C&O 487/687: Assignment #3

Due date: November 4, 2024 (Monday)

If you are requesting an extension or accommodation because of a verification of illness form, self-declared student absence, or AccessAbility Services accommodation, see the "Missed assignment / extension policy and forms" instructions on the CO 487/687 LEARN site.

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:

   - AES-256 for encryption in cipher feedback mode,
   - HMAC to provide integrity,
   - Scrypt as a key derivation function,
   - SHA3-512 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) [2 marks] Implement your encryption and decryption procedures in the provided Jupyter notebook. If you need to include multiple values in an output, you might find it helpful to use tuples or JSON data structures.

   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 mark] 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 mark] For the Scrypt parameters you chose, what is the estimated runtime for a single execution? How much memory will be required for Scrypt to operate?

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

[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.

2. [8 marks] **Password hash cracking**

> This question uses randomization to customize to you specifically. Please include your max-8-character UW user id (`b54khan`) 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 from several websites, giving you knowledge of the hashes of several of Alice's passwords. All hashes in this question are SHA256.

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

```
303b8f67ffbe7f7d5e74e9c2177cd1cabbc9dc53154199f540e1901591c7d5fa
```

`example.com` prepends salts to their passwords before hashing, as well as requiring that passwords be exactly 6 digits (0-9). The salt that was included with Alice's password is 19147384. What is Alice's password? Submit any code you write along with your answer.

(b) [6 marks] On `example.org`, they have stricter security requirements. First, they prepend a 64-bit salt k_1 to each password. The same salt k_1 is prepended to each password, and is kept secret (in particular, it is not stored alongside the password in the database, so compromising the user database does not reveal the fixed secret salt). The concatenated salt and password is then hashed with a cryptographic hash function. Then, the result is encrypted with a modified version of AES that uses a 64-bit key k_2. Again, the same key k_2 (which is not part of the database compromise) is used to encrypt every password. Finally, the result is stored in the password database alongside its corresponding username. More concisely:

$$c = \mathsf{AES.Enc}(k_2, \mathsf{H}(k_1 \| pw))$$

Suppose that, before you compromised the user database, you registered a few fake accounts on the server for which you know the plaintext password.

i. [2 marks] Is it feasible to carry out a exhaustive search to recover the salt and encryption key? Why or why not?

ii. [2 marks] Describe an attack which recovers both the key and the secret salt that takes only about $\sqrt{X}$ hash/encryption operations (or some small constant multiple thereof), where X is the number of operations required for a exhaustive search.

iii. [1 mark] Would the same attack work if `example.org` first encrypted each password and then hashed it with their secret salt? Why or why not?

iv. [1 mark] Would you recommend using this method of password hashing? Why or why not?

3. [4 marks] **Variable-length input MAC schemes**

In this question, we will explore attempts to build a MAC that handles variable-length inputs from MAC schemes that have fixed-length inputs.

Let $\mathcal{M} : \{0,1\}^{256} \times \{0,1\}^{256} \to \{0,1\}^{256}$ be a secure MAC scheme that takes as input a 256-bit key and a fixed-length 256-bit message, and produces as output a 256-bit tag.

While useful for small messages of fixed length, this MAC scheme cannot be used for longer messages without modification. Here are two attempts at such a modification. For each proposal, your task

is to decide if the scheme is secure or not. If it is, prove it using the assumption that $\mathcal{M}$ is secure. Otherwise, propose an attack.

For the rest of this question, for the variable-input-length MAC scheme we are trying to construct, we will simplify to the setting where the input messages have a length that is a multiple of the block size, and ignore the issue of padding.

(a) For a message m whose length is a multiple of 256, split it into blocks $(m_1, \ldots, m_n)$ each of length 256, and apply the following algorithm:

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

1 : **for** $i = 1, \ldots, n$:
2 : $\quad t_i \leftarrow \mathcal{M}(k, m_i)$
3 : **return** $t_1 \| \ldots \| t_n$

(b) For a message m whose length is a multiple of 256, split it into blocks $(m_1, \ldots, m_n)$ each of length 256, and apply the following algorithm:

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

1 : $t_0 \leftarrow 0^{256}$
2 : **for** $i = 1, \ldots, n$:
3 : $\quad t_i \leftarrow \mathcal{M}(k, m_i \oplus t_{i-1})$
4 : **return** t_n

4. [6 marks] **Authenticated encryption using ChatGPT**

We wanted to see if ChatCPT was a good cryptographer.[2] We asked ChatGPT to construct an authenticated encryption scheme by combining an encryption scheme with a MAC. The following are some of the bad constructions it proposed, ignoring those that do not compute at all. Your task is to explain why each construction is not secure.

Let E_{k_E} be an IND-CPA secure symmetric key encryption scheme with secret key k_E, and let M_{k_M} be a secure (existentially unforgeable under chosen message attack, a.k.a. EUF-CMA) message authentication code with secret key k_M.

For each of the following authenticated encryption schemes:

(i) Write out the corresponding decryption procedure. If you want to indicate that the ciphertext is not valid, you can use the pseudocode "**return error**".

(ii) Demonstrate that the scheme is insecure by doing the following:

1. Choose a secure symmetric key encryption scheme for E_{k_E} and explain why it is secure.
2. Choose a secure MAC for M_{k_M} and explain why it is secure.
3. Prove that the proposed authenticated encryption scheme is not secure (either it lacks IND-CPA security, or it lacks EUF-CMA security) when using your encryption scheme and MAC by explaining how to attack it.

Note that when we say "Choose a secure symmetric key encryption scheme" (or "Choose a secure MAC"), this means that you can assume there exists a generic secure symmetric key encryption scheme (or secure MAC), and you can either use it directly, or you can use the "degenerate counterexample proof technique" (Topic 2.1, slides 23–26) to build a second symmetric key encryption scheme (or MAC) from the generic one that remains secure (show this!), but conveniently has some strange property that helps you demonstrate the insecurity of the proposed authenticated encryption scheme.

[2] Good news: I am not out of a job yet.

(a) "Cascading Encryption and MAC Structure":

$\underline{\mathsf{AEON}(k = (k_E, k_M), m)}$

1 : $C \leftarrow E_{k_E}(m)$
2 : $T_1 \leftarrow M_{k_M}(m)$
3 : $T_2 \leftarrow M_{k_M}(T_1 \| C)$
4 : **return** (C, T_1, T_2)

(b) "XOR-Dependent Combined Encryption and MAC":

$\underline{\mathsf{AETHER}(k = (k_E, k_M), m)}$

1 : // select a uniformly random nonce of length equal to the plaintext
2 : $N \leftarrow\!\!\$\ \{0,1\}^{|m|}$
3 : $C \leftarrow E_{k_E}(m \oplus N)$
4 : $T \leftarrow M_{k_M}(C \oplus N)$
5 : **return** (C, T, N)

5. [9 marks] **RSA encryption**

The rivalry between math students and engineers has advanced to the next stage. After CO 487 students successfully broke the Awesome Engineering Squad's Not Very Permute-y cipher a month ago, a splinter group called the Really Super Awesome Association of Engineering Students (RSA-AES) have taken over with their revolutionary use of public key cryptography.

However, as everyone knows,[citation needed] engineers are very odd. So odd, in fact, that the Really Super Awesome Association of Engineering Students will only consider RSA-encrypted messages sent to them if those decrypted messages are odd (recall that messages for RSA are integers). To avoid angering people who do not get a response because their decrypted message is not odd, when the RSA-AES receives a ciphertext, they immediately decrypt it, then send a plaintext response back to the sender indicating whether they will be considering their message or not. (We call this interaction the "RSA plaintext parity checking oracle".)

Let (N, e) be the RSA public key of the Really Super Awesome Association of Engineering Students. Suppose you intercept a ciphertext, c^*, of a message $m^* \in [0, N-1]$ (with m^* odd), encrypted under (N, e).

(a) [2 marks] Show that, if $m_1, m_2 \in [0, N-1]$ with RSA encryptions c_1 and c_2, respectively, then

$$\mathsf{RSA.Enc}((N, e), m_1 \cdot m_2) = c_1 \cdot c_2 \bmod N$$

(b) [4 marks] Show how you can determine whether $m^* > \frac{N}{2}$ or $m^* < \frac{N}{2}$, by interacting with the RSA plaintext parity checking oracle. Justify mathematically why your approach works.

(c) [3 marks] Show how you can efficiently recover all of m^*. Justify mathematically why your approach works. How many interactions with the RSA plaintext parity checking oracle do you need for your attack?

Hint: Binary search.

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 11:59:59pm on November 4, 2024.

If you are requesting an extension or accommodation because of a verification of illness form, self-declared student absence, or AccessAbility Services accommodation, see the "Missed assignment / extension policy and forms" instructions on the CO 487/687 LEARN site.