18.3. Show how someone who knows both Alice's and Bob's public encryption keys (and neither side's private key) can construct an entire IKE exchange based on public encryption keys that appears to be between Alice and Bob.

Let Trudy be the person attempting to construct an IKE exchange that looks like it's between Alice and Bob. For all IKE public encryption variants, the following applies. Trudy chooses Diffie-Hellman numbers $a$ and $b$ for each side, and nonces for each side. The proof of identity is a function of the other side's nonce (which ordinarily would require knowledge of one's private key since it is transmitted encrypted with the public key, but in this case Trudy has chosen the nonce and therefore knows it), the Diffie-Hellman values, and the cookies, all of which Trudy knows. Trudy can also compute the session keys, since she knows all the inputs, including $g^{ab} \mod p$, since she knows both $a$ and $b$.

18.6. For Photuris, and for each of the Phase 1 IKE variants, say how it performs on hiding endpoint identifiers. Does it hide the initiator's and/or the responder's IDs from eavesdroppers? How about active attackers? (Hint for one tricky case: what are the implications if Alice sends the hash of Bob's certificate as she can optionally do in the public encryption variants?)

Photuris: hides both from passive attackers. Hides Bob's from active attackers since Alice has to divulge and prove her identity first.

IKE public signature key, main: both hidden from passive attackers. Hides Bob's from active attackers since Alice to divulge and prove her identity first.

IKE public signature key, aggressive: neither hidden from passive or active attacker

IKE public encryption key, main, and aggressive: both hidden from passive as well as active attacker. If Alice sends the hash of Bob's certificate, then she leaks his identity to an attacker who has a list of possible recipients and their keys.
Design a protocol in which one side has a public signature key and the other side has a public encryption key.

In the following, Alice sends her Diffie-Hellman value signed with her signature key and encrypted with Bob's public key.

Alice has signature key

- “Alice”, \( ([g^a \mod p]_{Alice})_{Bob} \)
- \( g^b \mod p, \text{hash}(g^{ab} \mod p) \)
- \( \text{hash}'(g^{ab} \mod p) \)

Bob has encryption key

Bob needs his decryption key to know \( g^a \mod p \)

In the public encryption key case, SKEYID is defined as hash(nonces, cookies). SKEYID is supposed to be something that is not computable except by Alice and Bob. Why can’t an eavesdropper or active attacker calculate SKEYID?

Because the nonces are encrypted with the public keys of Alice and Bob (one is encrypted with Bob’s public key, the other with Alice’s, and you need to know both in order to compute SKEYID).

Compare the performance of doing PFS as implemented in SSLv3, vs. the recommended modification in §19.13.2 Exportability in SSLv3 vs. doing a Diffie-Hellman exchange for each connection. Assume the ephemeral key pair is of adequate length, rather than done to meet export rules. Assume also that PFS doesn’t need to be “perfect”, but rather the ephemeral key can change, say, every hour, and the cost of generating the key pair can be amortized over all the connection requests that occur during that hour.
IKE achieves PFS using ephemeral Diffie-Hellman key exchange. In other words, each Diffie-Hellman private/public key pair must be used only for one session. The long-term keys (e.g., RSA keys, or symmetric keys) are used only for authenticating the ephemeral Diffie-Hellman public keys.

PFS, SSLv3, as implemented: client does a cheap RSA encrypt, and a cheap RSA verify signature (cheap because both use the public key). The server has an expensive decrypt and an expensive sign on every authentication.

With the recommended modification: the server only has to do the expensive sign once an hour or so. The disadvantage of the modification is that it requires putting in an expiration time, and therefore requires relatively synchronized clocks.

With Diffie-Hellman, each side has to do an expensive Diffie-Hellman exponentiation, and in addition the server would have to sign its Diffie-Hellman number on each interaction. If PFS is less than perfect, the server can reuse its Diffie-Hellman b value, and not have to keep re-signing that.

6. IKE achieves PFS using ephemeral Diffie-Hellman key exchange. In other words, each Diffie-Hellman private/public key pair must be used only for one session. The long-term keys (e.g., RSA keys, or symmetric keys) are used only for authenticating the ephemeral Diffie-Hellman public keys.