C&O 487/687: Assignment #3

Due date: October 31, 2023 (Tuesday)

Note: Some questions use randomization to customize to you specifically. Please include your max-8-character UW user id (`g66xu`) 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, specifically the primitives in the "hazardous materials" layer of the cryptography library.[1] You can read the documentation here: `https://cryptography.io/en/latest/`.

   The starting code for this question is distributed as a Jupyter notebook called `a3q1.ipynb` which you can download from LEARN. You can then upload this notebook to the University of Waterloo Jupyter server at `https://jupyter.math.uwaterloo.ca/`. You can do all the programming for this question in the web browser, you do not have to install anything on your computer.

   You should implement two functions, one for encryption and one for decryption. Each function will take as input a string (the message or ciphertext, respectively), then prompt the user for a password, perform any relevant cryptographic operations, and then return the result.

   You should use the following cryptographic primitives to build an IND-CCA-secure authenticated encryption scheme using password-derived keys:

   - Use AES-128 for encryption in counter mode,
   - Use PKCS7 padding,
   - Use HMAC to provide integrity,
   - Use PBKDF2 as a KDF, with 200 000 iterations,
   - Use SHA3-256 as a hash function,

   You can use any of the primitives in the Hazardous Materials Layer of the Cryptography package (no recipes).

   (a) [5 marks] Write your encryption and decryption procedures in pseudocode.

   (b) [3 marks] Implement your encryption and decryption procedures in the provided Jupyter notebook.

   Submit your code through Crowdmark, as you would a normal assignment – as a PDF or screenshot of your Jupyter notebook. Make sure your code is readable and has helpful comments, as we will be grading it by hand, not executing it.

   (c) [1 marks] How many bytes of overhead does your encryption function add on top of the minimum number of bytes required to represent the message itself?

   (d) [1 marks] What is one mistake that someone could make when solving this question that could undermine security?

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

   Please include your max-8-character UW user id (`g66xu`) at the beginning of your answer so we can look up your custom solution.

[1] This portion of the library is labelled "hazardous materials" because it is easy for inexperienced users – and people who didn't do well in CO 487 – to use these building blocks incorrectly. Good cryptographic libraries often include all-in-one functions that are harder to use incorrectly. The Python 3 `cryptography` library has a package called "Fernet" recipes which are meant to be used by non-experts and are harder to misuse. If you are writing programs that cryptography in your career after CO 487, I recommend trying to use good libraries with good APIs for all-in-one recipes whenever possible to reduce the risk of human error.

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

`80f5846485aa518c92418bc7b0f2afe8eae6b65a20ec92362802346ec5a83cbe` .

`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

`aacc7cb90ee724457d09b06024761ac51f791d3394438442f950b17a31e16baf` .

`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 27615912. 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

`9edabf325bc63599725e1aa6a01a9b1a0bc09ff264bdbe7debb35f73a352a2cc` .

`789.com` also uses password salts (prepended to the password before hashing), and you recovered the salt 89535971. `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 at (download 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` (but without any spaces). 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. [8 marks] **Message Authentication Codes**

Suppose that

$$\mathsf{MAC} : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\tau$$

is a secure MAC scheme for fixed-length messages, which takes in a key $k \in \{0,1\}^\kappa$ and a message $m \in \{0,1\}^\ell$, and outputs a tag $t \in \{0,1\}^\tau$. We can construct two MAC schemes

$$\mathsf{BIGMAC} : \{0,1\}^\kappa \times \{0,1\}^{\ell \cdot *} \to \{0,1\}^\tau$$

and

$$\mathsf{WHOPPER} : \{0,1\}^\kappa \times \{0,1\}^{\ell \cdot *} \to \{0,1\}^{\tau \cdot *}$$

for messages of length $n \cdot \ell$ (where $n \geq 1$) as follows:

$\mathsf{BIGMAC}(k, m)$

1 : divide m into blocks $m_1, \ldots, m_n$, each of length ℓ
2 : **return** $t \leftarrow \mathsf{MAC}(k, m_1) \oplus \mathsf{MAC}(k, m_2) \oplus \cdots \oplus \mathsf{MAC}(k, m_n)$

$\mathsf{WHOPPER}(k, m)$

1 : divide m into blocks $m_1, \ldots, m_n$, each of length ℓ
2 : **return** $t \leftarrow \mathsf{MAC}(k, m_1) \| \mathsf{MAC}(k, m_2) \| \cdots \| \mathsf{MAC}(k, m_n)$

(a) [2 marks] Write verification algorithms for BIGMAC and WHOPPER.

(b) [3 marks] Is BIGMAC secure? If yes, provide a proof by contrapositive (show how an adversary who can break BIGMAC can be used to break MAC); if no, show how an adversary can forge a valid tag.

(c) [3 marks] Is WHOPPER secure? If yes, provide a proof by contrapositive (show how an adversary who can break WHOPPER can be used to break MAC); if no, show how an adversary can forge a valid tag.

4. [4 marks] **Authenticated encryption**

Suppose that $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is an IND-CPA-secure symmetric key encryption scheme for messages of length ℓ, and that $\Sigma = (\mathsf{KGen}, \mathsf{MAC}, \mathsf{Vf})$ is a secure (existentially unforgeable under chosen message attack) deterministic MAC scheme for messages of length ℓ.

We define an authenticated encryption scheme $\Pi' = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, for messages of length 2ℓ, using the encrypt-and-MAC paradigm, as follows:

$\Pi'.\mathsf{KGen}()$

1 : $k_1 \leftarrow \Pi.\mathsf{KGen}()$
2 : $k_2 \leftarrow \Sigma.\mathsf{KGen}()$
3 : **return** (k_1, k_2)

$\Pi'.\mathsf{Enc}((k_1, k_2), (m_1, m_2))$

1 : $c_1 \leftarrow \Pi.\mathsf{Enc}(k_1, m_1)$
2 : $c_2 \leftarrow \Pi.\mathsf{Enc}(k_1, m_2)$
3 : $t_1 \leftarrow \Sigma.\mathsf{MAC}(k_2, m_1)$
4 : $t_2 \leftarrow \Sigma.\mathsf{MAC}(k_2, m_2)$
5 : **return** $((c_1, c_2), (t_1, t_2))$

$\Pi'.\mathsf{Dec}((k_1, k_2), ((c_1, c_2), (t_1, t_2)))$

1 : **if** $\Sigma.\mathsf{Vf}(k_2, m_1, t_1) = \mathsf{false}$ **then return** $\perp$
2 : **if** $\Sigma.\mathsf{Vf}(k_2, m_2, t_2) = \mathsf{false}$ **then return** $\perp$
3 : **return** $(\Pi.\mathsf{Dec}(k_1, c_1), \Pi.\mathsf{Dec}(k_1, c_2))$

(where $\perp$ is a generic error symbol). Show that Π' does not satisfy semantic security.

5. [6 marks] **Pseudorandom functions**

   Suppose $F : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\ell$ is a secure pseudorandom function, in other words, for a random seed k of length λ and an arbitrary label x, the output $F(k,x)$ is indistinguishable from a random string of length ℓ, even for an adversary who can get the output of F on the same seed k with any other labels $x' \neq x$ of its choice.

   Which of the following are also secure pseudorandom functions? Briefly justify your answer.

   (a) $A : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$ defined by $A(k,x) = F(k,x) \parallel F(k,x)$

   (b) $B : \{0,1\}^\lambda \times \{0,1\}^{2*} \to \{0,1\}^\lambda$ defined by $B(k,x) = F(k,x_1) \oplus F(k,x_2)$ where $x = x_1 \| x_2$

   (c) $C : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$ defined by $C(k,x) = F(k,0\|x) \parallel F(k,1\|x)$

6. [6 marks] **RSA prime generation** Alice and her friend Alicia are finding that generating primes for their RSA modulus is quite slow and are looking for ways to speed things up.

   (a) [2 marks] Alice tells Alicia there is no need to generate two distinct primes p and q. Instead, she generated a single prime p and used $N = p^2$ as her modulus. Is this a good way to speed things up? Why or why not?

   (b) [2 marks] Alicia has a different idea. She tells Alice that they can each generate a prime and then multiply them together to get a shared modulus $N = pq$. Alice doesn't like this idea because she doesn't want Alicia to read her conversations with Bob. Alicia tells her they can just use different exponents. Explain why this is a bad idea and show how an attacker could recover any message m that Bob sends to both Alice and Alicia by using Euclid's algorithm on the public exponents. If it is helpful, you may assume that Alice and Alicia use co-prime public exponents.

   (c) [2 marks] Alice agrees to Alicia's idea but still has some reservations so instead of using $N = pq$ as her modulus, she generates another prime r and uses $M = pr$ as her modulus instead. Explain why this is a big mistake.

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 October 31, 2023. Late assignments will not be accepted.