Cryptography II

Question 1 ***Diffie–Hellman key exchange*** **(15 min)**

Recall that in a Diffie-Hellman key exchange, there are values a, b, g and p. Alice computes $g^a \bmod p$ and Bob computes $g^b \bmod p$.

(a) Which of these values (a, b, g, and p) are publicly known and which must be kept private?

> **Solution:**
>
> g and p are publicly known. Implementations of Diffie-Hellman often have carefully picked values of g and p which are known to everyone. Alice and Bob must keep a and b secret respectively.

(b) Mallory can eavesdrop, intercept, and modify everything sent between Alice and Bob. Alice and Bob perform Diffie-Hellman to agree on a shared symmetric key k. After the exchange, Bob gets the feeling something went wrong and calls Alice. He compares his value of k to Alice's and realizes that they are different. Explain what Mallory has done.

> **Solution:**
>
> Mallory is performing a **man-in-the-middle attack**. Mallory pretends to be Bob when she talks to Alice, and Mallory also pretends to be Alice when she talks to Bob. In this way, both Alice and Bob are unknowingly talking to Mallory. Mallory can then decrypt/re-encrypt the traffic in both directions and modify it however she wishes to.
>
> More technically, when Alice sends $A = g^a \bmod p$ to Bob, Mallory intercepts this (preventing it from going to Bob), and sends back to Alice: $M = g^c \bmod p$. Now when Alice sends a message to Bob, she uses $K_{bad} = M^a \bmod p$ which Mallory knows as $K_{bad} = A^c \bmod p$. Mallory can then decrypt all messages sent from Alice. She can also send messages to Alice which Alice thinks are from Bob. Mallory then does the same trick to Bob.

(c) Alice and Bob want to prevent Mallory from tampering with their keys by attaching a hash to each message (ie. Alice sends $(g^a, H(g^a))$ and Bob sends $(g^b, H(g^b))$). Does this successfully stop Mallory?

Solution: No, this does not stop Mallory. Since a hash function is public, Mallory can do the same attack as before by additionally attaching a hash of her spoofed key.

Question 2 ***Perfect Forward Secrecy*** **(15 min)**

Alice (A) and Bob (B) want to communicate using some shared symmetric key encryption scheme. Consider the following key exchange protocols which can be used by A and B to agree upon a shared key, k.

El Gamal-Based Key Exchange

Message 1	$A \to B$:	$\{k\}_{PK_B}$
	Key exchanged	
Message 2	$A \leftarrow B$:	$\{secret1\}_k$
Message 3	$A \to B$:	$\{secret2\}_k$

Diffie-Hellman Key Exchange

Message 1	$A \to B$:	$g^a \mod p$
Message 2	$A \leftarrow B$:	$g^b \mod p$
	Key exchanged	
	$k = g^{ab} \mod p$	
Message 3	$A \leftarrow B$:	$\{secret1\}_k$
Message 4	$A \to B$:	$\{secret2\}_k$

Some additional details:

- PK_B is Bob's long-lived public key.
- k, the Diffie-Hellman exponents a and b, and the messages themselves are destroyed once all messages are sent. That is, these values are not stored on Alice and Bob's devices after they are done communicating.

Eavesdropper Eve records all communications between Alice and Bob, but is unable to decrypt them. At some point in the future, Eve is lucky and manages to compromise Bob's computer.

(a) Is the confidentiality of Alice and Bob's prior El Gamal-based communication in jeopardy?

Solution: Yes. The compromise of Bob's computer gives Eve access to Bob's private key, allowing Eve to decrypt the traffic she previously recorded that was encrypted using Bob's public key. Once decrypted, she obtains k, and can then apply it to decrypt the traffic encrypted using symmetric key encryption.

(b) What about Alice and Bob's Diffie-Hellman-based communication?

Solution: No. Since Alice and Bob destroy the DH exponents a and b after use, and since the key computed from them itself is never transmitted, there is no information present on Bob's computer that Eve can leverage to recover k. This means that with Diffie-Hellman key exchanges, later compromises in no way harm the confidentiality of previous communication, even if the ciphertext for that communication was recorded in full. This property is called *Perfect Forward Secrecy*.

Question 3 ***Hashing passwords with salts*** **(15 min)**

When storing a password pw, a website generates a random string salt, and saves:

$$(\mathsf{salt}, \mathsf{Hash}(\mathsf{pw} \parallel \mathsf{salt}))$$

in the database, where Hash is a cryptographic hash function.

(a) If a user tries to log in with password pw' (which may or may not be the same as pw), how does the site check if the user has the correct password?

(b) Why use a hash function Hash rather than just store pw directly?

(c) Suppose the site doesn't use a salt and just stores $\mathsf{Hash}(\mathsf{pw})$. What attack becomes easier?

(d) Suppose the site has two candidate hash functions Hash_1 and Hash_2. Their properties are shown in the table below.

Function	One-Way	Collision Resistant
Hash_1	Yes	No
Hash_2	Yes	Yes

Which of them suffice for password hashing?

Solution:

(a) The site computes $\mathsf{Hash}(\mathsf{pw}' \parallel \mathsf{salt})$ using the salt in the database. If this hash output equals the stored hash value, the password is correct.

(b) If the hash function is secure and the password has good entropy, even if an attacker hacks into the site, the attacker cannot figure out the passwords.

(c) It makes inverting the hash much easier. Many hackers use a precomputed inverse hash table for some common passwords to reverse the hashes of common passwords.

Salts disable such tables and force the attacker to perform at least a dictionary attack for each user.

(d) Both suffice since we only need one-wayness.