3 connection with any website. It will abort if the website does not support TLS 1.3. Note that at this stage the tool is still very fragile, and many websites are still using version 1.2. Build instructions have been improved – see the readme file – and it should be simply enough to build the tool yourself on any Linux system. I use Ubuntu in WSL 2.0 in Windows.

Just tried

```
client www.google.com
```

and it worked first time! With a little work this could make a useful tool for analysing the security of websites