CSCI 432 Handout: RSA Encryption

Name: _____

April 30, 2026

Modular Arithmetic Warm-up

Recall the three equivalent ways to write $a \equiv b \pmod{n}$:

$$a \equiv b \pmod{n} \iff n \mid (a-b) \iff \exists k \in \mathbb{Z} \text{ s.t. } a = kn + b.$$

1. Compute the following.
   (a) $17 \bmod 5$
   (b) $100 \bmod 13$
   (c) $-4 \bmod 9$

   Answer

2. Two integers a and b are *coprime* if $\gcd(a, b) = 1$. Use the Euclidean algorithm to compute the following GCDs and decide which pairs are coprime.
   (a) $\gcd(60, 17)$
   (b) $\gcd(120, 35)$
   (c) $\gcd(3120, 17)$

   Answer

3. Find an integer d such that $7d \equiv 1 \pmod{40}$. (You will need to do this for every RSA key generation, so practice the Extended Euclidean algorithm here.)

   Answer

Euler's Totient

Euler's totient $\varphi(n)$ counts the integers in $\{1, 2, \ldots, n\}$ that are coprime to n. Two facts make this easy for RSA:

- If p is prime, $\varphi(p) = p - 1$.
- If p and q are distinct primes, $\varphi(pq) = (p-1)(q-1)$.

4. Compute $\varphi(n)$ for the following.
   (a) $n = 7$
   (b) $n = 11$
   (c) $n = 77$ (note $77 = 7 \cdot 11$)
   (d) $n = 3233$

   Answer

Key Generation

Recall the steps for RSA key generation:

1. Choose two distinct primes p and q.
2. Compute $n = pq$.
3. Compute $\varphi(n) = (p-1)(q-1)$.
4. Choose a public exponent e with $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$.
5. Compute the private exponent $d \equiv e^{-1} \pmod{\varphi(n)}$.

The public key is (e, n) and the private key is (d, n).

Reference: RSA in pseudocode. Use this for problems 5–11.

```
KeyGen():
    p, q  <- two distinct large primes
    n     <- p * q
    phi   <- (p - 1) * (q - 1)
    e     <- integer with 1 < e < phi  and  gcd(e, phi) = 1
    d     <- modular_inverse(e, phi)        # via Extended Euclidean
    return  public_key = (e, n),  private_key = (d, n)

Encrypt(m, e, n):
    return  mod_exp(m, e, n)                # m^e mod n

Decrypt(c, d, n):
    return  mod_exp(c, d, n)                # c^d mod n

mod_exp(base, exp, mod):                    # square-and-multiply
    result <- 1
    base   <- base mod mod
    while exp > 0:
        if exp is odd:
            result <- (result * base) mod mod
        base <- (base * base) mod mod
        exp  <- exp >> 1                    # integer divide by 2
    return result
```

5. Generate an RSA key pair with $p = 11$ and $q = 13$. Show your work.
   (a) Compute n and $\varphi(n)$.
   (b) Pick a valid e. (Try $e = 7$ — is it valid?)
   (c) Compute d using the Extended Euclidean algorithm.

Answer

6. In RSA, the value $e = 65537 = 2^{16} + 1$ is overwhelmingly the most common choice for the public exponent in real systems. Give two reasons why this choice is convenient. (Hint: think about its binary representation, and what that means for modular exponentiation by squaring.)

   Answer

Encryption and Decryption

To encrypt a message m (with $0 \leq m < n$):

$$c = m^e \bmod n.$$

To decrypt a ciphertext c:

$$m = c^d \bmod n.$$

7. Using your key pair from problem 5 ($p = 11, q = 13$), encrypt the message $m = 9$. Then decrypt the ciphertext to verify you recover $m = 9$.

   Answer

8. The naive way to compute $m^e \bmod n$ is to multiply m by itself $e - 1$ times, reducing mod n each step. This is $\Theta(e)$ multiplications. With $e = 65537$ and 2048-bit n, that is roughly 65,000 multiplications of 2048-bit numbers — still fast, but not the optimal approach.

   Modular exponentiation by repeated squaring computes $m^e \bmod n$ in $\Theta(\log e)$ multiplications.

   (a) Write pseudocode for repeated squaring (the "square-and-multiply" algorithm).

   (b) Trace your algorithm to compute $7^{13} \bmod 23$.

   (c) For a b-bit modulus n and b-bit exponent e, what is the asymptotic running time of one encryption (or decryption) using schoolbook multiplication?

   Answer

9. **Why decryption works.** Euler's theorem states that if $\gcd(m, n) = 1$ then

$$m^{\varphi(n)} \equiv 1 \pmod{n}.$$

Use this to prove that $c^d \equiv m \pmod{n}$ for any m with $\gcd(m, n) = 1$. (Hint: write $ed = 1 + k\varphi(n)$ for some integer k, then expand m^{ed}.)

Answer

Breaking RSA with Bad Prime Choices

The security of RSA rests entirely on the hardness of factoring n. If an attacker can factor $n = pq$, they can compute $\varphi(n) = (p-1)(q-1)$ and then recover d from the public e exactly the way the legitimate user did.

10. You intercept a public key $(e, n) = (7, 35)$ and a ciphertext $c = 12$.

    (a) Factor n to recover p and q.

    (b) Compute $\varphi(n)$.

    (c) Compute the private exponent d.

    (d) Decrypt c.

    Answer

11. **Fermat's factorization.** If p and q are close together, $n = pq$ can be written as a difference of squares $n = a^2 - b^2 = (a-b)(a+b)$ where a is just slightly larger than $\sqrt{n}$. The attack is:

    Start with $a = \lceil \sqrt{n} \rceil$.
    While $a^2 - n$ is not a perfect square, increment a.
    Then $p = a - b$ and $q = a + b$, where $b = \sqrt{a^2 - n}$.

    Use Fermat's method to factor $n = 1517$. How many iterations did it take?

    Answer

Big Picture: Complexity and Security

12. Fill in the table below summarizing what is "easy" (polynomial in the bit-length b of n) and what is believed to be "hard" (super-polynomial) for RSA with b-bit modulus.

Operation	**Best known runtime**	**Easy or hard?**
Encrypt $m^e \bmod n$		
Decrypt $c^d \bmod n$		
Generate a b-bit prime		
Factor n (classical, GNFS)		

Answer

13. **CRT decryption (real-world optimization).** Real RSA implementations almost always speed up decryption using the Chinese Remainder Theorem. Instead of computing $m = c^d \bmod n$ directly, precompute $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$, then:

    $m_p = c^{d_p} \bmod p$

    $m_q = c^{d_q} \bmod q$

    Combine m_p and m_q via CRT to recover $m \bmod n$.

    (a) Using your key from problem 5 ($p = 11$, $q = 13$) and the ciphertext you computed in problem 7, perform CRT decryption and verify you get the same m.

    (b) Assume schoolbook multiplication, so multiplying two b-bit numbers costs $\Theta(b^2)$. Naive decryption $c^d \bmod n$ runs in $\Theta(b^3)$ time. Show that CRT decryption runs in $\Theta(b^3/4)$ time — a constant-factor speedup of roughly $4\times$.

    (c) Why doesn't this trick help with encryption? (Hint: who has access to p and q, and who only has n?)

Answer

14. In practice, RSA is almost never used to encrypt large files directly. Instead, RSA is used to encrypt a short symmetric key (e.g., a 256-bit AES key), and AES is used to encrypt the bulk data. Give an algorithmic reason why this hybrid approach is preferred.

    Answer