# Homework 5

---

**Due Date**: **Tuesday, 11/24, 11:59pm**

## Instructions

1. Unless otherwise specified, any assignment involving programming may be completed with the programming language of your choice. If asked, you should be able to explain the details of your source code (e.g. program design and implementation decisions).
2. You are bound by the Stevens Honor System. All external sources must be properly cited, including the use of LLM-based tools (e.g. ChatGPT). Your submission acknowledges that you have abided by this policy.

### Additional Instructions

1. Solutions are to be uploaded on Gradescope under the corresponding assignment names. Note that portions of the homework may be autograded and require you to upload your programming solutions to a separate "assignment" on Gradescope, *in addition* to your written responses.
2. For full credit, programming solutions must pass the provided test cases in Gradescope, as well as any hidden test cases.

# Problem 1 - (*30 pts*) Domain-Extension MAC Implementation

In this problem you will be implementing CBC message authentication codes (CBC-MAC) to authenticate *arbitrary-length messages*. Given the supporting Python code (`cbc_mac.zip`), complete the following:

1. (*10 points*) Implement the `mac()` method.
2. (*10 points*) Implement the `verify()` method.
3. (*10 points*) In CBC-MAC, when the sender and receiver have agreed upon a fixed message length $l$, they can select the initalization vector $\text{IV} = 0$. However, when the messages are arbitrary length, such constructions are no longer secure against forgeries.

   In particular, assume that the sender only authenticates messages of length $2n$, while the receiver accepts arbitrary length messages. Show that when $\text{IV} = 0$ and an adversary has access to an oracle $\text{Mac}_k(m_i) \rightarrow t_i$, the adversary can forge an authenticated message of length $2n$.

# Problem 2 - (*35 pts*) On the RSA Cryptosystem

In this problem you will be implementing the basic RSA cryptosystem.

Given the supporting Python code (`rsa.zip`), complete the following. *Python skeleton code provides additional requirements and information for each function.*

1. (*5 points*) Implement the `modexp()` method for fast modular exponentiation. **Explain your code in your written solutions and cite any sources you used.** *Note: You must use your implementation fof `modexp`, not Python's built-in `pow` method.*
2. (*5 points*) Implement the RSA encryption algorithm `enc()`. **Explain your code in your written solutions and cite any sources you used.** *Note: You must use your implementation fof `modexp`, not Python's built-in `pow` method.*
3. (*5 points*) Implement the RSA decryption algorithm `dec()`. **Explain your code in your written solutions and cite any sources you used.** *Note: You must use your implementation fof `modexp`, not Python's built-in `pow` method.*
4. (*5 points*) Implement an algorithm to find the private key $d$ from the public key $(e, N)$ with a fixed value of $e = 65537$, `gen_private()`. **Explain your code in your written solutions and cite any sources you used.**
5. (*5 points*) Implement the RSA key generation algorithm `keygen()`. **Explain your code in your written solutions and cite any sources you used.**

   *Note: We have provided an oracle function `gen_primes` that generates the prime numbers $p$ and $q$ for you.*
6. (*10 points*) The most common choice of public exponent $e$ is 65537. Read this blog post to learn more about this choice of $e$. Briefly name and describe the property(-ies) of this value that makes it an efficient exponent.

## Problem 3 - (*35 pts*) Password Cracking!

You have compromised an online server and stolen their list of stored passwords along with the associated usernames and now you want to break into those accounts. (shame on you.) One problem: the server only stored the password *hashes*. (Seems the server had at least a little bit of defense in depth). Let's see if you can recover some passwords anyways…

You figure that there must be at least a few users who've picked common passwords or reused them, so you go download the top 2000 passwords from the RockYou wordlist, which contains over 14 million unique passwords leaked in data breaches. You put them in a file named `rockyou2000.txt`. Time to crack some passwords…

1. (*5 points*) Write a script named `md5.py` that takes a path to a plaintext file as a command line argument (e.g. `<filename>.txt`) containing a newline-separated list of passwords, hashes every password in the list, and outputs a file named `<filename>md5.txt` containing each password/hash pair. Each line of the output should be formatted as `<password>: <md5hash (hex)>`, listed in sorted order. *Note: you may use the MD5 implementation provided by the `hashlib` module in Python's standard library.*
2. (*5 points*) Write a script named `sha256.py` that takes a path to a plaintext file as a command line argument (e.g. `<filename>.txt`) containing a newline-separated list of passwords, hashes every password in the list, and outputs a file named `<filename>sha256.txt` containing each password/hash pair. Each line of the output should be formatted as `<password>: <md5hash (hex)>`, listed in sorted order. *Note: you may use the SHA256 implementation provided by the `hashlib` module in Python's standard library.*
3. (*5 points*) Write a script named `find_collisions.py` that takes 2 command line arguments: a path to a newline-separated list of user/hash pairs (formatted as `<username>: <pwhash (hex)>`, and a path to a newline separated list of password/hash pairs (a file generated by one of the previous scripts). The script will output a file named `cracked.txt` containing a list of user/plaintext password pairs. Each line of the output should be formatted as `<username>: <password>`, listed in sorted order.
4. (*10 points*) What security controls could have been implemented to help mitigate this attack? Name at least 2 distinct controls and describe the effect they would have.
5. (*10 points*) There are many password cracking tools available to attackers. Take a look at hashcat and read this article on cracking password hashes. List and describe the different kinds of attacks featured in the article.