Establishing an HTTPS Connection

Our modern world is heavily reliant on the idea of secure internet communication. Everybody is so quick to utilize technologies such as online banking, online shopping, and text messaging. Given that our network traffic has to travel through countless servers, owned by strangers and governments, over physical wires, how can we be confident that our secrets stay secret? The short and simple answer is a protocol known as HTTPS, a secure version of HTTP. This 'meta-protocol' describes methods which allow security using a wide variety of cryptographic constructs. In this paper, we will briefly go over some of the highlights of establishing an HTTPS connection.

The first step in the process is known as the 'client hello.' In this stage, the client sends a few key pieces of information to the server it wants to talk with. The two most important pieces of information are 1) random bytes, and 2) acceptable cipher suites. The random bytes will be used later to ensure the master key is chosen securely. The acceptable cipher suites list will be considered by the server and used to pick the final cipher. After the 'client hello' has been received, the server will respond with three sequential packets. Firstly, a 'server hello' message which contains 3 main things: more random bytes, the chosen cipher suite, and a session ID which can be used in the future to skip past the slow handshake process.

The second packet sent by the server will be a certificate chain which can be used to verify the server is who it says it is as well as providing the server's RSA public key. The client is forced to do some simple RSA calculations (just modular exponentiation) and hashing in order to verify that the certificates are valid. It is worth noting that the certificate chain can only be validated due to some level of trust in root authorities. Without this root trust, it is impossible to have any confidence in internet safety. After verification has been checked, the client now can trust that Amazon's public key that was sent is Amazon's actual public key. After the second packet has been sent, the server will send its final message indicating that the client's identity will not be verified.

Once the client has been provided with Amazon's public RSA values, it can generate a random 'pre-master secret'. This value will be encrypted with Amazon's public key, then sent over the wire. Once both sides have the pre-master key, the master key can be found by combining the pre-master secret with the random bytes sent earlier in the process. The combination process typically involves an HMAC of the pre-master and random bytes. Once the master key has been obtained by both parties, they are able to generate keys of the appropriate size for the various crypto constructs using a 'Pseudo-Random Function' that acts as a secure generator given a secret seed.

Once all the keys have been derived, the server sends an encrypted message back to the client proving that it was able to decrypt the client's previous messages and encrypt its own messages correctly. After all this takes place, the application layer takes over and sends data securely back and forth across the wire.

Questions:
What is the process for adding and removing cipher suites from the specification? Who decides which options are added or deprecated?

What is the performance cost of this handshake? It seems like it would greatly increase latency on the initial request due to the handshake requiring multiple back/forths before the application layer can start sending data.

How is the HMAC which is using sha-1 and MD5 still secure? I would think when both of its component algorithms are not secure, it would not be secure.