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

- The stored passwords are encrypted using AES-GCM before being stored in the key-value store (kvs).
- Since AES-GCM produces ciphertexts that include random initialization vectors (IVs), the lengths of passwords are masked.
- Even if an adversary sees the encrypted storage, they cannot infer the actual password length from ciphertext size alone.

2. **Briefly describe your method for preventing swap attacks (Section 2.2). Provide an argument for why the attack is prevented in your scheme.**

- Swap attacks occur when an attacker swaps encrypted password entries.
- The HMAC function (hashDomain) ensures that domain names are hashed before storage, preventing direct swaps based on domain lookups.
- Additionally, passwords are encrypted under a unique key derived from the master password, preventing an attacker from reusing encrypted values across different accounts.

3. **In our proposed defense against the rollback attack (Section 2.2), 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? Briefly justify your answer.**

- The SHA-256 hash of the password manager state is used as an integrity check.
- If an attacker rolls back the stored state to an earlier version, the hash will no longer match the expected checksum.
- If no trusted storage is available, rollback attacks could still be mitigated by using a user-generated counter or a Merkle tree-based verification system stored redundantly across multiple devices.

4. **Because HMAC is a deterministic MAC (that is, its output is the same if it is run multiple times with the same input), we were able to look up domain names using their HMAC values. There are also randomized MACs, which can output different tags on multiple runs with the same input. 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**?

- With randomized MACs, the same input produces different outputs each time, preventing direct lookups.
- To perform lookups, we would store a mapping from plaintext domain names to all previously generated randomized MACs.
- This introduces a performance overhead, as each lookup would require scanning multiple stored values rather than a direct key-value retrieval.

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. Specifically, if there are k records, your scheme should only leak ⌊log2 (k)⌋ (that is, if k1 and k2 are such that ⌊log2 (k1)⌋ = ⌊log2 (k2)⌋, the attacker should not be able to distinguish between a case where the true number of records is k1 and another case where the true number of records is k2).**

- Instead of storing k records directly, we can store records in "buckets" determined by $\lfloor \log_2(k) \rfloor$.
- This means an attacker can only infer the approximate range of stored records rather than the exact count.
- Padding or dummy records could also be added to further obscure the exact number of entries.

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.

- Encrypt stored passwords using a shared secret key between Alice and Bob for the nytimes account.
- Each user derives independent keys for their personal passwords, ensuring they remain private.
- A user-specific HMAC key prevents one user from accessing another's unrelated stored passwords.
- A public-key encryption scheme could also be employed to allow shared access without revealing personal encryption keys.