C&O 487/687: Assignment #4 Solutions

Due date: November 14, 2024 (Thursday)

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.

Throughout this assignment, you make use of the following facts about number theory:

- Fact 1: If $a, b \in \mathbb{Z}_p^*$ and p is prime, then ab is a square modulo p if and only if either a and b are both squares modulo p, or a and b are both non-squares modulo p.
- Fact 2: If p is prime, then exactly half the elements in $\mathbb{Z}_p^*$ are squares.
- Fact 3: There exists an efficient algorithm for determining if an element a has a square root modulo p.
- Fact 4: There exists an efficient algorithm for computing square roots modulo p, if it exists.

1. [9 marks] **Hybrid encryption.**

   Recently, the University of Waterloo LEARN server was hacked by the Really Super Awesome Association of Engineering Students (RSA-AES), who are continuing their maniacal plot for world domination. The next phase of their evil plan is to ensure that engineering students have better grades and hence get better co-op jobs than math students, so they hacked into the LEARN server and encrypted this CO 487 assignment in hopes of causing you to get a lower mark in this course. Luckily for you, they made some mistakes during encryption because they did not take CO 487.

   Your first task for this assignment is to decrypt the encrypted assignment PDF so that you can actually do the rest of your assignment.[1]

   I managed to capture some logs showing some of the Python code they used to encrypt the assignments, which you might find helpful in figuring out how to break their encryption. This is in the file `a4q1_attacker_log.py` on LEARN.

   You might have to do some number theory operations in Python as you try to solve this question. Here are a few functions that might be helpful:

   - `pow(x, y, z)`: computes $x^y \bmod z$
   - `isprime(x)`: determines whether the integer x is prime; see `https://docs.sympy.org/latest/modules/ntheory.html#sympy.ntheory.primetest.isprime`
   - `nextprime(x)`: returns the next prime number larger than x; see `https://docs.sympy.org/latest/modules/ntheory.html#sympy.ntheory.generate.nextprime`
   - `factorint(x)`: attempts to factor the integer x, and if successful returns the factors and their multiplicities; see documentation at `https://docs.sympy.org/latest/modules/ntheory.html#sympy.ntheory.factor_.factorint`
   - `mod_inverse(x, z)`: computes $x^{-1} \bmod z$
   - `decimal.getcontext().prec = 100; decimal.Decimal(x).sqrt()`: compute high-precision square roots of real numbers

   `pow` is included with Python by default. To get `decimal`, you have to put `import decimal` at the top of the program. To get the rest of the functions, you have to install an additional library called `sympy` (try running `pip3 install sympy`) and then importing those functions into your program using the following source code:

[1] You can also download the decrypted version of (most of) the assignment from LEARN if you want to get started on other questions, but the decrypted version from LEARN is missing a part (worth a few marks) that can only be obtained by decrypting your own customized encrypted assignment.

```
from sympy import mod_inverse
from sympy.ntheory import isprime, nextprime
```

Note that all of the required packages are installed on the UW math Jupyter notebook server `https://jupyter.math.uwaterloo.ca`, but you need to make sure you use the "Python 3 extra" kernel.

(a) [4 marks] By inspecting `a4q1_attacker_log.py`, describe 4 cryptographic mistakes made by the hackers and what you would do differently.

Solution.

- RSA keys are 1024-bit; should be at least 2048-bit
- Random number generation is done using Python's insecure random number generator `random`; should use `secrets` or `os.urandom`
- The random number generator is seeded with a low-entropy timestamp
- q is not a random prime, it is the next prime after p
- e is selected to be coprime with n, it should be selected to be coprime with $\varphi(n)$
- AES encryption is using ECB mode

(b) [4 marks] Describe the procedure you used to decrypt the file. In your description, focus on the main cryptographic break you found. Submit any code you write through Crowdmark, as you would a normal assignment – as a PDF or screenshot. We will be reading it, but not executing it.

Solution. There are two ways to recover the factors of the RSA modulus n:

1. Compute the real-number square root of n, then consecutively search integers down or up from $\sqrt{n}$ for the closest primes which will be p and q.
2. Search through the low-entropy seed space, then re-seed the random number generator and see if any of the primes we produce are the same.

Then we can compute $\varphi(n) = (p-1)(q-1)$, and then compute the decryption exponent $d = e^{-1} \mod \phi(n)$. After that, we just decrypt by reversing the remaining symmetric key cryptography operations.

```
import base64
import datetime
import decimal
import json
import random
from sympy import mod_inverse
from sympy.ntheory import isprime, nextprime
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

def bytes2string(b):
    return base64.urlsafe_b64encode(b).decode('utf-8')

def string2bytes(s):
    return base64.urlsafe_b64decode(s.encode('utf-8'))

# helper functionfor decryption method 2 below
def find_seed_match(n):
    bitlength = 512
    year = 2024
    # iterate through all possible month/day/hour/minute seeds
    for month in [11]: # guess that the assignment was created in November to save time
        for day in range(1, 32):
            for hour in range(0, 24):
                print(f"Trying␣y-m-d-h␣=␣{year}-{month}-{day}-{hour}...")
                for minute in range(0, 60):
                    seed = datetime.datetime(year, month, day, hour, minute).strftime("%Y-%m-%d␣%H:%M")
                    random.seed(seed)
                    p = random.randint(2 ** (bitlength - 1), 2 ** bitlength)
                    while not(isprime(p)): p = random.randint(2 ** (bitlength - 1), 2 ** bitlength)
                    q = nextprime(p)
                    if n == p * q:
```

```
                return (p, q)
    assert False

def do_decryption():

    # Read and parse the JSON data structure
    with open("encrypted_assignment.json.txt", 'r') as fh:
        input = json.loads(fh.read())
    n = input["n"]
    e = input["e"]

    # method 1: real square root then find primes
    decimal.getcontext().prec = 200
    r = decimal.Decimal(n).sqrt()
    p = int(r)
    while not(isprime(p)):
        p = p - 1
    q = n // p
    assert p * q == n

    # method 2: search through the RNG seed
    (p2, q2) = find_seed_match(n)
    assert p2 == p
    assert q2 == q

    # compute phi(n)
    phin = (p - 1) * (q - 1)

    # compute private key d
    d = mod_inverse(e, phin)

    # decrypt the AES key
    c_1 = input["c_1"]
    aes_key_int = pow(c_1, d, n)
    aes_key = aes_key_int.to_bytes(16, byteorder="big")

    # decrypt the AES ciphertext
    cipher = Cipher(algorithms.AES(aes_key), modes.ECB())
    decryptor = cipher.decryptor()
    aes_ct = string2bytes(input["c_2"])
    plaintext_padded = decryptor.update(aes_ct) + decryptor.finalize()

    # unpad the plaintext
    unpadder = padding.PKCS7(128).unpadder()
    plaintext = unpadder.update(plaintext_padded) + unpadder.finalize()

    # write the decrypted assignment to a file
    with open("a4_decrypted.pdf", 'wb') as fh:
        fh.write(plaintext)

do_decryption()
```

(c) [1 marks] To prove that you successfully decrypted the file, copy the following random number into your solution: 17853

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.

Solution. Your random number was 17853.

2. [9 marks] **Diffie–Hellman and discrete logarithms**

Let G be a cyclic group of prime order q, with generator g; this information is publicly known. Consider the following three computational hardness assumptions:

- **Decisional Diffie–Hellman (DDH):** Given either (g^a, g^b, g^{ab}) or (g^a, g^b, g^c), where a, b, and c are random, distinguish whether you were given a triple of the first form or a triple of the second form.
- **Computational Diffie–Hellman (CDH):** Given g^a and g^b, where a and b are random, compute g^{ab}.
- **Discrete Logarithm Problem (DLP):** Given g^a, where a is random, compute a.

(a) [2 marks] Show that, if the DDH problem is hard, then the CDH problem must also be hard.
Hint: Prove the contrapositive. Assume that there exists an algorithm, $\mathcal{A}$, which can successfully solve CDH, and write an algorithm, $\mathcal{B}$, that has blackbox access to $\mathcal{A}$, which can solve the DDH problem.

Solution. Suppose $\mathcal{A}$ is an adversary which can successfully solve CDH. We write a reduction $\mathcal{B}$, which can solve DDH for inputs (A, B, C), where $A = g^a$, $B = g^b$, and C is either g^{ab} or g^c, as follows:

$\mathcal{B}(A, B, C)$

1 : $C' \leftarrow \mathcal{A}(A, B)$
2 : **if** $C = C'$: **return** opinion that $C = g^{ab}$
3 : **else** : **return** opinion that $C = g^c$

(b) [2 marks] Show that, if the CDH problem is hard, then the discrete logarithm problem must also be hard.
Hint: Prove the contrapositive. Assume that there exists an algorithm, $\mathcal{A}$, which can successfully solve DLP, and write an algorithm, $\mathcal{B}$, that has blackbox access to $\mathcal{A}$, which can solve the CDH problem.

Solution. Suppose $\mathcal{A}$ is an adversary which can successfully solve DLP. We write a reduction $\mathcal{B}$, which can solve CDH for inputs (A, B), where $A = g^a$ and $B = g^b$, as follows:

$\mathcal{B}(A, B)$

1 : $a \leftarrow \mathcal{A}(A)$
2 : **return** B^a

(c) [5 marks] It is not the case that DDH is hard in every group. Suppose our adversary has an oracle, O, which tells them whether a given input is a square modulo p (where p is prime). For instance, if $p = 7$, then $O(2) = true$ since $3^2 \equiv 2 \mod 7$, but $O(3) = false$, since 3 is not a square modulo 7. Write an adversary $\mathcal{A}$, with access to O,[2] which can solve DDH in the group $\mathbb{Z}_p^*$ (for prime p). *Hint: Your adversary does not have to succeed with probability 1; it is enough that they succeed with probability $\frac{1}{2} + x$, where x is non-negligible.*

Solution.

If g is a generator of the entire group $\mathbb{Z}_p^*$, then g cannot be a square, otherwise it would not be able to generate non-squares modulo p, which exist.

Suppose g^a is a square modulo p, namely that $g^a \equiv x^2 \mod p$ for some $x \in \mathbb{Z}_p^*$. Since x is in the group generated by g, there exists $\alpha \in \mathbb{Z}_q$ such that $x \equiv g^\alpha \mod p$. So $g^a = (g^\alpha)^2 = g^{2\alpha}$. Then $g^{ab} = g^{2\alpha b}$ must also be a square.

Similarly, if g^b is a square modulo p, then g^{ab} must also be a square.

Now suppose we are given a tuple (A, B, C) and are asked to determine if it is a real DH triple ($C = g^{ab}$) or random ($C = g^c$). Use our oracle O to check whether each of A, B, and C is a square. If at least one of A or B is a square, but C is not a square, then it cannot be a real DH triple, because of the reasoning in the previous two paragraphs. In these cases, we will be able to recognize random triples with certainty. When (A, B, C) is a random triple, since A, B, and C are chosen uniformly at random, and since by Fact 2 exactly half the elements in $\mathbb{Z}_p^*$ are squares, there are 8 equally likely scenarios: square/non-square for each of the 3 values. We are able to recognize random triples with certainty for three of these cases: (square, square, non-square); (square, non-square, non-square); and (non-square, square, non-square). So we succeed with probability substantially greater than $\frac{1}{2}$.

[2]Giving the adversary such an oracle is not as outlandish as it may seem. The Legendre symbol is an efficiently computable function from elementary number theory which returns 1 if the input is a square modulo p, -1 if the input is not a square modulo p, and 0 if the input is divisible by p. So, in the real world, every adversary has access to O!

(It is okay that we don't win with certainty on all the cases. With a little extra work we can also win on the case (non-square, non-square, square) to get up to half the cases.)
Hence DDH is not hard when the group is the entire multiplicative group $\mathbb{Z}_p^*$.

3. [4 marks] **Backdoored PRG**

A *pseudorandom generator* (PRG) is a deterministic function that expands a short random seed into a longer random looking sequence. We will construct a pseudorandom generator from a deterministic function f that takes as input a state s_{i-1} and produces a new state s_i and outputs $t_i \in S$ for some set S.

To produce a long pseudorandom sequence of ℓ numbers from set S, start with an initial seed s_0, iterate the function as many times as needed by computing

$$(s_i, t_i) \leftarrow f(s_{i-1})$$

for $i = 1, \ldots, \ell$, and output the final output $t_1, \ldots, t_\ell \in S^\ell$

We will design a PRG based off of discrete logarithms. The following information is public:

- A prime number p, an element $g \in \mathbb{Z}_p^*$ of large prime order q, and the the order q.
- Another element h in the subgroup generated by g (in other words, $h = g^u \bmod p$ for some u; but u is *not* given.

Our PRG is based on the following function f, defined as follows:

$$f(s_{i-1}) = (s_i, t_i)$$

where

$$r_i = g^{s_{i-1}} \qquad s_i = g^{r_i} \qquad t_i = h^{r_i}$$

(a) [1 marks] For parameters $p = 167$, $g = 21$, $q = 83$, and $h = 6$, and seed $s_0 = 123$, produce the output for $\ell = 3$ (produce the values t_1, t_2, t_3).
Solution. For these parameters, we obtain

$$(s_1, s_2, s_3) = (57, 124, 152)$$

$$(t_1, t_2, t_3) = (7, 54, 150)$$

(b) [2 marks] It is possible to show that, for well-chosen parameters, this PRG is secure under the assumption that the Diffie–Hellman problem is hard. If, however, the adversary knows the value u such that $h = g^u$, called a backdoor, the PRG is not secure. Suppose the value of t_1 had been published. Explain how, for arbitrary parameters meeting the conditions described, an adversary could use u and t_1 to find s_1, and use this to predict t_2. Justify your response. Verify your attack works for the above examples; you may make use of the backdoor I chose while generating this question: namely, that $6 \equiv 21^{39} \bmod 167$.
Solution. Since $t_i \equiv h^{r_i} \equiv g^{ur_i} \bmod p$ and $s_i \equiv g^{r_i} \bmod p$, it holds that $t_i \equiv s_i^u \bmod p$. Equivalently, $s_i \equiv t_i^{u^{-1} \bmod q} \bmod p$. Once we know t_1, we can use the backdoor to compute s_1, and then we can compute all future outputs $t_2, \ldots$. For $u = 39$ and $q = 83$, we have that $u^{-1} \bmod q = 66$. Given $t_1 = 7$, we can thus compute

$$s_1 \equiv t_1^{u^{-1} \bmod q} \bmod p \equiv 7^{66} \bmod 167 = 57$$

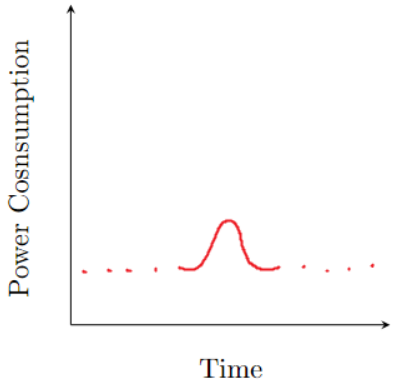
(c) [1 marks] Suggest a countermeasure to the attack of part (b). Your countermeasure might include a modification to the scheme to make it so that even an adversary who knows the backdoor u cannot predict the next output, or your countermeasure might allow for the person who generated the parameters to prove that they could not know the backdoor u, or you might try some other approach.

Solution. Many answers here, the point is just to avoid generating h as a known power of g, so sampling h at random from $\mathbb{Z}_p^*$ would work; or computing h as a hash of a random value (since the hash function should destroy all structure).

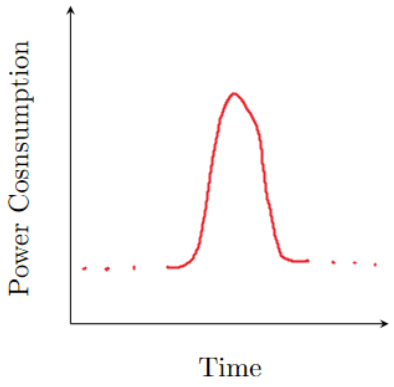
4. [9 marks] **Side channel attack on elliptic curve cryptography**

Eve and Mallory, the leaders of the Engineering Committee for Digital Hijinks (ECDH), are up to no good. Luckily, while proctoring one of Eve's midterms, a clever CO 487 TA managed to sneakily attach a (very small and undetectable) device to Eve's phone that measures power consumption. By observing Eve's network traffic, the TAs were able to deduce that on Friday, November 1, 2024, at 07:13:41 UTC time, she performed an elliptic curve Diffie–Hellman key exchange with Mallory.

While the power consumption measurement tool is impressive, it can only do so much. Modular addition, modular subtraction, and modular reduction don't require enough power to produce any meaningful output on the device; however, modular multiplication and the computation of modular inverses do. On a graph of power consumption versus time, multiplication and modular inverse operations look roughly as follows:

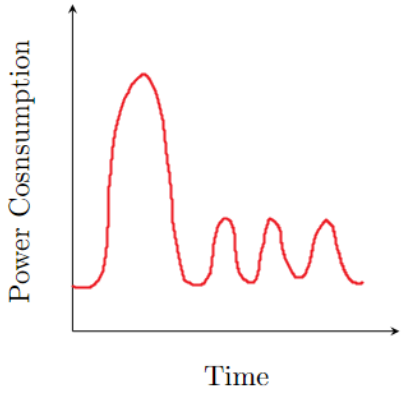


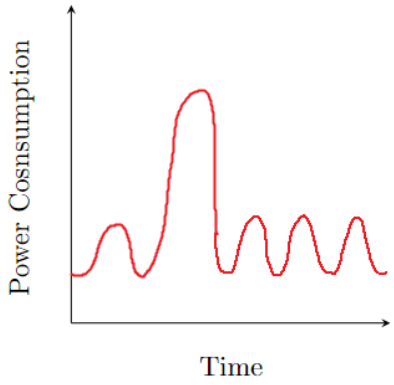
Modular multiplication power usage



Modular inversion power usage

(a) [2 marks] Let P and Q be distinct points on an elliptic curve, and consider the following two power consumption graphs:





Of the graphs above, one is the result of computing $P + Q$ and the other is the result of computing $P + P$. Which is which? Justify your reasoning.

Solution. The formula for elliptic curve point addition is

$$m = \frac{y_Q - y_P}{x_Q - x_P} \quad ; \quad x_{P+Q} = m^2 - x_P - x_Q \quad ; \quad y_{P+Q} = -(m(x_{P+Q} - x_P) + y_P)$$

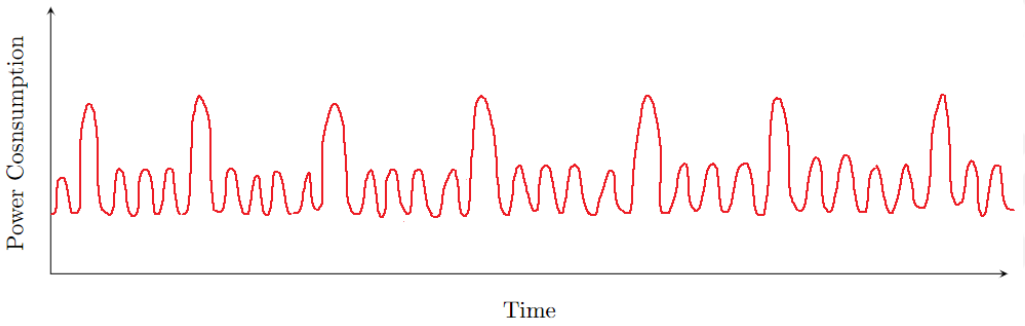
In the point addition formula, we perform one modular inverse operation followed by a multiplication operation to compute m, then a multiplication (squaring) operation to compute the x-coordinate, then a multiplication operation to compute the y-coordinate. So we have 1 inverse followed by 3 multiplications; this matches the first graph.

The formula for elliptic curve point doubling is

$$m = \frac{3x_P^2 + a}{2y_P} \quad ; \quad x_{P+P} = m^2 - x_P - x_P \quad ; \quad y_{P+P} = -(m(x_{P+P} - x_P) + y_P)$$

In the point doubling formula, we perform a multiplication operation (squaring) followed by a modular inverse operation and another multiplication to compute m, then a multiplication operation to compute the x-coordinate, then a multiplication operation to compute the y-coordinate. So we have 1 multiplication, 1 inverse, and then 3 multiplications; this matches the second graph.

(b) [3 marks] Here is the beginning of the power consumption graph that the TAs obtained from Eve's phone on Friday, November 1, 2024, at 07:13:41 UTC time:



All operations captured in the graph were performed before sending the first message of the elliptic curve Diffie–Hellman key exchange to Mallory. Based on the operations performed, determine the first (i.e. most significant) 4 bits of Eve's secret key. Justify your reasoning.

Solution. The power consumption graph shows that Eve performed the following operations, in this order:

DOUBLE - ADD - DOUBLE - DOUBLE - DOUBLE - ADD - DOUBLE.

The first step of elliptic curve Diffie–Hellman key exchange is for Eve to compute kP, where k is her secret key and P is the agreed-upon generator for the elliptic curve, which is done using the double-and-add algorithm. There are two versions of this from class. In the first version, all the DOUBLEs are performed, and then all the ADDs are performed. In the second version, for each bit in the binary representation of k (starting with the MSB), a DOUBLE is performed, and then an ADD is performed immediately after iff that bit is a 1. Since the ADDs are interspersed with the DOUBLEs in the power consumption graph, the second version must have been used, and the position of the ADDs tells us that the first four bits of Eve's secret key are 1001.

(c) [2 marks] Let E be the elliptic curve $y^2 = x^3 + 3x + 2$ over $\mathbb{Z}_{11}$. Let $P = (2, 4)$ be a point on E. Compute $9P$. Show your work. You may use computational tools as needed.

Solution. Here is one way to do it using the double-and-add algorithm:

Compute $2P$:

$$m = (3(2^2) + 3)(2(4))^{-1} = 6 \mod 11$$
$$x_{2P} = 6^2 - 2 - 6 = 10 \mod 11$$
$$y_{2P} = -(6(10 - 2) + 4) = 3 \mod 11$$

Compute $4P$:

$$m = (3(10^2) + 3)(2(3))^{-1} = 1 \mod 11$$
$$x_{4P} = 1^2 - 10 - 10 = 3 \mod 11$$
$$y_{4P} = -(1(3 - 10) + 3) = 4 \mod 11$$

Compute $8P$:

$$m = (3(3^3) + 3)(2(4)^{-1} = 1 \mod 11$$
$$x_{8P} = 1^2 - 3 - 3 = 6 \mod 11$$
$$y_{8P} = -(1(6 - 3) + 4) = 4 \mod 11$$

Compute $8P + P$:

$$m = (4 - 4)(6 - 2) = 0 \mod 11$$
$$x_{9P} = 0^2 - 2 - 6 = 3 \mod 11$$
$$y_{9P} - (0(3 - 2) + 4 = 7 \mod 11$$

So, $9P = (3, 7)$.

(d) [2 marks] Suggest a modification to the algorithm used to compute scalar multiplication of elliptic curve points that would prevent the side channel attack that you carried out in part (b).

Hint: In some cases, your algorithm might perform more operations than the one you learned in class.

Solution. If point addition does not need to be performed, we could perform a "dummy addition," where we perform a predetermined computation that mimics point addition for no reason, and do nothing with the result. This way, the power consumption graph with show an ADD after every DOUBLE, regardless of the bits of the secret key.

Alternatively, we could try to come up with point addition and point doubling formulas that have the same number and order of operations.

Note that dummy operations can address some types of side-channel attacks, but can still be vulnerable to other types of attacks, such as fault attacks. Imagine a scenario where an adversary can inject a fault into the computation that causes an operation to be skipped. Now consider what that would mean if they inserted a fault to skip an operation that might be a dummy operation (if key bit $k_i = 0$) or might not be a dummy operation (if key bit $k_1 = 1$). If the adversary causes the skip of a dummy operation, then the overall computation is still correct. Whereas if the adversary causes the skip of a non-dummy operation, then the overall computation lead to a wrong result. If the adversary is subsequently able to determine whether the result if right or wrong (for example, by checking if an ECDSA digital signature verifies), then they would still have learned the relevant key bit. So dummy operations are not always a complete solution – protecting against side channels attacks and fault attacks and other physical attacks is a tricky task.

5. [6 marks] **Learning with errors**

Here is simplified version of the LWE encryption scheme.

- The parameters n and q are positive integers, q a prime.
- The secret key s is random vector with n components in $\{-2,-1,0,1,2\}$.
- The public key is a pair (A,b), where A is a uniformly random square matrix of $\mathbb{Z}_q^{n\times n}$ and $b = As + e$, and e is a random secret vector with n components in $\{-2,-1,0,1,2\}$.

Note that the matrix A has a very high probability of being invertible when sampled uniformly at random; for the rest of this question, you can assume that A is invertible (modulo q). You can also assume that $A + I$ is invertible (modulo q) with high probability.

The encryption algorithm is the following. Given a plaintext $m \in \{0,1\}$:

i. Sample a random vector $s' \in_R \{-2,-1,0,1,2\}^n$.
ii. Sample a random vector $e' \in_R \{-2,-1,0,1,2\}^n$.
iii. Sample a random value $e'' \in_R \{-2,-1,0,1,2\}$.
iv. Compute $c_1 = s'A + e' \bmod q$.
v. Compute $c_2 = s'b + e'' + \lfloor \frac{q}{2} \rfloor m \bmod q$.
vi. Return the ciphertext (c_1, c_2).

(a) In this part, let $n = 4$ and $q = 3329$. The following public key was generated:

$$A = \begin{pmatrix} 1855 & 3164 & 2694 & 2409 \\ 1038 & 2187 & 922 & 2111 \\ 3116 & 772 & 2743 & 217 \\ 1613 & 842 & 3260 & 256 \end{pmatrix} \quad , \quad b = (674, 2069, 399, 2489)$$

and the corresponding secret key is

$$s = (-1, 1, 1, 0).$$

The message $m = 1$ was encrypted to

$$c_1 = (220, 1211, 2479, 1812) \quad , \quad c_2 = 1799.$$

Write out the steps showing the decryption of (c_1, c_2) using s to verify that it successfully recovers $m = 1$. You may use computational tools as needed.

As the name "learning with errors" suggests, this scheme comes with two error terms e, e'. It is natural to ask if we really need these terms. The goal of rest of this question is to prove that we indeed need them.

(b) Prove that if, instead of being random, e was always zero, the scheme would not be secure.
Solution. Without e, the public key becomes As. Since A is public, we could compute the secret key $s = A^{-1}As$.

(c) Prove that if, instead of being random, e' was always zero, the scheme would not be secure.
Solution. In this case, we can recover $s' = c_1 A^{-1}$. We could then compute $s'b$ which is very close to $\lfloor \frac{q}{2} \rfloor m$, which reveals m.

(d) What if s was sampled as required, but we fixed e to be equal to s? Would it be secure?
Solution. No. Observe that $As + s = (A + I)s$. We could therefore recover $s = (A + I)^{-1}b$.

(e) What if s' was sampled as required, but we fixed e' to be equal to s'? Would it be secure?
Solution. No. Observe that $s'A + s' = s'(A + I)$. Since $A + I$ is invertible, we could therefore recover $s' = c_1(A + I)^{-1}$, and then use s' to obtain m using the same method as in part (c).

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 14, 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.

Changelog

- Wed. Nov. 6: Added some useful number theory facts at the top of the assignment.
- Mon. Nov. 11: Added some guidance on Q3(c).