EECS 376: Foundations of Computer Science **Due 8:00pm, April 23**
University of Michigan, Winter 2024

Homework 11

(accepted until 9:59 pm, no credit after)

This homework is due 6 days later than usual, on *Tuesday 4/23* at 8pm. This is because some of the questions (labeled below) rely on the lectures on 4/15 and 4/17.

This homework has 7 questions, for a total of 100 points and 0 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't turn in the previous homework, then (1) state that you didn't turn it in, and (2) pick a problem that you think is particularly challenging from the previous homework, and explain the answer in your own words. You may reference the answer key, but your answer should be in your own words.

Solution:

2. **Short-answer potpourri.**

(5 pts) (a) Suppose that Alice's bit string (a binary number) is $x = 100000100$ and Bob's bit string is $y = 10010001$. List, with justification, all the "bad" primes p for which the fingerprinting protocol from class will *fail* to detect that $x \neq y$. (Recall that x and y are interpreted as integers written in base two, with digits written from most to least significant.)

Solution: We have $x = (100000100)_2 = 260$ and $y = (10010001)_2 = 145$. So $x - y = 115$. The fingerprinting protocol fails for exactly those primes p where $x = y \bmod p$, i.e., p divides $x - y$. Since the prime factorization of $x - y$ is $115 = 5 \cdot 23$, the "bad" primes are 5 and 23.

(5 pts) (b) In class we saw that 2 is not a generator of $\mathbb{Z}_7^*$, but 3 is. State, with justification, all the other elements of $\mathbb{Z}_7^*$ that are generators of $\mathbb{Z}_7^*$.

Solution: We list the powers (modulo 7) of each element of $\mathbb{Z}_7^*$ to identify which ones are (and aren't) generators.

The powers of 1 are $1, 1, 1, \ldots$ (repeating), so 1 is not a generator.

The powers of 2 are $2, 4, 1, 2, 4, 1, \ldots$ (repeating), so 2 is not a generator.

The powers of 3 are $3, 2, 6, 4, 5, 1, \ldots$ (repeating), so 3 is a generator.

The powers of 4 are $4, 2, 1, 4, 2, 1, \ldots$ (repeating), so 4 is not a generator.

The powers of 5 are $5, 4, 6, 2, 3, 1, \ldots$ (repeating), so 5 is a generator.

The powers of 6 are $6, 1, 6, 1, \ldots$ (repeating), so 6 is not a generator.

We make a few observations about these. Notice that $4, 5, 6$ are equivalent, modulo 7, to $-3, -2, -1$ (respectively), so their powers are the same, except for alternating signs. For example, the powers of $6 \equiv -1 \pmod 7$ are $1, -1, 1, -1, \ldots$, which are the powers of 1 with alternating signs. Even though 2 is not a generator because it has only three distinct powers, $5 \equiv -2 \pmod 7$ is a generator because the alternating signs yield six distinct powers. The same goes for 4 and $3 \equiv -4 \pmod 7$.

Also notice that $4 \equiv 2^2 \pmod 7$, so the powers of 4 are just every other power of 2. Similarly, $1 \equiv 2^3 \pmod 7$, so the powers of 1 are just every third power of 2. In general, if a number is not a generator for some modulus, then no power of that number is a generator either.

Finally, for any element of $\mathbb{Z}_7^*$, its number of distinct powers—called the "order" of the element—is a divisor of 6. This holds in general for any prime p: any element of $\mathbb{Z}_p^*$ has an order that divides $p - 1$. This is a special case of what is called Lagrange's Theorem.

(5 pts) (c) *This part relies on material from the lecture on 4/15.*

On the distant planet of Arrakis, every month has exactly 35 days (numbered from zero). The Bene Gesserit's plans are measured in *centuries*. One month, on the 0th day, they set a plan in motion that, exactly 3^{28} days later, will finally defeat the tyrannical Emperor Leto (the son of the Kwisatz Haderach, of course). Determine the day of the month when this will occur. Do not use any electronic computations, use as little by-hand calculation as you can, and show your work.

Solution: Since each month has 35 days, we want to know the value of $3^{28} \bmod 35$. Observe that $35 = pq$ where $p = 5$ and $q = 7$ (or vice versa), so $(p-1)(q-1) = 24$. Hence, by Euler's Theorem, $3^{25} \equiv 3 \pmod{35}$, so $3^{28} \equiv 3^4 \equiv 9^2 \equiv 81 \equiv 11 \pmod{35}$. Emperor Leto will be defeated on day 11 of some month (counting from zero).

Alternatively, by using a good deal more calculation (for partial credit), we could compute the answer using repeated squaring. We have that $3^{28} \equiv 3^4 \cdot 3^8 \cdot 3^{16} \pmod{35}$. By the above, $3^4 \equiv 11 \pmod{35}$, so $3^8 \equiv 11^2 \equiv 121 \equiv 16 \pmod{35}$, so $3^{16} \equiv 16^2 \equiv 256 \equiv 11 \pmod{35}$. Multiplying these and reusing the calculations, we get that $3^{28} \equiv 11^2 \cdot 16 \equiv 16^2 \equiv 11 \pmod{35}$, as above.

(5 pts) (d) *This question relies on material from the lecture on 4/17.*

Pam shows Creed two pictures and claims that they are different, but Creed cannot see any difference between them. Pam wants to convince Creed that they really are different, but without revealing what the difference is. How can Creed test Pam so that if the pictures really are different, Pam will pass the test while revealing nothing more than this fact to Creed; and if the pictures are identical, Pam will fail the test with probability at least 99%? (Just describe the test; you do not need to prove any of its properties.)

Solution: This essentially the "pen test" from lecture. Creed and Pam agree to call one picture A and the other one B. Creed randomly shuffles the two pictures behind his back (out of Pam's sight), remembering which one is which. Then, he challenges Pam to identify the pictures by name. They repeat this at least 7 times. If Pam correctly identifies the pictures every time, she passes the test; otherwise, she fails.

Though we did not ask for any analysis, here it is. If the pictures really are different (and Pam can tell them apart), then she can easily pass the test. But if the pictures are the identical, then due to Creed's random shuffling, Pam's probability of guessing correctly in each single iteration is only $1/2$, so by independence, for the entire test it is $(1/2)^7 = 1/128 < 1\%$. Finally, this test conveys zero knowledge to Creed, apart from the fact that the pictures are different: when Creed shuffles the pictures, he knows which one is which, and this is the only thing that Pam tells him. In other words, everything that Pam tells Creed, he could have generated himself! (However, the test is still convincing to Creed because he actually interacted with Pam, and she would have been very unlikely to pass the test if the pictures were different.)

3. **Near median.**

In the Quicksort algorithm, we would like to use a pivot that is "nearly" the median value of the array, because it partitions the array into two roughly equal-sized parts for recursive sorting. In this problem, we will develop a randomized algorithm for finding a near-median in *constant* time (with good probability), independent of the input size.

Let S be an array of n orderable elements, which are distinct without loss of generality We say that $a \in S$ has *rank* $\text{rank}(a) = k$ if a is the kth smallest element in S. Given input S, the goal is to find an element having rank between $2n/5 + 1$ and $3n/5$ (inclusive). For simplicity, *we assume that n is divisible by 5* (e.g., we can "pad" the end of the array with up to four extra elements).

(5 pts) (a) Consider the simplest algorithm:

Choose an element $a \in S$ uniformly at random.

Prove that $\Pr[\text{rank}(a) \le 2n/5] = 2/5$ and $\Pr[\text{rank}(a) > 3n/5] = 2/5$. Conclude that this algorithm is correct with probability at least $1/5$.

Solution: We have

$$\Pr[\text{rank}(a) \le 2n/5] = \frac{2n/5}{n} = 2/5$$

because there are exactly $2n/5$ elements of rank at most $2n/5$. We have

$$\Pr[\text{rank}(a) > 3n/5] = \frac{2n/5}{n} = 2/5$$

because there are exactly $2n/5$ elements of rank strictly more than $3n/5$.

The algorithm is wrong if and only if $\text{rank}(a) \leq 2n/5$ or $\text{rank}(a) > 3n/5$. Both of these events happens with probability $2/5$. As the two events are disjoint, one of these two events with probability $4/5$. So the probability that the algorithm is correct is exactly $1 - 4/5 = 1/5$.

(7 pts) (b) The previous result is not too bad, but its probability of correctness isn't so good. We would like to improve the success probability to at least $1 - \delta$, for any desired $\delta > 0$. Consider the following algorithm:

> Uniformly and independently sample k elements from S *with replacement* (i.e., with repetitions allowed), for some *odd* k to be determined below. Return the median of these samples. (Because k is odd, the median is uniquely determined.)

Let X^{tiny} and X^{huge} denote the number of sampled elements whose ranks are, respectively, at most $2n/5$, and more than $3n/5$. Prove that the algorithm's output is correct if both $X^{tiny}, X^{huge} < k/2$.

Solution: Because k is odd, the output value is the $\lceil k/2 \rceil$th smallest, and $\lceil k/2 \rceil$th largest, value among the samples.

If $X^{tiny} < k/2 < \lceil k/2 \rceil$, then the returned value must have rank strictly more than $2n/5$. Similarly, if $X^{huge} < k/2 < \lceil k/2 \rceil$, then the returned value must have rank at most $3n/5$. So, if both hold, the output value has rank between $2n/5 + 1$ and $3n/5$ (inclusive), hence the output is correct.

(5 pts) (c) Determine, with proof, a *constant* value of k (which may depend on δ, but not on n) so that $\Pr[X^{tiny} \geq k/2] \leq \delta/2$ and $\Pr[X^{huge} \geq k/2] \leq \delta/2$.

Hint: Set up and use an appropriate Chernoff bound.

Solution: Let X_i^{tiny} be the indicator that the rank of the ith sample is at most $2n/5$. Note that $\mathbb{E}[X_i^{tiny}] = \Pr[X_i^{tiny} = 1] = 2/5$ by the first part of the question. Observe that $X^{tiny} = \sum_{i=1}^{k} X_i^{tiny}$, so

$$\mathbb{E}[X^{tiny}] = \sum_{i=1}^{k} \mathbb{E}[X_i^{tiny}] = 2k/5.$$

Since $X_i^{tiny} \in [0, 1]$ are independent, we can apply the small-deviation, upper-tail

Chernoff bound with $\lambda = 1/4$ to get

$$\begin{aligned}\Pr[X^{tiny} \geq k/2] &= \Pr[X^{tiny} \geq (1+1/4)\,\mathbb{E}[X^{tiny}]] \\ &\leq \exp(-(1/4)^2 \cdot 2k/15) \\ &= \exp(-k/120)\ .\end{aligned}$$

We want this to be at most $\delta/2$, so by taking natural logs and solving for k, we get that any $k \geq 120\ln(2/\delta)$ suffices for the desired conclusion. For example, we can use the first odd integer above this lower bound, which is a constant that depends only on δ, not n.

Symmetrically, let X_i^{huge} be the indicator random variable that the rank of the ith sample is greater than $3n/5$. Then $\mathbb{E}[X_i^{huge}] = \Pr[X_i^{huge} = 1] = 2/5$ by the first part of the question. Then, for the same value of k as above, we can show that

$$\Pr[X^{tiny} \geq k/2] \leq \exp(-k/120) \leq \delta/2$$

in exactly the same way.

(5 pts) (d) Conclude that the algorithm is correct with probability at least $1 - \delta$.

Solution: By the second part of the question, we have that

$$\Pr[\text{algorithm is incorrect}] \leq \Pr[X^{tiny} \geq k/2 \text{ or } X^{huge} \geq k/2].$$

By the union bound, we have that

$$\Pr[X^{tiny} \geq k/2 \text{ or } X^{huge} \geq k/2] \leq \Pr[X^{tiny} \geq k/2] + \Pr[X^{huge} \geq k/2] \leq 2 \cdot \delta/2 = \delta.$$

Hence, the algorithm is wrong with probability at most δ. So, it is correct with probability at least $1 - \delta$.

4. **Diffie–Hellman.**

Daphne and Julie are writing the solutions to Homework 11. Daphne has a draft of the solutions on her laptop and wants to send them confidentially to Julie. Since they have not had a private moment to choose a shared secret key together, they decide to use the Diffie-Hellman protocol to establish one over the public network.

(5 pts) (a) Suppose they use the Diffie–Hellman protocol with the small prime $p = 19$ and generator $g = 2$ for $\mathbb{Z}_p^*$. Daphne chooses secret exponent $a = 7$, and Julie chooses $b = 11$. Derive the values that they send each other and the secret key that each one computes, to conclude that they both compute the same value.

You may use a calculator for basic arithmetic (multiplication and division), but beyond that you should use the efficient algorithms that a computer would employ. Show your work (repeated squaring etc.), but you can omit the details of modular reductions. Keep numbers small by reducing modulo p where appropriate, and use negative number where they help.

Solution: Daphne sends $g^a \bmod p$, which is $2^7 \bmod 19$. We can calculate this using repeated squaring as follows:

$$\begin{aligned}
2^1 &\equiv 2 \pmod{19}\\
2^2 &\equiv 4 \pmod{19}\\
2^4 &\equiv 16 \equiv -3 \pmod{19}\\
2^7 &\equiv 2^4 \cdot 2^2 \cdot 2 \equiv (-3) \cdot 2 \cdot 4 \equiv -24 \equiv -5 \equiv 14 \pmod{19}\ .
\end{aligned}$$

Julie sends $g^b \bmod p$, which is $2^{11} \bmod 19$. We will again use repeated squaring, but we can reuse the work we already did to calculate what Daphne sent.

$$2^{11} \equiv 2^7 \cdot 2^4 \equiv 14 \cdot (-3) \equiv -42 \equiv -4 \equiv 15 \pmod{19}\ .$$

Using repeated squaring, the key that Julie computes is Daphne's public message raised to Julie's secret exponent, modulo p:

$$\begin{aligned}
(2^7)^{11} &\equiv (-5)^{11} \equiv 25^5 \cdot (-5) \pmod{19}\\
&\equiv 6^5 \cdot (-5) \pmod{19}\\
&\equiv (6^2)^2 \cdot 6 \cdot (-5) \pmod{19}\\
&\equiv (-2)^2 \cdot 6 \cdot (-5) \pmod{19}\\
&\equiv 24 \cdot (-5) \pmod{19}\\
&\equiv -5 \cdot 5 \equiv 13 \pmod{19}\ .
\end{aligned}$$

Similarly, the secret key that Daphne computes is Julie's public message raised to Daphne's secret exponent, modulo p:

$$\begin{aligned}
(2^{11})^7 &\equiv (-4)^7 \pmod{19}\\
&\equiv 16^3 \cdot (-4) \pmod{19}\\
&\equiv (-3)^3 \cdot (-4) \pmod{19}\\
&\equiv 27 \cdot 4 \pmod{19}\\
&\equiv 8 \cdot 4 \equiv 13 \pmod{19}\ .
\end{aligned}$$

Thus, both Daphne and Julie are able to compute the same shared secret key, which is 13.
(Though this was not asked for, if we wish, we can also confirm by repeated squaring that $g^{ab} \equiv 2^{7\cdot 11} \equiv 2^{77} \equiv 13 \pmod{19}$.)

(12 pts) (b) After realizing that the parameters from the previous part are much too small (since they are breakable by hand), Daphne and Julie decide to use the Diffie–Hellman protocol with a different, huge prime p and generator g of $\mathbb{Z}_p^*$. As specified by the protocol, Daphne chooses secret $a \in \mathbb{Z}_p^*$ uniformly at random, and sends $x = g^a \bmod p$ to Julie. Similarly,

Julie chooses secret $b \in \mathbb{Z}_p^*$ uniformly at random, and sends $y = g^b \bmod p$ to Daphne.

Unbeknownst to them, there is a student Malcolm in the middle of their network, who is able to intercept *and undetectably replace* what they send over the network with other values of his choice. (In class we considered only an *eavesdropper* Eve who can merely read all the network traffic; this Malcolm is more powerful.)

Specifically, Malcolm chooses a secret $c \in \mathbb{Z}_p^*$ himself, and sends $z = g^c \mod p$ to both Daphne and Julie in place of their messages to each other. That is, Daphne thinks that Malcolm's message came from Julie, and Julie thinks that Malcolm's message came from Daphne.

1. Give an expression for the key that Daphne k_D computes, and the key k_J that Julie computes, in terms of the secret exponents a, b, c and the public values p, g.

   Solution: Following the protocol, Daphne and Julie each raise the value they received (from Malcolm) to the power of their own secret exponents, modulo p. So, Daphne computes $k_D = z^a \bmod p = (g^c)^a \bmod p = g^{ac} \bmod p$, and Julie computes $k_J = z^b \bmod p = (g^c)^b \bmod p = g^{bc} \bmod p$.

2. Daphne and Julie work on the homework solutions by encrypting their messages using the keys they computed in the previous part (which they believe are the same key). Malcolm knows the encryption scheme that Daphne and Julie are using. Describe how Malcolm can decrypt all of Daphne and Julie's communications without being detected, i.e., so that they believe they are communicating confidentially, and when they decrypt the ciphertexts they receive, they get exactly the messages that were intended for them.

   A valid solution should show the following:

   - how Malcolm can successfully decrypt all of the ciphertexts that Daphne and Julie send to each other;
   - how Malcolm can avoid detection by replacing the ciphertexts that Daphne and Julie send with ones that will yield the intended messages when they decrypt (using what they believe to be their shared key).

   Solution: Importantly, Malcolm can efficiently compute both k_D and k_J since he knows c and intercepted $x = g^a \bmod p$ from Daphne and $y = g^b \bmod p$ from Julie. Specifically, Malcolm calculates $k_D \equiv x^c \equiv (g^a)^c \equiv g^{ac} \pmod p$ and $k_J \equiv y^c \equiv (g^b)^c = g^{bc} \pmod p$. Thus, when Daphne sends Julie a ciphertext that encrypts a message under k_D, Malcolm can intercept the ciphertext, decrypt and read it, reencrypt it with k_J, and send the result to Julie. He can do the same process in reverse when Julie sends an encrypted message to Daphne. Daphne and Julie will not notice that anything is wrong, because they each correctly decrypt the actual intended message from the other one.

5. **Diffie–Hellman as public-key encryption.** *This question relies on some material from the lecture on 4/15.*

In class we said that Diffie–Hellman is an interactive protocol for two parties to establish a shared secret over a public channel, but it was not until RSA that a full public-key encryption scheme was known. However, it turns out that, by viewing the Diffie–Hellman protocol in the right way, we can actually use it for public-key encryption! (This perspective was discovered several years after RSA was proposed, and is widely used in practice now.)

Let p be a huge prime and g be a generator of $\mathbb{Z}_p^*$, both public. To generate a public key, Alice simply performs her "side" of the DH protocol: she chooses a secret exponent $a \in \mathbb{Z}_p^* = \{1, \ldots, p-1\}$ uniformly at random, and publishes $x = g^a \bmod p$ as her public key.

(6 pts) (a) Suppose that somebody (Bob, or Charlie, or Dan, etc.) wants to send Alice a confidential message m using her public key (and the other public data). Describe what the sender should do to accomplish this, and how Alice can compute the intended message from what is sent to her. (The algorithms run by Alice and the sender should be efficient.) Also give some brief intuition for why the method is plausibly secure against an eavesdropper. (You may assume that the DH protocol itself is secure, and that the sender has Alice's genuine public key.)

Hint: Have the sender perform the other "side" of the DH protocol, and also compute and send something more. How can the message be hidden so that only Alice can compute it using her secret key?

Solution: The sender performs the other "side" of the DH protocol: he chooses a secret exponent $b \in \mathbb{Z}_p^* = \{1, \ldots, p-1\}$ uniformly at random, and computes $y = g^b \bmod p$. Then, using Alice's public key x, he also computes the shared DH secret key $k = x^b \bmod p$; recall that this is $k = g^{ab} \bmod p$ because $x = g^a \bmod p$. He then encrypts the message m with k to get a ciphertext c, e.g., using k as a one-time pad (or more generally, using k as the key in a symmetric-key encryption scheme). Finally, he sends the ciphertext (y, c) to Alice.

To decrypt the ciphertext (y, c) using her secret key a, Alice treats y just as in the DH protocol, using it and her secret exponent a to compute the shared DH secret key $k = y^a \bmod p$; recall that this is the same $k = g^{ab} \bmod p$ the sender computed. She then uses k to decrypt c and obtain the message.

For security, the intuition is that an eavesdropper sees exactly the same public values as in the DH protocol—the fixed public values p, g, the public key $x = g^a \bmod p$, and the sender's public $y = g^b \bmod p$—along with the c that hides m under the shared DH secret key k. Under the DH assumption, an eavesdropper cannot compute k from these public values, so the message is well hidden by c.

To be more careful, we should not use k itself as a one-time pad, because while the eavesdropper cannot compute k entirely (under the DH assumption), the eavesdropper might be able to compute some parts of k, i.e., k might not appear entirely uniform. So, we should first "extract" some uniform-looking value from k, and use that as the one-time pad or other symmetric encryption key. Take EECS 388/475/575 for further details!

(5 pts) (b) In class we mentioned a security problem with the oversimplified RSA encryption scheme, as we presented it. Briefly describe the problem, and explain why the DH-style encryption scheme from the previous part does *not* suffer from this problem.

Solution: The problem is that the encryption algorithm is *deterministic*, so encrypting the same message (under the same public key) always yields the same ciphertext. This allows an eavesdropper to detect repeated messages, or even do frequency analysis if the set of actually encrypted messages is not very large.

The above DH-style encryption does not have this issue, because the encryption process is meaningfully *randomized*: every time we encrypt a (possibly repeated) message, we choose a fresh random exponent $b \in \mathbb{Z}_p^*$, compute $y = g^b \bmod p$ and $k = x^b \bmod p$, and hide the message using k. Each choice of b is almost certain to be different (because p is enormous), so each time we encrypt we get different values of y and k, and therefore different ciphertexts (even if the messages are the same).

6. **RSA.** *This question relies on material from the lecture on 4/15.*

Here you will run the RSA signature scheme on some small "toy" parameters. You may use a calculator for basic arithmetic (multiplication and division), but beyond that you should use the efficient algorithms that a computer would employ (where applicable), and show your work. Keep the numbers small by reducing modulo n where appropriate, and use negative numbers where they help.

Let $n = 77$ be the public modulus and $e = 7$ be the public exponent.

(5 pts) (a) Compute the private exponent d using the extended Euclidean algorithm (which we have repeated below for convenience).

Input: integers $x \geq y \geq 0$, not both zero
Output: a triple (g, a, b) of integers where $g = \gcd(x, y)$ and $ax + by = g$

```
1: function ExtendedEuclid($(x, y)$)
2:     if $y = 0$ then
3:         return $(x, 1, 0)$                    ▷ Base case: $1x + 0y = \gcd(x, y) = x$
4:     else
5:         Write $x = qy + r$ for an integer $q$, where $0 \leq r < y$
6:         $(g, a', b') \leftarrow$ ExtendedEuclid($(y, r)$)
7:         $a \leftarrow b'$
8:         $b \leftarrow a' - b'q$
9:         return $(g, a, b)$
```

Solution: We can factor the modulus as $n = 7 \cdot 11$ using brute force (trying to divide it by various primes), or just by inspection. That is, $p = 7$ and $q = 11$ (or vice versa). From here we can compute $\phi(n) = (p-1)(q-1) = (7-1)(11-1) = 60$. We now need to find a d such that $7 \cdot d \equiv 1 \pmod{60}$. For this we will use the extended Euclidean algorithm on $x = 60, y = 7$:

input		division		rec ans		output		
x	y	q	r	a'	b'	g	a	b
60	7	8	4	-1	2	1	2	-17
7	4	1	3	1	-1	1	-1	2
4	3	1	1	0	1	1	1	-1
3	1	3	0	1	0	1	0	1

From this, we see that $2 \cdot 60 - 17 \cdot 7 = 1$, so $-17 \cdot 7 \equiv 1 \pmod{60}$. We conclude that $d = -17 \bmod 60 = 43$.

(5 pts) (b) Next, compute the correct signature for $m = 3$.

Solution: To sign m, we raise m to the power of the private key d, modulo n. This gives the signature $s = m^d \bmod n$.
To start, we calculate powers $m^{2^i} \pmod{n}$ by repeated squaring for later fast modular exponentiation:

$$\begin{aligned}
3^0 &\equiv 1 \pmod{77}\\
3^1 &\equiv 3 \pmod{77}\\
3^2 &\equiv 9 \pmod{77}\\
3^4 &\equiv 9^2 \equiv 81 \equiv 4 \pmod{77}\\
3^8 &\equiv 4^2 \equiv 16 \pmod{77}\\
3^{16} &\equiv 16^2 \equiv 256 \equiv 25 \pmod{77}\\
3^{32} &\equiv 25^2 \equiv 625 \equiv 9 \pmod{77}\ .
\end{aligned}$$

Now we can compute

$$\begin{aligned}
m^d &\equiv 3^{43} \pmod{77}\\
&\equiv 3^{32} \cdot 3^8 \cdot 3^2 \cdot 3^1 \pmod{77}\\
&\equiv 9 \cdot 16 \cdot 9 \cdot 3 \pmod{77}\\
&\equiv (9 \cdot 9) \cdot 48 \pmod{77}\\
&\equiv 4 \cdot 48 \pmod{77}\\
&\equiv 38 \pmod{77}\ .
\end{aligned}$$

So our signature is $s = 38$.

(5 pts) (c) Show how one would verify, using only the public information n, e, that the signature s you found in the previous part is correct for $m = 3$.

Solution: To verify that signature $s = 38$ is correct, we raise s to the power of the public key e, modulo n. If this results in m, then we conclude that the signature is

valid for this message and public key.
To start, we calculate powers $s^{2^i} \pmod{77}$:

$$\begin{aligned}38^1 &\equiv 38 \pmod{77}\\ 38^2 &\equiv 1444 \equiv 58 \equiv -19 \pmod{77}\\ 38^4 &\equiv (-19)^2 \equiv 361 \equiv 53 \equiv -24 \pmod{77}\ .\end{aligned}$$

Next, we raise s to the eth power, modulo n:

$$\begin{aligned}s^e &\equiv 38^7 \pmod{77}\\ &\equiv 38^4 \cdot 38^2 \cdot 38 \pmod{77}\\ &\equiv (-24) \cdot (-19) \cdot 38 \pmod{77}\\ &\equiv 3 \pmod{77}.\end{aligned}$$

Because $s^e \equiv m \pmod{n}$, we conclude that the signature $s = 38$ is valid for m.

(5 pts) (d) Now suppose that we had a huge (non-"toy") RSA modulus n that was, say, 1000 bits or more in size (i.e., $n \geq 2^{1000}$). Which algorithm(s) that you used in the previous parts are *feasible* for an ordinary computer to perform for this size, and which would be *infeasible* for even a state-of-the-art supercomputer to perform? Explain your reasoning.

Solution:

Algorithms Used	Feasible	Part(s) Used
Factorization	No	a
Extended Euclidean	Yes	a
Fast Modular Exponentiation	Yes	b,c

Brute-force factorization is not feasible for such huge numbers, because factoring $n \approx 2^{1000}$ in this way requires attempting to divide n by about $\sqrt{n} \approx 2^{500}$ candidate divisors. This is (far) more than the number of elementary particles in the universe! In general, trial division on a b-bit number $n \approx 2^b$ takes about $2^{b/2}$ attempts, which is exponential in the size of the number (its bit length).

As we saw earlier in the term, the (extended) Euclidean algorithm is efficient: its running time is a (small) polynomial in the bit length of the input numbers. So, it can feasibly be run on an ordinary computer with 1000-bit numbers.

As we have also seen in lecture and discussion, fast modular exponentiation is efficient: the number of modular multiplications it performs is linear in the bit length of the exponent, and modular multiplication is also efficient.