C&O 487/687: Assignment #3

Due date: November 3, 2022 (Thursday)

Note: Some questions use randomization to customize to you specifically. Please include your max-8-character UW user id (`j2tham`) at the beginning of your answer so we can look up your custom solution.

1. [10 marks] **Symmetric Encryption in Python**

   In this problem, you will be asked to create a small Python 3 script for encrypting and decrypting snippets of text. We will be using the `cryptography` library for Python 3. You can read the documentation here: `https://cryptography.io/en/latest/hazmat/primitives/cryptographic-hashes/`. The script will take input from "standard in", prompt the user for a password, and then return the result to "standard out".

   In `fernet.py` (available for download from LEARN), you will find a Python script that does just this. To use it, you can create a file `plain.txt` and then feed it to the script as standard in. You can easily encrypt and decrypt the file using the command line.

   To encrypt:

   ```
   python3 fernet.py -e fernet.py < plain.txt > cipher.txt
   ```

   To decrypt:

   ```
   python3 fernet.py -d fernet.py < cipher.txt > decrypted.txt
   ```

   The operator `<` specifies that the following file should be used as standard in, while the operator `>` tells the computer to put standard out into that file. Note that on some systems, `python3` might simply be called `python`. You can check by running `python --version` and seeing if it's on a version $\geq$ to 3.7.

   Your first task in this question is to make sure you can run `fernet.py`. Run the script as described above. If you are missing any packages, you can use `pip` to install them (for example `pip install cryptography`) (or `pip3` instead of `pip`), which comes with most python installations nowadays. Otherwise, there are plenty of tutorials online for getting `pip` and any python packages you may need.

   `fernet.py` makes use of a Fernet "recipe". This recipe combines together several cryptographic primitives in order to make things easier for the programmer. Your task is to write a script that does the same thing as `fernet.py`, but without relying on Fernet recipes. In particular, your script should:

   - Use AES-128 for encryption in CBC mode,
   - Use PKCS7 padding,
   - Use HMAC to provide integrity,
   - Use PBKDF2 as a KDF, with 100 000 iterations,
   - Use SHA256 as a hash function,
   - Use the URL-safe encoding that Fernet uses before printing to standard out.

   You can use any of the primitives in the Hazardous Materials Layer of the Cryptography package (no recipes). Submit your code through Crowdmark, as you would a normal assignment – as a PDF or screenshot Make sure your code is readable and has helpful comments, as we will be grading it by hand, not executing it.

2. [11 marks] **Password Hash Cracking**

Please include your max-8-character UW user id (`j2tham`) at the beginning of your answer so we can look up your custom solution.

In this problem, you will be cracking password hashes. You will have to write your own code to do this. We recommend using Python and its `hashlib` library, which includes functions for computing a SHA256 hash. For example, the following line of Python code will compute the SHA256 hash of a string `s`:

```
hashlib.sha256(s.encode()).hexdigest()
```

Please include all code used in your submission.

Suppose you are a hacker that is interested in gaining access to Alice's online accounts. You've managed to obtain the user databases several websites, giving you knowledge of the hashes of several of Alice's passwords. All hashes in this question are SHA256.

(a) [1 marks] The hash for Alice's password on `123.com` is

`f5fd5ffef3c528d3d9282d1299503b669a3b0549c93cbb8a6aef7a12225cbe10` .

`123.com` doesn't use salts when hashing their passwords, and they require that their passwords have only lowercase letters. You should be able to look this hash up in an online hash database. What is Alice's password?

(b) [2 marks] The hash for Alice's password on `456.com` is

`7bc5c1e383607e95a206d20e1d1d3fd8eab147bcb7420a5eb0fc672f4ed92755` .

`456.com` prepends salts to their passwords before hashing (so you won't be able to look this one up), but they require their passwords to be 6 digits (0-9). The salt that was included with Alice's password is 16705816. Write a program to brute-force Alice's password.

Note: if you think this kind of password requirement is silly, BMO (the major Canadian bank) had exactly this as their password requirement until sometime in March 2019).

(c) [4 marks] The hash for Alice's password on `789.com` is

`cd01a419235b5e283b12b7da8bbf53d04c89231c169d6ca4594227cde0ffa85e` .

`789.com` also uses password salts (prepended to the password before hashing), and you recovered the salt 97375774. `789.com` has more sensible password requirements. Here are the requirements:

- The password must be at least 12 characters, and can include letters, numbers, and the 6 special characters !, ?, *, $, #, and &, but nothing else (NOTE: this is too much to brute force!).
- The password must contain at least one uppercase and at least one lowercase letter.
- The password must contain at least one number (0-9).
- The password must contain one of the 6 special characters.

Alice is smart, and never re-uses her passwords, and never writes them down. However, she needs to make sure they are easier to remember than random passwords. Three of Alice's accounts have previously been compromised in large scale compromises. The passwords she used for those accounts were `Directive82!`, `Villages615*`, and `Witness2904#`. We have compiled a small dictionary of words for you to use on LEARN. Figure out how Alice comes up with her passwords, and then write a program that makes use of that information to brute-force Alice's password on `789.com`. Keep track of how many hashes your program computes for the next part.

(d) [2 marks] Suppose Alice changes her strategy. She picks two words from a dictionary of 20000 words, capitalizes the first letter of each, and then picks a 3-digit number and one of the 6 symbols, and concatenates them like so: `word1||word2||number||symbol`. How many hashes would it take to test all possible passwords made in this way? How many hashes did your program have to check in part (c)? Is this new strategy better for Alice?

(e) [1 marks] Specialized computing equipment (e.g. https://www.bitmain.com/) has been developed for the Bitcoin mining network that claims to be able to compute up to 110 TH per second of SHA256 hashes (that is, 110 trillion hashes per second). This equipment cannot currently be used for password-cracking, but suppose it was able to be re-purposed for this (or similar equipment developed for this purpose). Using your answer for part (d), calculate how long it would take such a machine to try all hash values for passwords using the scheme given in that part.

(f) [1 marks] What could `789.com` do to improve their password security practices?

3. [6 marks] **MAC schemes derived from hash functions.**

Let $f\colon \{0,1\}^{160} \to \{0,1\}^{80}$ be a compression function. Let $H : \{0,1\}* \to \{0,1\}^{80}$ be an iterated hash function derived from f as follows: To hash a message $m \in \{0,1\}^*$, do the following:

(i) Append a single 1 bit to the right of m, followed by just enough 0's so that the bit length of the resulting message m' is a multiple of 80. Let the 80-bit blocks of m' be $m_1, m_2, ..., m_\ell$

(ii) Let b be the 80-bit binary representation of ℓ.

(iii) Compute $H_1 = f(m_1, 0), H_i = f(m_i, H_{i-1})$ for $2 \le i \le \ell$. Then $H(m) = f(b, H_\ell)$.

Consider now the following three MAC schemes each with 80-bit secret key k.

(a) $\text{MAC}_k(m) = H(m) \oplus k$.

(b) $\text{MAC}_k(m) = H(k, m) \oplus k$.

(c) $\text{MAC}_k(m) = H(k, m) \oplus H(m)$.

Show that 2 of these 3 MAC schemes are insecure. That is, describe attacks on 2 of these 3 MAC schemes to demonstrate that they are not existentially unforgeable under adaptive chosen-message attacks. (Note: You should try to make your attacks as "practical" as possible—that is, use as few hash function evaluations as possible, and use as few calls to the MACing oracle as possible.)

4. [6 marks] **Authenticated encryption - nested encryption**

Let (E, D) be a secure authenticated encryption scheme (meaning it provides both confidentiality and integrity). Consider the following derived cipher (E', D'):

$$E'\left((k_1, k_2), m\right) : = E\left(k_2, E(k_1, m)\right)$$

$$D'\left((k_1, k_2), c\right) = \begin{cases} D(k_1, D(k_2, c)) & \text{if } D(k_2, c) \neq \text{reject} \\ \text{reject} & \text{otherwise} \end{cases}$$

(a) Show that (E', D') is AE-secure if the adversary knows k_1 but not k_2.

(b) Show that (E', D') is not AE-secure if the adversary knows k_2 but not k_1.

(c) Design a cipher built from (E, D) where keys are pairs $(k_1, k_2) \in \mathcal{K}^2$ and the cipher remains AE-secure even if the adversary knows one of the keys, but not the other.

5. [7 marks] **RSA**

   Consider a smartcard that is used to do RSA encryption, for example to protect credit card information during a purchase at a point-of-sale machine. The smartcard generates RSA ciphertexts using the Chinese remainder theorem. With (n, e) being the RSA public key and d the corresponding private key, then a message M is encrypted as follows:

   (i) Compute $m = H(M)$, where $H(\cdot)$ is a cryptographic hash function.

   (ii) Compute $c_p = m^{e_p} \bmod p$ and $c_q = m^{e_q} \bmod q$, where $e_p = e \bmod (p-1)$ and $e_q = e \bmod (q-1)$.

   (iii) Find c, $0 \le c \le n-1$, such that

$$c \equiv \begin{cases} c_p & \pmod p \\ c_q & \pmod q \end{cases}$$

   (a) [2 marks] Show that $c = H(M)^e \bmod n$; that is, prove that c is the correct encryption of M.

   (b) [2 marks] Explain why it might be more efficient to compute c using the procedure described above as opposed to using the square-and-multiply algorithm.

   (c) [2 marks] Suppose an adversary can cause the smart card to compute c_p incorrectly and c_q correctly, for example by inducing a carefully timed power spike. Let c' be the incorrectly computed signature on M. Show how an adversary can efficiently factor n if they know the public key (n, e) and the incorrectly signed message (M, c').

   (d) [1 marks] Suggest a practical method for preventing such an attack.

Academic integrity rules

You should make an effort to solve all the problems on your own. You are also welcome to collaborate on assignments with other students in this course. However, solutions must be written up by yourself. If you do collaborate, please acknowledge your collaborators in the write-up for each problem. *If you obtain a solution with help from a book, paper, a website, or any other source, please acknowledge your source. You are not permitted to solicit help from other online bulletin boards, chat groups, newsgroups, or solutions from previous offerings of the course.*

Due date

The assignment is due via Crowdmark by 8:59:59pm on November 3, 2022. Late assignments will not be accepted.

Office hours

Office hours will take place online via the Gather.town platform; see the link on LEARN under Contents → Course Information → Office hours.

- Thursday October 27 1–2pm
- Monday October 31 11am–12pm
- Tuesday November 1 11am–12pm
- Wednesday November 2 11am–12pm

- Wednesday November 2 2–3pm
- Wednesday November 2 4–5pm
- Thursday November 3 1–2pm
