Recitation 4
Number Theory and Encryption

Part 1: Some Important Number Theory

Definitions and Properties

- If \( a \equiv c \pmod{m} \) and \( b \equiv d \pmod{m} \), then \( ab \equiv cd \pmod{m} \).
- If \( a \equiv b \pmod{m} \), then \( a^n \equiv b^n \pmod{m} \), where \( n \in \mathbb{Z}^+ \).
- Two integers \( a \) and \( b \) are relatively prime if \( \gcd(a, b) = 1 \), i.e. they share no common factors other than 1.
- (Euler phi function) Euler’s \( \phi \) function is defined over \( n \in \mathbb{Z}^+ \) such that
  \[
  \phi(n) = |\{k \in \mathbb{Z} \mid 1 \leq k \leq n, \gcd(n, k) = 1\}|
  \]
  In essence: the number of integers between 1 and \( n \) that are relatively prime to \( n \). This can be pretty burdensome to calculate for larger \( n \); however, if \( n \) is of the form \( n = pq \), where \( p \) and \( q \) are primes, then we know \( \phi(n) = (p - 1)(q - 1) \).

Relevant Theorems

**Theorem** (Fermat’s Little Theorem) Let \( p \) be a prime. If \( \gcd(a, p) = 1 \), then \( a^{p-1} \equiv 1 \pmod{p} \).

**Theorem** (Euler-Fermat Theorem) If \( \gcd(a, m) = 1 \), then \( a^{\phi(m)} \equiv 1 \pmod{m} \).

Part 2: A First Look at Encryption

Think back to the first time a teacher caught you passing a note in class: wouldn’t it have been cool if that note looked like complete nonsense to your teacher but made sense to you and your friend?

The purpose of encryption is to allow people to communicate securely over some medium. Without secure cryptosystems (e.g. RSA), we wouldn’t be able to purchase goods on the web without the fear of someone stealing your credit card information.

Suppose that Mitchell and Neil devise a simple cryptosystem so that they can pass notes in class. They decide to use only the first 20 letters of the alphabet (i.e. \( A - T \)), and replace letters with their corresponding number (i.e. \( A = 1, B = 2 \), etc.).
They agree to encrypt a letter by taking its corresponding number—call this number $x$—and then outputting the result of $\text{rem}(3x, 20)$. To decrypt any encrypted message $y$, they agree to multiply by 7 and then take the remainder when divided by 20, i.e. $\text{rem}(7y, 20)$.

In order to convince yourself that this encryption scheme works, encrypt a four letter word of your choosing and then have a partner decrypt it. (For this recitation, you may use Wolfram-Alpha or some analogous software to help calculate these remainders)

Why does Mitchell and Neil’s encryption scheme work for this 20 character alphabet? Why can any encrypted digit be recovered exactly by the decryption process? **Hint:** consider the equation $3x \equiv 1 \pmod{20}$.

Mitchell and Neil get bored with using only 20 letters in their messages. Create your own encryption scheme that operates on a modulus of 26 but a similar encryption/decryption factor setup, i.e. multiplies by some encryption factor $e$ to encrypt and then multiplies by some decryption factor $d$ to decrypt.
If Mitchell and Neil swap $e$ and $d$, does their encryption scheme still work? Why or why not?

With a TA discuss some of the security flaws of Mitchell and Neil’s encryption scheme.

Checkpoint - Call a TA over
Part 3: RSA Encryption

Mitchell and Neil realize that their encryption scheme, albeit simple, is altogether too insecure. They opt instead to use RSA encryption to send notes to each other in class. Recall that the RSA encryption algorithm is as follows:

a. Choose two primes $p, q$.
b. Calculate $n = pq$.
c. Calculate $\phi(n) = (p - 1)(q - 1)$.
d. Choose some $1 < k < \phi(n)$ such that $gcd(k, \phi(n)) = 1$.
e. Find a $d$ such that $kd \equiv 1 \pmod{\phi(n)}$.
f. Publish $n$, the modulus, and $k$, the encryption exponent.
g. Keep $d$ private to yourself

To encrypt a message $m$, compute $r \equiv m^k \pmod{n}$ and to decrypt $r$ calculate $x \equiv r^d \pmod{n}$.

Now, create your group’s own personal RSA key by choosing prime values of $p$ and $q$ (ideally both less than 30 for convenience’s sake) and following the steps of the RSA algorithm to create your values of $n$, $k$, and $d$. Pick a partner group to communicate with but do not tell them the values of $p$ and $q$.

Now that every group has generated their own values of $n$ and $k$, encrypt a message for your partner group using their modulus $n$ and their encryption exponent $k$. Again, use an alphanumeric cipher where $A = 1$, $B = 2, \ldots, Z = 26$. In the interest of time, try to keep your message to ten or fewer characters.
Send your message along to your partner group on a piece of paper. Even if you get caught passing notes in class, the TAs won’t be able to understand it!

Take the message your partner group sent to you and decrypt it using your personal decryption exponent $d$.

Checkpoint - Call a TA over
Part 4: Hacking an Encryption Scheme

Suppose Neil publishes the following RSA key with modulus \( n = 143 \) and public key exponent \( k = 17 \). Mitchell then sends Neil the following message: \((117, 1, 37, 112)\). Hack into Mitchell and Neil’s communication and decrypt their message!

**Hint:** reverse engineer Neil’s RSA key by determining what the two primes \( p \) and \( q \) are.

Now that you’ve hacked Mitchell’s message to Neil, have you shown that RSA encryption is flawed? If not, what is it about RSA encryption that makes it extremely difficult to reverse engineer?

Checkpoint - Call a TA over
Part 5 (Optional): Perfect Encryption

So, we’ve established that RSA is breakable, given that we have the patience to wait for a factoring algorithm to determine \( p \) and \( q \) from the public modulus \( n \). If we were to choose \( p \) and \( q \) to be sufficiently large—say primes with hundreds of digits—then reverse engineering \( p \) and \( q \) would be such a computationally intensive task that a personal computer would likely take many, many years longer than we’re willing to wait for.

Hence, the security of RSA relies on the inherent difficulty of factoring—computer scientists contend that factoring is an exceedingly hard task. If someone were to propose an algorithm that could factor numbers efficiently, then many of our cryptosystems would be rendered moot; making credit card purchases online would suddenly be unrealistic.

As it turns out, there does exist an algorithm that could theoretically factor numbers in an efficient manner. But there’s just one catch: this algorithm (known as Shor’s algorithm) can only run on quantum computers. The closer we get to creating a sufficiently powerful quantum computer, the closer we get to rendering RSA useless. On the flip side, however, there do exist algorithms that offer perfect and unbreakable encryption schemes. These algorithms rely on quantum mechanics, so we can’t get too in depth, but it’s worthwhile noting that many countries are already employing these techniques right now! Just last year China launched a satellite equipped to encrypt messages using a quantum encryption algorithm!