CS255, Winter 2020
Introduction to Cryptography

Project 1 Short Answer Questions

Silvia Gong, Julian Cooper

Question 1: Briefly describe your method for preventing the adversary from learning information about the lenths of the passwords stored in your password manager.

We will want to padd our password strings to ensure they are of equal length before encrypted with AES-GCM and stored. The encrypt() and decrypt() methods from the subtleCrypto library handle padding (and unpadding) for us.

Question 2: Briefly describe your method for preventing swap attacks. Provide an argument for why the attack is prevented in your scheme.

We defend against swap and rollback attacks in our scheme by checking that the SHA-256 hash of our json dump has not change between dump and load steps. To do this, we need to be able to store the hash of our json dump in a trusted location so we have a source of truth to check against.

In addition, we explicitly handle swap attacks by encrypting name —— password, not just password, when populating out key-value store in set(name, value). This then allows us to check whether the first part of the decrypted "value" matches name when we call get(name). This second line of defense is important because we might not always be guaranteed a trusted storage medium as required above.

Why does this work? Imagine our adversary switches the values of "service1" and "service2". Then when we call get("service1") to retrieve its password, we find that the first part of the decrypted "value" equals "service2". Since this does not match "service1", we throw and exception "Potential swap attack!". If no swap had occured, we would pass this check and return the decrypted password string usual to the caller.

Question 3: In our proposed defense against rollback attack, we assume that we can store the SHA-256 hash in a trusted location beyond the reach of an adversary. Is it necessary to assume that such a trusted location exists, in order to defend against rollback attacks?

Yes, having a trusted storage location beyond reach of an adversary is required in order to defend against rollback attacks. This is because rollback attacks reproduce legitimate domain names and passwords (i.e. signed with the right secret keys) and so key verification will not surface an attack.

Question 4: Explain how you would do the look up if you had to use a randomized MAC instead of HMAC. Is there a performance penalty involved, and if so, what?

In the case of a randomized MAC, there may be many valid tags $t_1, t_2, ...$ for a given message

m. Assuming the randomized MAC is secure, it must still be difficult for an adversary to produce a new valid tag $t\prime$.

For key-value store look up, we no are no longer able to recompute tag of the name and directly lookup key-value pair in O(1). Instead we will need to loop through all n elements in the key-value store dictionary, and for each run the randomized MAC verify function to query whether we are at the right key-value pair. This look up algorithm will cost O(n * [time to verify]).

Question 5: In our specification, we leak the number of records in the password manager. Describe an approach to reduce the information leaked about the number of records.

Instead of storing our key-value pairs in a "flat" dictionary, we could think about storing we could think about using a binary tree.

- Each node of the tree will contain pointers to its child nodes, until we reach the leaf nodes which will contain nullptrs for child nodes and data packet with key-value pairs of our key-value store.
- Since it is unlikely we have the exact number of key-value pairs as leaves of a sufficiently large binary tree (would need to be a power of 2), we will then pad the remainder of the leaves with fake data of same size as our real key-value pair data packets.
- The height of a binary tree is ceil(log(n+1)) where n is the number of nodes (= 2*leaves-1). We assume the adversary can see the size of the tree and therefore know its height and number of leaves. However, they will not know how many of leaves represent real vs fake key-value pairs.
- Finally, there could be anywhere from ceil(log(n)) (number of leaves at previous level of tree) to ceil(log(n+1)) (number of leaves at current level of tree) real key-value pairs. This range is equal to floor(log(n)).

Question 6: What is a way we can add multi-user support for specific sites to our password manager system without compromising security for other sites that these users may wish to store passwords of? That is, if Alice and Bob wish to access one stored password (say for nytimes) that either of them can get and update, without allowing the other to access their passwords for other websites.

We first use Diffie-Helman key exchange to generate a shared secret between Alice and Bob. We will call this the sharedMasterKey which Alice and Bob each then store separately in their own respective trusted locations.

With our sharedMasterKey, both Alice and Bob can use this to encrypt and decrypt any shared sites (domain names, e.g. nytimes). This need not compromise security for other sites that both users wish to maintain single-user access to because they can just continue to use their respective existing master keys for these.