Auth – Homework

Please answer the following questions. **We don't expect formal proofs: rather, just a high-level argument from intuition.** Please submit your answers as a **PDF** to Gradescope. Collaboration is allowed and encouraged, but you must write up your own answers and acknowledge your collaborators in your submission.

Due Date: *Monday, February 24th*

1 Block Cipher Modes of Operation

(1) If you were a security engineer and would like to adopt block cipher in your system for symmetric-key message encryption, which mode of operation (among ECB/CBC/C-TR/OFB modes) would you choose? Why?

(2) What do you need to pay attention to during the deployment of your chosen mode of operation?

2 Man-in-the-Middle Attacks

In the Auth project, the user and server first perform an authenticated key exchange step. In particular, the user first sends g^a which is the user's public value. Then, the server sends back (g^b, g^a, σ_s), where g^b is the server's public value and σ_s is a signature computed on (g^b, g^a). Consider a slightly modified protocol where instead the server sent back (g^b, σ'_s) where σ'_s is a signature on g^b.

(1) Explain a potential man-in-the-middle attack that could arise in the modified protocol, and why such a man-in-the-middle adversary still cannot learn anything about the encrypted messages sent from the user to the server.

(2) Consider a weaker authentication scheme *without* two-factor authentication (2FA). Suppose an adversary was able to successfully learn the shared secret g^{ab} during the login phase. Explain why, even though we never send the password over the wire, and even if the user chooses a very strong password, the adversary may be able to authenticate as this user in the future. Explain why 2FA solves the problem.

3 Offline Dictionary Attacks

Our password-based authentication scheme is designed to protect against adversaries that could potentially corrupt the entire storage of the server. Recall that given a hash function H (modeled as a random oracle) and a password pwd, the following is how we register and login:

Registration: First, the user sends their id and the server sends a random λ-bit salt to the user. Next, the user computes $h = H(\mathsf{pwd} \,\|\, \mathsf{salt})$ by hashing the password with the salt appended. The user then sends h to the server. Next, the server will choose a random $\mathsf{pepper} \in \{0,1\}^p$ for some small p and compute $h' = H(h \,\|\, \mathsf{pepper})$. Finally, the server stores a row $(\mathsf{id}, h', \mathsf{salt})$.

Login: First, the user sends their id and the server responds with the stored salt to the user. Next, the user computes and responds with $h = H(\mathsf{pwd} \,\|\, \mathsf{salt})$ by hashing the password with the salt appended. The server will then try for all $\mathsf{pepper}^* \in \{0,1\}^p$ computing $h^* = H(h \,\|\, \mathsf{pepper}^*)$, and authenticating the user if h^* matches the stored value h' for any pepper^*.

(1) Explain why this verification scheme is correct; that is, a valid password should be cleared for login.

(2) Explain an (inefficient) attack to recover users' passwords in case the server's entire database is compromised.

(3) Consider a weaker scheme where the the server only stores h (instead of hashing it again with a pepper to obtain and store h'). Explain an (inefficient) attack to recover users' passwords in case the server's entire database is compromised.

(4) What roles do the salt and pepper play against offline dictionary attacks, respectively?

(5) Given salt-and-pepper hashing, can a user safely use a simple/weak password?

4 Authentication

(1) During registration, we ask the user to provide a valid PRF response, even though we just sent the user the PRF key (or PRG seed). Why is this step necessary?

Hint: we allow each id *to register only once, and our network may not be stable.*

(2) A time-to-live, or TTL, specifies the expiration date for a certificate. This is useful if we don't want to indefinitely authenticate a user, but rather clear a user for the next day or so. Explain how you would extend our existing protocol to support TTLs. Be sure to ensure that a user can't change the TTL of their certificate without help from a trusted server.

5 Delegated Trust

The way that our authentication scheme works is that since we trust the server, the server can **delegate trust** to others that it trusts, allowing us to verify the identity of a third party without consulting directly with the server. We'll explore the ideas behind larger schemes such as Public Key Infrastructure (PKI).

(1) Propose a protocol that allows users to delegate trust to other users. What does delegation look like? What does verification look like?

(2) Suppose a secret signing key of some user u has been compromised, and suppose we have some way of invalidating certifications (e.g., a public revocation board). Which users should have their certificates invalidated and reissued in the case of such a compromise?