Week of February 12, 2018: Cryptography II

Question 1 ***Public Key Basics*** **(10 min)**

Assume Alice and Bob are trying to communicate over an insecure network with public key encryption. Both Alice and Bob have published their public keys on their websites. (You can assume they already know each other's correct public key)

(a) Alice receives a message: *Hey Alice, it's Bob. You owe me 100 bucks. Plz send ASAP*. The message is encrypted with Alice's public key. Can she trust it?

(b) Bob receives a message: *Hey Bob, that last message you sent me was sketchy. I don't think I owe you 100 bucks. You owe me.* The message is digitally signed using Alice's private key. Can he trust it was from Alice? How does he verify this message?

(c) Alice receives a message: *Hey Alice, I know things have been rough these last two messages. But I trust you now. Here's the password to my Bitcoin wallet which has over \$100.* The message is encrypted with Alice's public key. Alice decrypted this and tested the password, and it was in fact Bob's! Can an eavesdropper figure out the password?

Question 2 *Confidentiality and Integrity* (25 min)

Alice and Bob want to communicate with both confidentiality and integrity. They have:

- A symmetric key encryption function $E(K, M)$ and corresponding decryption function $D(K, M)$. They have already securely shared the key K.
- A cryptographic hash function H (recall that H is not keyed, so anyone can compute $H(x)$ given x).
- A secure MAC generation function $MAC(K, M)$.
- RSA Signature Algorithm $\mathit{SIGN}_d(M)$ and $\mathit{VERIFY}_n(M, S)$, with d being Alice's private key and n being her public key. (For convenience, we often leave out mention of e as part of the public key.)

You may assume that H, E, F, and SIGN_d do not interfere with each other when used in combination—for example, if you compute $H(E(K, M))$, the message M will be confidential because E guarantees it, and the computation of H makes no difference. To send message M to Bob, Alice has the option to use any of the six schemes listed below.

Alice Sends to Bob
1. $C = H(E(K, M))$
2. $C = C_1, C_2$: where $C_1 = E(K, M)$ and $C_2 = H(E(K, M))$
3. $C = C_1, C_2$: where $C_1 = E(K, M)$ and $C_2 = MAC(K, M)$
4. $C = C_1, C_2$: where $C_1 = E(K, M)$ and $C_2 = MAC(K, E(K, M))$
5. $C = \mathit{SIGN}_d(E(K, M))$
6. $C = C_1, C_2$: where $C_1 = E(K, M)$ and $C_2 = E(K, \mathit{SIGN}_d(M))$

(a) Consider the threat model where Eve is only able to eavesdrop and Alice and Bob are using the key K for the first time and will use it once and only once. Furthermore, Eve has no additional information about the plaintext messages that will be sent. For each scheme above, determine whether or not the scheme (1) provides *confidentiality* and (2) if Bob is able to *correctly decrypt* the message (if he is able to, write down the decryption procedure).

(b) Now consider the threat model where Mallory is able to eavesdrop and actively alter/forge the message Alice sends to Bob. For each decryptable scheme, determine whether or not the scheme provides *integrity*; that is, will Bob be able to detect any tampering with the message? If the scheme provides integrity, describe how Bob checks the integrity of the message. If not, describe how Mallory could alter the message without Bob detecting.

(c) If Alice and Bob use these schemes to send many messages, the schemes become vulnerable to a *replay attack*. In a replay attack, Mallory remembers a message that Alice sent to Bob, and some time later sends the exact same message to Bob, and Bob will believe that Alice sent the message. How might Alice and Bob redesign the scheme to prevent or detect replay attacks?

Question 3 ***Password Hashing*** **(10 min)**

When storing a password p for user u, a website randomly generates a string s (called a "salt"), and saves the tuple $(u, s, r = H(p||s))$, where H is a cryptographic hash function.

(a) Say user u tries to log in submitting a password p' (which may or may not be the same as p). How does the site check if the user should be allowed to log in?

(b) Why use a hash function H rather than just store (u, p)? Isn't that just so much simpler? Keep It Simple Stupid, right?

(c) What is the purpose of the salt s?

(d) Suppose the site has three candidate hash function to choose from, H_1, H_2, and H_3. They each satisfy the following properties as displayed in the table.

Functions	One-Way	Second Pre-Image Resistant	Collision Resistant
H_1	Yes	No	No
H_2	Yes	Yes	No
H_3	Yes	Yes	Yes

A function H is *second pre-image resistant* if given m_1, it is difficult to find m_2 such that $m_1 \neq m_2$ but $H(m_1) = H(m_2)$. (Sanity check: how does this differ from being collision resistant?)

Which of these hash functions are suitable choices for the website's password hashing scheme, in that they provide a significant gain in security over just storing passwords?