

UNIVERSITY
OF OSLO

TEK4500 – Introduction to Cryptography

Problem Set 11

Fall 2026

Problem 1

Read Chapter 4 and 2 (I suggest in that order) in [Men26]. Also read Chapter 20 in [Ros].

Problem 2

Let $q = 31$ and $\beta = 2$. Find all solutions to the following LWE instance.

$$\mathbf{A} = \begin{pmatrix} 11 & 3 & 27 \\ 12 & 21 & 7 \\ 6 & 23 & 30 \\ 5 & 6 & 2 \\ 21 & 0 & 14 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 25 \\ 25 \\ 12 \\ 29 \\ 17 \end{pmatrix}.$$

Note: In this problem the secret vectors $\mathbf{s}$ are chosen from the entire space $\mathbf{Z}_q^3$, rather than from the space of short vectors $[\pm\beta]^3$ (the errors $\mathbf{e}$ are of course chosen from $[\pm\beta]^5$ as usual).

Problem 3 [Problem 4.6 in [Men26]]

Consider the LWE-KEM presented in class with domain parameters $q = 997$, $n = 4$, $\beta = 5$ and keys

$$\mathbf{A} = \begin{pmatrix} 956 & 579 & 385 & 946 \\ 325 & 648 & 268 & 820 \\ 329 & 837 & 955 & 923 \\ 224 & 287 & 608 & 837 \end{pmatrix}, \quad \mathbf{s} = \begin{pmatrix} -2 \\ 5 \\ 5 \\ -2 \end{pmatrix}, \quad \mathbf{e}_1 = \begin{pmatrix} 4 \\ -2 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 23 \\ 294 \\ 475 \\ 359 \end{pmatrix}.$$

a) Verify that the encryption of plaintext $k = 1$ using random quantities

$$\mathbf{r} = \begin{pmatrix} 2 \\ -1 \\ 4 \\ -4 \end{pmatrix}, \quad \mathbf{e}_2 = \begin{pmatrix} 5 \\ -4 \\ 4 \\ 0 \end{pmatrix}, \quad e_3 = -4.$$

yields the the ciphertext $C = (\mathbf{u}, v)$ where

$$\mathbf{u} = \begin{pmatrix} 18 \\ 712 \\ 897 \\ 419 \end{pmatrix}, \quad v = 711.$$

b) Decrypt C.

Problem 4 [Problem 2.1 in [Men26]]
Consider the lattice $\mathcal{L} \subseteq \mathbf{Z}_q^3$ with basis matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 0 & -5 \\ 4 & 3 & 1 \\ -2 & -3 & 2 \end{pmatrix}.$$

a) show that $(7, -3, 2)^T$ is in $\mathcal{L}$.

b) show that $(-4, 4, 0)^T$ is not in $\mathcal{L}$.

Problem 5
In class we stated that the Search- and Decisional-LWE problems are equivalent, and proved how you could solve the Search-LWE problem given an algorithm that solves the Decisional-LWE problem. Show the other direction. That is, suppose you have an algorithm that solves the Search-LWE problem, show how you can use this to solve the Decisional-LWE problem.

Hint: Lemma 2 in [Lyu24].

Problem 6
In this problem we'll show that the LWE problem with short secrets is equivalent to the LWE problem where the secret can be of arbitrary length. Specifically, the *short-secret* LWE problem is the following: given $\mathbf{A} \in \mathbf{Z}_q^{m \times n}$ and $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$, where $\mathbf{s} \in [\pm\beta]^n$ and $\mathbf{e} \in [\pm\beta]^m$, find $\mathbf{s}$. The (*normal*) LWE problem is the same as short-secret LWE, except that now $\mathbf{s}$ can be an arbitrary element in $\mathbf{Z}_q^n$.

Note: For both subproblems below you can assume that the LWE parameters are chosen so that for each instance $(\mathbf{A}, \mathbf{b})$ there is a unique solution $\mathbf{s}$.

a) Suppose you have an oracle that solves the normal LWE problem. Show how you can use this to solve the short-secret LWE problem.

Hint: Pick a random $\mathbf{r} \twoheadleftarrow \mathbf{Z}_q^n$ and set $\mathbf{b}' \leftarrow \mathbf{A}\mathbf{r} + \mathbf{b}$. What can you say about the distribution of $\mathbf{b}'$?

b) Suppose you have an oracle that solves the short-secret LWE problem with parameters m, n, q, β. Show how you can use this to solve the normal LWE problem with parameters $(m-n), n, q, \beta$.

Hint: Let $(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$ be a normal $\mathsf{LWE}(m, n, q, \beta)$ instance. Suppose $\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$ and $\mathbf{e} = \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix}$, where

- $\mathbf{A}_1 \in \mathbf{Z}_q^{n \times n}$, $\mathbf{b}_1 \in \mathbf{Z}_q^n$, $\mathbf{e}_1 \in \mathbf{Z}_q^n$
- $\mathbf{A}_2 \in \mathbf{Z}_q^{(m-n) \times m}$, $\mathbf{b}_2 \in \mathbf{Z}_q^{(m-n)}$, $\mathbf{e}_2 \in \mathbf{Z}_q^{(m-n)}$.

Let $\mathbf{A}' = -\mathbf{A}_2\mathbf{A}_1^{-1} \in \mathbf{Z}_q^{(m-n) \times n}$ and $\mathbf{b}' = \mathbf{A}'\mathbf{b}_1 + \mathbf{b}_2 \in \mathbf{Z}_q^{m-n}$.

What can you say about the instance $(\mathbf{A}', \mathbf{b}')$? (Expand $\mathbf{b}'$)

Problem 7 [The NTT algorithm]

In this problem we'll look at the *Number Theoretic Transform* (NTT), which is one of the key optimization tricks underlying the ML-KEM scheme. The NTT can be seen as a finite analogue of the Fast Fourier Transform (FFT), and allows to speed up the multiplication of two polynomials from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.

Note: For this problem it is *highly* recommended that you watch the following YouTube video on the FFT algorithm. It can also be useful to read Chapter 13 in [Men26].

In the following, let $\mathbf{Z}_q[X]$ denote the set of all polynomials in the indeterminate X, with coefficients in $\mathbf{Z}_q$. For example, if $q = 17$, then $a(X) = 9 + 2X^2 + 15X^3 + X^5$ and $b(X) = 1 + 16X^8 + X^9$ are two examples of polynomials in $\mathbf{Z}_{17}[X]$.

a) Let $q = 257$ (this is a prime number). Multiply the two polynomials

$$a(X) = 129 + 6X + 101X^2 + 2X^3$$
$$b(x) = 122 + 225X + 206X^2 + 199X^3$$

in $\mathbf{Z}_q[X]$ using the normal schoolbook multiplication method. How many multiplications of elements in $\mathbf{Z}_q$ did this require?

The first key observation of the NTT algorithm is that there are two different ways of representing a polynomial: the *coefficient representation*, as used above; or a *point-value representation*. A point-value representation of a polynomial of degree n is a set of $\geq n+1$ pairs consisting of the values of the polynomial evaluated at $\geq n+1$ different input points, i.e.,

$$\{(x_0, y_0), (x_1, y_2), \ldots, (x_n, y_n)\},$$

where $y_0 = a(x_0)$, $y_1 = a(x_1)$, and so on. For example, a point-value representation of the polynomial $a(X) = 1 + 2X + X^3 \in \mathbf{Z}_7[X]$ is

$$\{(1, 4), (2, 6), (3, 6), (4, 3)\}.$$

Note that a polynomial has many different point-value representations, depending upon which input points you used to evaluate with. The amazing fact about point-value representations is that they *uniquely* determine a polynomial.

Theorem 1. *Let $\{(x_0, y_0), (x_1, y_2), \ldots, (x_n, y_n)\}$ be a point-value representation where each element is in $\mathbf{Z}_q$ for a prime q. Then there is a* unique *polynomial $a(X) = a_0 + a_1X + \cdots + a'_nX^{n'} \in \mathbf{Z}_q[X]$ of degree $n' \leq n$ such that $a(x_0) = y_0, \ldots, a(x_n) = y_n$.*

Proof. See Wikipedia. □

So how does point-value representations help speed up polynomial multiplication? As you demonstrated in a), the product of $a(X)$ and $b(X)$ is unfortunately *not* as simple as just multiplying their coefficients pairwise. However, for point-value representations this *is* the case! That is, suppose $\{(x_0, y_0), \ldots, (x_n, y_n)\}$ is a point-value representation of a degree n polynomial $a(X)$, and $\{(x_0, y'_0), \ldots, (x_n, y'_n)\}$ is a point-value representation of a degree n polynomial $b(X)$, then the set consisting of the pairwise products of their values

$$\{(x_0, y_0 \cdot y_0), \ldots, (x_n, y_n \cdot y'_n)\}$$

is *almost* a point-value representation of $c(X) = a(X) \cdot b(X)$[1]. We say almost, because the polynomial $c(X)$ has degree $2n$, which means that $n+1$ point-value pairs is not enough to uniquely determine it. However, this is simple to fix: simply extend the point-value representations of $a(X)$ and $b(X)$ to consist of $2n+1$ pairs instead.[2] So let $\{(x_0, y_0), \ldots, (x_{2n}, y_{2n})\}$ be a point-value representation of $a(X)$ consisting of $2n+1$ pairs, and $\{(x_0, y'_0), \ldots, (x_{2n}, y'_{2n})\}$ be a point-value representation of $b(X)$ consisting of $2n+1$ pairs, then the following *is* a point-value representation of $c(X)$:

$$\{(x_0, y_0 \cdot y'_0), \ldots, (x_{2n}, y_{2n} \cdot y'_{2n})\}.$$

[1] Specifically, if $a(x_i) = y_i$ and $b(x_i) = y'_i$, then $c(x_i) = a(x_i) \cdot b(x_i) = y_i \cdot y'_i$.

[2] Adding additional points to a point-value representation beyond the required $n+1$ pairs does not change which polynomial it represents.

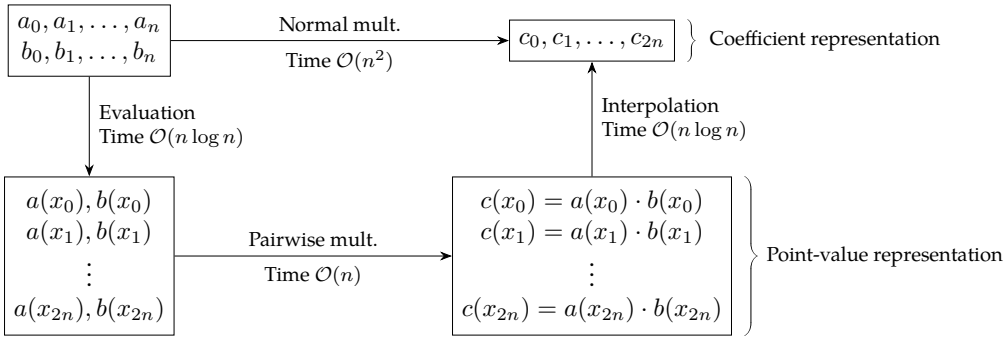


Figure 1: Multiplying polynomials using the NTT algorithm.

Notice that producing a point-value representation of $c(X)$ from point-value representations of $a(X)$ and $b(X)$ only requires $\mathcal{O}(n)$ multiplications! This suggests the following strategy for multiplying $a(X)$ and $b(X)$. First transform $a(X)$ and $b(X)$ into point-value representations using $2n + 1$ points, then multiply the pairs pairwise to produce a point-value representation of $c(X)$. Finally, transform $c(X)$ back into coefficient representation to get the final answer. See Fig. 1 for an illustration of this approach.

b) Evaluate the polynomials $a(X)$ and $b(X)$ from a) at the points $1, 2, 3, 4, 5, 6, 7$, and compute a point-value representation of $c(X) = a(X) \cdot b(X)$. How many multiplications in $\mathbf{Z}_q$ did this require?

c) Translate the point-value representation of $c(X)$ you computed in b) into its coefficient representation. Compare your result with what you computed in a).

Hint: Note that the point-value representation of $c(X)$ corresponds to a linear equation system $\mathbf{y}_c = M \cdot \mathbf{c}$, where the vector $\mathbf{y}_c$ represents all the values of the point-value representation of $c(X)$, matrix M represents all powers of the evaluation points, and the vector $\mathbf{c}$ are the unknown coefficients $c_0, c_1, \ldots, c_{2n}$ of $c(X)$. Specifically, we have

$$\begin{pmatrix} 104 \\ 114 \\ 48 \\ \vdots \\ 145 \end{pmatrix} = \begin{pmatrix} 1 & 1^1 & 1^2 & 1^3 & 1^4 & 1^5 & 1^6 \\ 1 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 \\ 1 & 3^1 & 3^2 & 3^3 & 3^4 & 3^5 & 3^6 \\ & & & \vdots & & & \\ 1 & 7^1 & 7^2 & 7^3 & 7^4 & 7^5 & 7^6 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_6 \end{pmatrix} \pmod{q}$$

From this, you can solve for $\mathbf{c}$ by computing $M^{-1}\mathbf{y}_c = \mathbf{c}$.

The approach in c) unfortunately does not improve upon the normal polynomial multiplication method used in a). The problem is that both the translation from coefficient to point-value representation (evaluation), and the translation back to coefficient representation (interpolation) still takes time $\mathcal{O}(n^2)$ with the evaluation points we used.

Here comes the next big observation of the NTT algorithm: there are some evaluation points that are better than others. Specifically, suppose $a(X)$ is a polynomial having only even powers, and r is a point such that $r^2 = -1 \pmod{q}$, For example, suppose $a(X) = 1 + 3X^2 + X^4 \in \mathbf{Z}_{17}[X]$ and $r = 4$ (note that $r^2 = -1 \pmod{17}$). If we evaluate $a(X)$ at $\pm r$ we get

$$a(r) = 1 + 3 \cdot 4^2 + (4^2)^2 = 1 + 3 \cdot (-1) + (-1)^2 = 16 \pmod{17}$$

and

$$a(-r) = 1 + 3 \cdot (-4)^2 + ((-4)^2)^2 = 1 + 3 \cdot (-1) + (-1)^2 = 16 \pmod{17}.$$

In other words, once we know $a(r)$ we immediately know $a(-r)$ too! And this is true for all polynomials that only have even powers.

What about polynomials with only odd powers? In this case we can factor out one X and be left with a product of X and a polynomial of only even powers. For example, suppose $a(X) = 7X + 2X^3 + 9X^5$, then $a(X) = X \cdot (7 + 2X^2 + 9X^4)$. Consequently, when we evaluate at $\pm r$ we get

$$a(r) = 4 \cdot (7 - 2 + 9) = 5 \pmod{17}$$

and

$$a(-r) = -4 \cdot (7 - 2 + 9) = -5 \pmod{17}.$$

For odd-powered polynomials we always have $a(r) = -a(-r)$.

Finally, for arbitrary polynomials we can combine the above observations as follows. Suppose $a(X) = 1+7X+3X^2+2X^3+X^4+9X^5$. From the even-indexed and odd-indexed coefficients of $a(X)$ define the following two new polynomials

$$a_{even}(Y) = 1 + 3Y + Y^2$$
$$a_{odd}(Y) = 7 + 2Y + 9Y^2.$$

That is, $a_{even}(Y)$ consists of all the even-indexed coefficients of $a(X)$, and $a_{odd}(Y)$ consists of all the odd-indexed coefficients of $a(X)$. Moreover, notice that

$$a(X) = a_{even}(X^2) + X \cdot a_{odd}(X^2). \tag{1}$$

Suppose we knew the values of $a_{even}(r^2)$ and $a_{odd}(r^2)$, then by the above we would also immediately know the values of $a(r)$ and $a(-r)$:

$$a(r) = a_{even}(r^2) + r \cdot a_{odd}(r^2). \tag{2}$$

and

$$a(-r) = a_{even}((-r)^2) - r \cdot a_{odd}((-r)^2) = a_{even}(r^2) - r \cdot a_{odd}(r^2). \tag{3}$$

In other words, in order to evaluate $a(X)$ at $2n$ points, it is sufficient to evaluate it at n points that have the property $r^2 = -1 \pmod{q}$. If q is a prime with $q = 1 \pmod{2n}$, then n such r values are guaranteed to exist (which is the case for $q = 17$ and $n = 8$ in our example).

So how do we evaluate $a_{even}(r^2)$ and $a_{odd}(r^2)$? Notice that this is just the same problem as before, but now for polynomials of degree $n/2$. This means that we can recursively solve the problem in (1).

d) Implement the recursive NTT transform and use it to translate the polynomials in a) into point-value representations. Also implement the inverse NTT transform that translate back into coefficient representation.

To help you get started, some initial Sage code is provided below. You should fill in the functions denoted `ntt` and `ntt_inverse`.

The degree n of $a(X)$ and $b(X)$ is less than $n = 4$, which means that $N = 8 = 2 \cdot n$ is sufficient to represent their product in point-value representation. Thus, the coefficient representations of $a(X)$ and $b(X)$ should be padded with zeros to consist of $N = 8$ coefficients before applying the NTT transform. Moreover, $q = 257 = 1 \pmod{2N}$ which ensures that there exists N values r such that $r^2 = -1 \pmod{q}$.

Hint: The r values can all be generated from a single one by raising it to all powers between 1 and $n/2$. Moreover, $r^{j+n/2} = -r^j \pmod{q}$.

Hint: Take inspiration from the code shown in the linked YouTube presentation. This video is presenting the Fast Fourier Transform (FFT) over the real numbers, rather than over the finite field $\mathbf{Z}_q$ as here. However, only minor modifications are necessary in order to move to $\mathbf{Z}_q$.

```
N = 8
q = 257
Zq = Integers(q)
g = Zq(3)
R.<X> = PolynomialRing(Zq, 'X')   # define Zq[X]

a = 129 + 6*X + 101*X^2 + 2*X^3
b = 122 + 225*X + 206*X^2 + 199*X^3

if g.is_primitive_root():
    print("g =", g, "is a generator for Z_q^*")

# neccessary reqs. for 2N-root of unity to exist
if is_prime(q) and q % (2*N) == 1:
    print("q is a prime and q (mod 2N) = 1", )

# omega is an N-root of unity mod q, i.e., omega^N = 1 (mod q)
omega = g^((q-1) / N)


# convert polynomial to list of coefficients;
# pad with zeros to make it length n
def to_list(a, n):
  return list(a) + [Zq(0)] * (n - (a.degree() + 1))

# evaluate a polynomial at n points
#    input: a list of n coefficients
#    return: a list of n evaluation points
def ntt(a):
  n = len(a)
  if n == 1:
      return a
  r = omega^(N/n)                    # r is an n-th root of -1 (mod q)
  y = [Zq(0)]*n
  a_even, a_odd = a[::2], a[1::2]
  #
  # Your code goes here
  #
  return y

# transform point-value repr. back to coefficient repr.
#    input: a list of n evaluation points coefficients
#    return: a list of n coefficients
def ntt_inverse(a):
  n = len(a)
  if n == 1:
      return a
  r = omega^(-N/n)                   # r is an n-th root of -1 (mod q)
  y = [Zq(0)]*n
```

```
    a_even, a_odd = a[::2], a[1::2]
    #
    # Your code goes here
    #
    if n == N:
        return = [y[j] * Zq(n)^-1 for j in range(len(y))]

    return y
```

e) Use your code from d) to implement a proper NTT-based polynomial multiplication. Run your multiplication on $a(X)$ and $b(X)$ from a). Compare your results. How many multiplications did the NTT-based multiplication require?

f) How large does N have to be before the NTT-based polynomial multiplication requires fewer multiplications (in $\mathbf{Z}_{257}$) than the normal multiplication from a)? Try increasing N to $16, 32, 64, \ldots$ and multiply two polynomials of degree $N/2 - 1$.

Hint: To create a random polynomial in $\mathbf{Z}_q[X]$ of degree $N/2 - 1$ you can use the following command in Sage: `R.random_element(N // 2 - 1)`

References

[Lyu24] Vadim Lyubashevsky. Basic lattice cryptography: The concepts behind Kyber (ML-KEM) and Dilithium (ML-DSA). *IACR Cryptol. ePrint Arch.*, 2024. https://eprint.iacr.org/2024/1287.

[Men26] Alfred Menezes. *A Gentle Introduction to Lattice-Based Cryptography*. 2026. https://cryptography101.ca/lattice-based-cryptography.

[Ros] Mike Rosulek. *The Joy of Cryptography*. https://joyofcryptography.com.