Question 1 ***DNS Walkthrough*** **(15 min)**

Your computer sends a DNS request for "www.google.com"

(a) Assume the DNS resolver receives back the following reply:

```
com. NS a.gtld-servers.net
a.gtld-servers.net A 192.5.6.30
```

Describe what this reply means and where the DNS resolver would look next.

Solution: The IP address for "www.google.com" is not known. However, "a.gtld-servers.net" is a name server for .com, and that is where the resolver should ask next, at the IP address 192.5.6.30.

(b) If an off-path adversary wants to poison the DNS cache, what values does the adversary need to guess?

Solution: The adversary will need to guess the identification number (16 bits). Some resolvers even randomize source ports.

The reason an off-path attack is because the ID (and port numbers) have to match exactly, but once the legitimate reply reaches the resolver and is cached, the server is no longer vulnerable to the poisoning attempts.

(c) What are some issues with using TLS to secure DNS?

Solution: DNS is designed to be lightweight and TLS will add lots of overhead.

TLS also works very poorly with DNS caching, which is required for scalability.

But the biggest issue is that we do not know which name servers to trust and TLS provides no protection against that. This is a fundamental difference between object security and channel security.

Note however, there is a DNS-over-HTTPS protocol that can be used today.

Question 2 ***TLS threats*** **(15 min)**

An attacker is trying to attack the company Boogle and its users. Assume that users always visit Boogle's website with an HTTPS connection, using ephemeral Diffie-Hellman. You should also assume that Boogle does not use certificate pinning. The attacker may have one of three possible goals:

1. Impersonate the Boogle web server to a user
2. Discover some of the plaintext of data sent during a past connection between a user and Boogle's website
3. Replay data that a user previously sent to the Boogle server over a prior HTTPS connection

For each of the following scenarios, describe if and how the attacker can achieve each goal.

(a) The attacker obtains a copy of Boogle's certificate.

> **Solution:** None of the above. The certificate is public. Anyone can obtain a copy simply by connecting to Boogle's webserver. So learning the certificate doesn't help the attacker.

(b) The attacker obtains the private key of a certificate authority trusted by users of Boogle.

> **Solution:**
>
> The attacker can impersonate the Boogle web server to a user. The attacker can't decrypt past data. First, Boogle's private key is used in the protocol, not the CA's. Second, Diffie–Hellman provides "forward-secrecy" (as in part (c)), and so the attacker could not decrypt it regardless.
>
> The CA's private key can be used for creating bogus certificates, which can be used to fool the client into thinking it is talking to Boogle.
>
> Replays aren't possible, due to the nonces in the TLS handshake.

(c) The attacker obtains the private key corresponding to an old certificate used by Boogle's server during a past connection between a victim and Boogle's server. Assume that this old certificate has been revoked and is no longer valid. Note that the attacker does not have the private key corresponding to current certificate.

Solution: None. Unless the attacker can figure out either a or b, the attacker will not be able to decrypt the data of past connections.

This can't be used to impersonate a Boogle server because the attacker doesn't have a fresh valid certificate corresponding to the stolen private key, and can't use the previous certificate for that key because it's been revoked.

Question 3 ***TLS protocol details*** **(20 min)**

Depicted below is a typical instance of a TLS handshake.

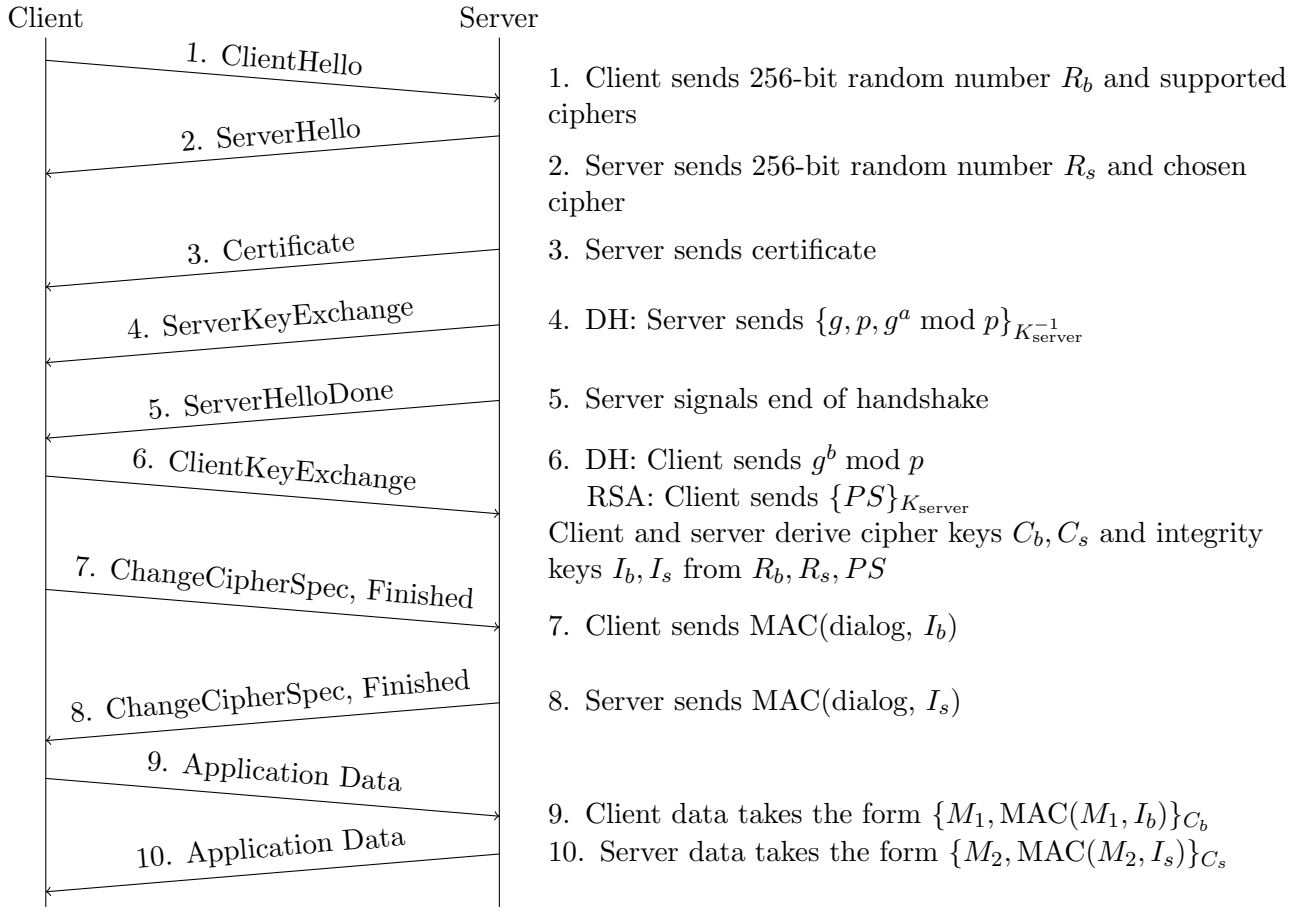

Figure 1: TLS 1.2 Key Exchange

(a) What is the purpose of the *client random* and *server random* fields?

> **Solution:** Because the master secret depends on these, they act as nonces that prevent replay attacks.

(b) ClientHello and ServerHello are not encrypted or authenticated. Explain why a man-in-the-middle cannot exploit this. (Consider both the Diffie-Hellman and RSA case.)

> **Solution:** The use of either public key encryption (RSA handshake) or a Diffie-Hellman exchange prevents an eavesdropper from learning the Premaster Secret.
>
> A MITM attacker who alters any of the values will be exposed, as follows.
>
> For the RSA handshake case, the MITM will be unable to read the Premaster

Secret sent by the client because it is encrypted using the server's public key. When the client and server exchange MACs over the the handshake dialog, the MITM will be unable to compute the correct MACs for their altered dialog because they will not know the corresponding integrity keys derived from the Master secret.

For the Diffie-Hellman case, the MITM will be unable to alter the value of $g^a \mod p$, because the client requires that the value have a correct signature using the server's public key. Because the MITM cannot alter the value, they cannot substitute $g^{a'} \mod p$ for which they know a'. Without knowledge of the exponent, the MITM cannot compute $g^{ab} \mod p$ to obtain the Premaster Secret.

(c) Note that in the TLS protocol presented above, there are two cipher keys C_b and C_s. One key is used only by the client, and the other is used only by the server. Likewise, there are two integrity keys I_b and I_s. Alice proposes that both the server and the client should simply share one cipher key C and one integrity key I. Why might this be a bad idea?

Solution: Vulnerable to **reflection** attacks: a man-in-the-middle can send a client's application data back to them. It will still verify the appropriate checks, but the user will think that this is the response from the server. Likewise, an attacker can reflect a server's response back to the client. Note that due to the existence of sequence numbers in the TLS specification, the actual attack would be a little more complicated.

(d) The protocol given above is a simplified form of what actually happens. After step 8 (ChangeCipherSpec), the protocol as described above is still vulnerable. What is the vulnerability and how could you fix this?

Solution: An attacker can perform a **replay** attack, where they send the same application data twice. The solution is to add sequence numbers (which is what the actual TLS specification does, with some extra details involved).