HW3: Authenticated Encryption (Due: March 22, 2016)

Recall from class that symmetric encryption secure against passive adversaries is insufficient. We want security in the face of active attackers that can attempt to modify ciphertexts as well. Authenticated encryption (AE) schemes target confidentiality and unforgeability: no attacker can force decryption to accept a ciphertext not produced by someone with the secret key. In this exercise we'll implement AE schemes.

The deliverable with be a replacement fernet2.py that extends fernet.py to handle AEAD more generlaly. Also fernet2.py should handle decrypting both Fernet version 0x80 ciphertexts as well as a new version 0x81. The new encryption will provide more robust key handling, removal of timestamps, and allowing caller-specified associated data. You should start by copying and pasting fernet.py to fernet2.py, and then work in fernet2.py.

Part .1 Key Generation (20 points): Fernet.py uses base64 URL safe encoded 32 byte keys. This could be dangerous: imagine the developers at your startup read elsewhere that AES-based encryption requires only 16 byte keys, and so common bug was padding a random 16 byte value to satisfy the Fernet API. This would violate the spec (https://github.com/fernet/spec/blob/master/Spec.md) but bugs happen.

Instead, Fernet2 should derive two sub-keys from the main key. Use a secure PRF or one of the key-derivation function (KDF) implementations in hazmat to generate from the 32-byte fernet.py key the _encrypt_key and the _signing_key. For compatibility reasons, don't change the interface of the constructor of Fernet2 class, which should also accept 32-byte keys encoded in urlsafe-base64 format.

Part 2. AEAD (20 points): Implement an authenticated encryption with associated data (AEAD) scheme. The Fernet2.encrypt function should now take a message as well as data that won't be encrypted but will be included in the authentication scope. The format of ciphertexts should be 0x81 || IV || Ctxt || Tag, meaning one omits timestamps and also does not include the associated data in the output of encryption. The function Fernet2.decrypt should reject if the wrong associated data is provided.

Part 3 (20 points): Make Fernet2.py transparently handle legacy version 0x80 ciphertexts. This means that Fernet2.decrypt() should figure out what kind of ciphertext it has been given, and still successfully decrypt either the output of Fernet.encrypt() or Fernet2.encrypt(). Make necessary design choices for the interface, ensuring the backward compatibility.

Part 4. PW-based AEAD (20 points): Write a new class in fernet2.py, called PwFernet, that implements password-based AEAD. That is, replace derivation of the two keys with use of a valid PBKDF function. There is one in hazmat. Remember that each encryption should use a freshly generated salt with the PBKDF. The ciphertext format should match that of Fernet2 version 0x81, except with a version number 0x82 to indicate password-based encryption.

Part 5. Testing (20 points): Write a test_fernet2.py testing routine that starts with the test_fernet.py routines and extends them to cover all the new functionality. Write a spec2.md in the style of spec.md that explains the new encryption modes and how to use them securely.