Learning with Errors based Cryptosystems

Nov 2023

Dhanyamraju Harsh Rao
George Rahul

TABLE OF CONTENTS

LEARNING WITHOUT ERRORS PROBLEM

$$\begin{bmatrix} 1 & 2 & 3 \\ 7 & 4 & 8 \\ 12 & 6 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix} = \begin{bmatrix} 36 \\ 95 \\ 36 \end{bmatrix}$$

$$A_{3x3} \times S_{3x1} = T_{3x1}$$

Given A & T, Find S

Gaussian Elimination !

LEARNING WITH ERRORS PROBLEM

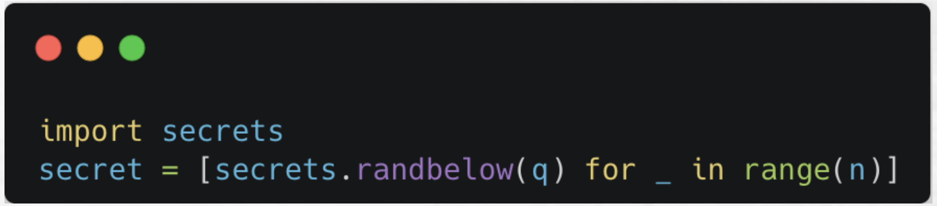
$$\left(\begin{bmatrix} 1 & 2 & 3 \\ 7 & 4 & 8 \\ 12 & 6 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix} + \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}\right) \bmod 13 \equiv \begin{bmatrix} 11 \\ 4 \\ 9 \end{bmatrix}$$

$$\left(A_{3x3} \times S_{3x1} + E_{3x1}\right) \bmod q \equiv T_{3x1}$$

Given q, A & T, Find S

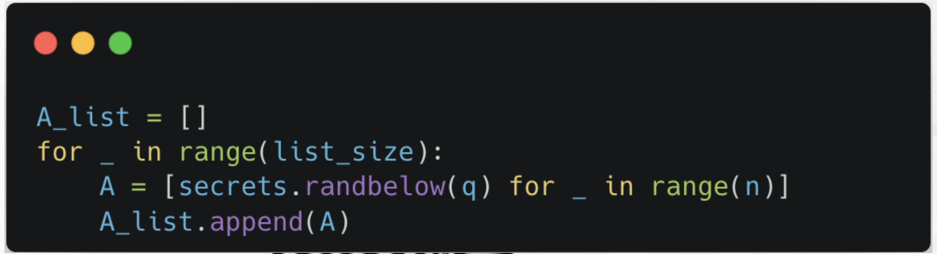
LWE PUBLIC KEY CRYPTO: KEY GENERATION

1) Decide the values of q, n, max_error, and list_size.
2) Randomly Generate Secret S_{nx1}



```python
import secrets
secret = [secrets.randbelow(q) for _ in range(n)]
```

3) Randomly Generate $A_{list\_size\ x\ n}$

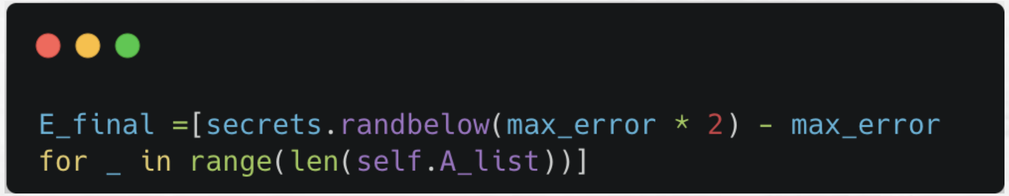


```python
A_list = []
for _ in range(list_size):
    A = [secrets.randbelow(q) for _ in range(n)]
    A_list.append(A)
```

LWE PUBLIC KEY CRYPTO: KEY GENERATION

4) Randomly Generate Errors $E_{\text{list\_size x 1}}$

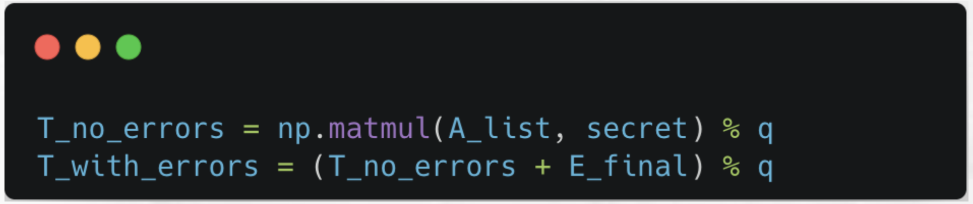
Error Range = Integers in [-max_error, max_error]



```
E_final =[secrets.randbelow(max_error * 2) - max_error
for _ in range(len(self.A_list))]
```

Error Distribution can be a Uniform or Gaussian Distribution

5) Calculate $T_{\text{list\_size x n}}$



```
T_no_errors = np.matmul(A_list, secret) % q
T_with_errors = (T_no_errors + E_final) % q
```

LWE PUBLIC KEY CRYPTO: KEY GENERATION

$$\begin{bmatrix} 1 & 2 & 3 \\ 7 & 4 & 8 \\ 12 & 6 & 0 \end{bmatrix} \quad \begin{bmatrix} 11 \\ 4 \\ 9 \end{bmatrix} \quad 13 \quad 1 \quad \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$$

$A_{\text{list\_size x n}}$, $T_{\text{list\_size x 1}}$, q (prime) , max_error S_{nx1}

Public Key **Private Key**

LWE PUBLIC KEY CRYPTO: ENCRYPTION

$$\begin{bmatrix} 1 & 2 & 3 \\ 12 & 6 & 0 \\ 7 & 4 & 8 \end{bmatrix} \quad \begin{bmatrix} 11 \\ 9 \\ 4 \end{bmatrix}$$

$A_{\text{list\_size x n}}$ $T_{\text{list\_size x 1}}$

Public Key

LWE PUBLIC KEY CRYPTO: ENCRYPTION

$$\begin{bmatrix}1 & 2 & 3\\ 12 & 6 & 0\\ 7 & 4 & 8\end{bmatrix} \quad \begin{bmatrix}11\\ 9\\ 4\end{bmatrix} \rightarrow + \rightarrow \begin{array}{l} A_{new} \\ \begin{bmatrix}8 & 6 & 11\end{bmatrix} \\ T_{new} \begin{bmatrix}2\end{bmatrix} \end{array}$$

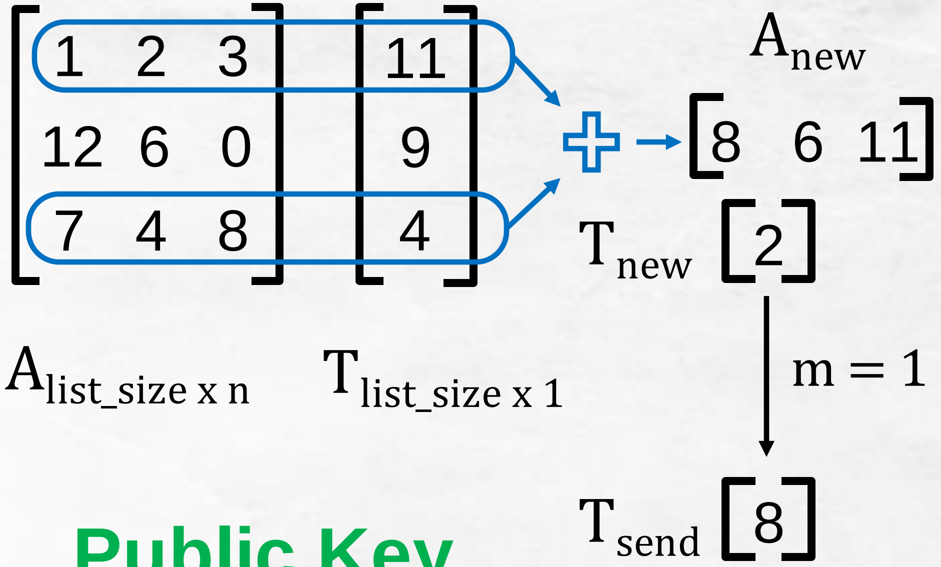
$A_{\text{list\_size x n}}$ $T_{\text{list\_size x 1}}$

1) Select some rows from A & T (with repetition) and add them (in mod q).

$$no_{selected} \times \text{max\_}error \le \left\lfloor \frac{q}{4} \right\rfloor$$

Public Key

LWE PUBLIC KEY CRYPTO: ENCRYPTION



Public Key

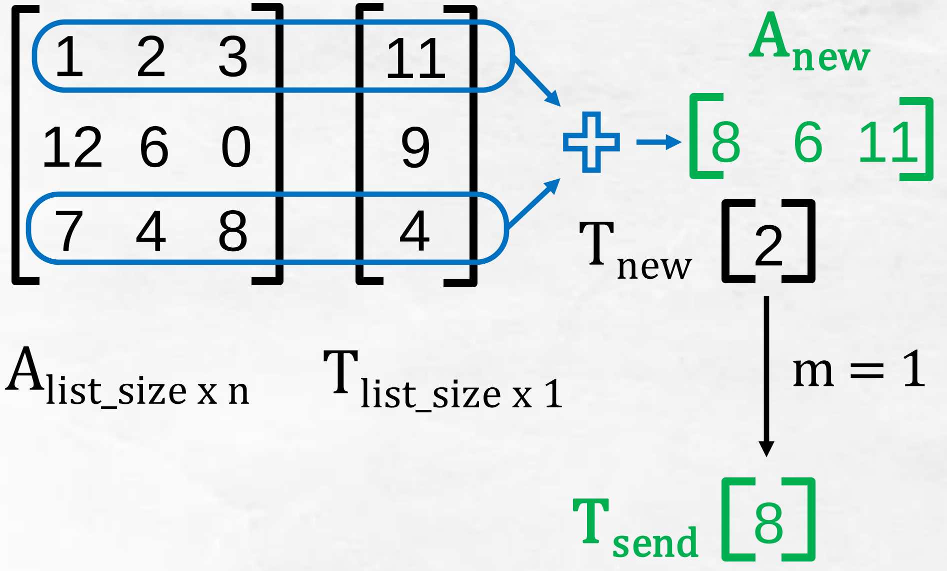
1) Select some rows from A & T (with repetition) and add them (in mod q).

$$no_{selected} \times \text{max\_}error \le \left\lfloor \frac{q}{4} \right\rfloor$$

2) Add q/2 times the message bit m to T_{new} .

$$\boxed{T_{send} = [T_{new} + (m \ * \ \left\lfloor \frac{q}{2} \right\rfloor)] \ mod \ q}$$

LWE PUBLIC KEY CRYPTO: ENCRYPTION



1) Select some rows from A & T (with repetition) and add them (in mod q).

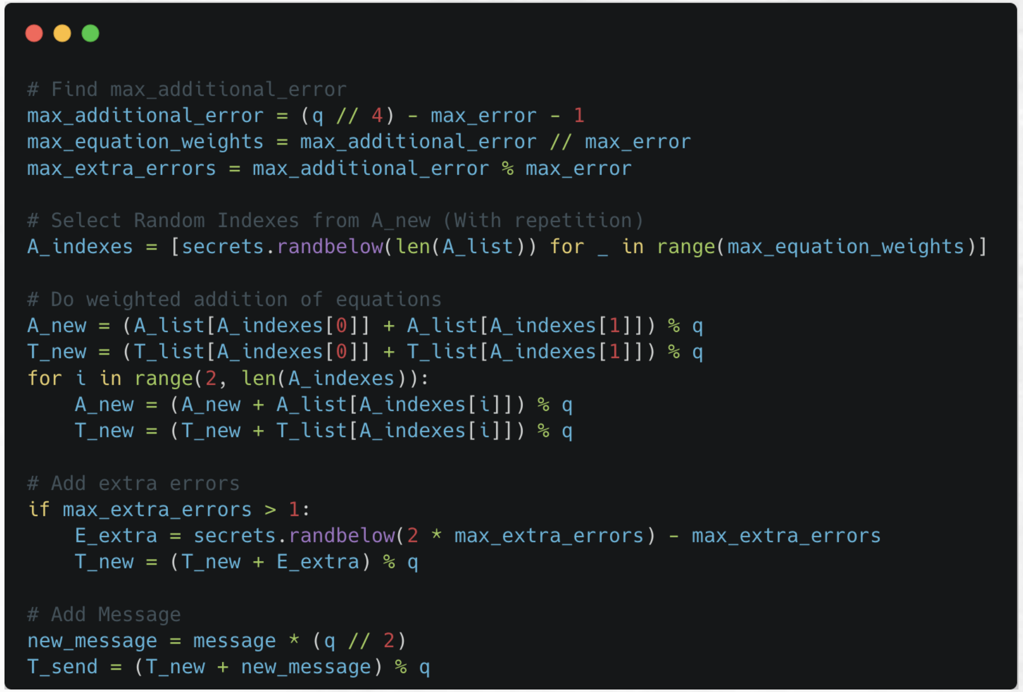
$$no_{selected} \times \max\_error \le \left\lfloor \frac{q}{4} \right\rfloor$$

2) Add q/2 times the message bit m to T_{new} .

$$\boxed{T_{send} = [T_{new} + (m * \left\lfloor \frac{q}{2} \right\rfloor)] \ mod\ q}$$

3) Send A_{new} and T_{send}

LWE PUBLIC KEY CRYPTO: ENCRYPTION



```python
# Find max_additional_error
max_additional_error = (q // 4) - max_error - 1
max_equation_weights = max_additional_error // max_error
max_extra_errors = max_additional_error % max_error

# Select Random Indexes from A_new (With repetition)
A_indexes = [secrets.randbelow(len(A_list)) for _ in range(max_equation_weights)]

# Do weighted addition of equations
A_new = (A_list[A_indexes[0]] + A_list[A_indexes[1]]) % q
T_new = (T_list[A_indexes[0]] + T_list[A_indexes[1]]) % q
for i in range(2, len(A_indexes)):
    A_new = (A_new + A_list[A_indexes[i]]) % q
    T_new = (T_new + T_list[A_indexes[i]]) % q

# Add extra errors
if max_extra_errors > 1:
    E_extra = secrets.randbelow(2 * max_extra_errors) - max_extra_errors
    T_new = (T_new + E_extra) % q

# Add Message
new_message = message * (q // 2)
T_send = (T_new + new_message) % q
```

1) Select some rows from A & T (with repetition) and add them (in mod q).

$$no_{selected} \times \max\_error \leq \left\lfloor \frac{q}{4} \right\rfloor$$

2) Add q/2 times the message bit m to T_{new} .

$$\boxed{T_{send} = [T_{new} + (m * \left\lfloor \frac{q}{2} \right\rfloor)] \bmod q}$$

3) Send A_{new} and T_{send}

LWE PUBLIC KEY CRYPTO: DECRYPTION

$$\underbrace{\begin{bmatrix} 8 & 6 & 11 \end{bmatrix}}_{A_{new}} \times \underbrace{\begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}}_{S_{3x1}} \equiv \overset{T_{ideal}}{\begin{bmatrix} 1 \end{bmatrix}}$$

T_{send} $\begin{bmatrix} 8 \end{bmatrix}$

1) Calculate T_{ideal}

$$T_{ideal} = (A_{new} \times S) \bmod q$$

LWE PUBLIC KEY CRYPTO: DECRYPTION

$$\begin{tikzcd}
\underset{A_{new}}{\begin{bmatrix}8 & 6 & 11\end{bmatrix}} \times \underset{S_{3x1}}{\begin{bmatrix}1\\4\\9\end{bmatrix}} \equiv & \overset{T_{ideal}}{\begin{bmatrix}1\end{bmatrix}} \arrow[d] \\
T_{send} \begin{bmatrix}8\end{bmatrix} \arrow[r] & - \arrow[d] \\
\text{Message Bit} = & \begin{bmatrix}1\end{bmatrix}
\end{tikzcd}$$

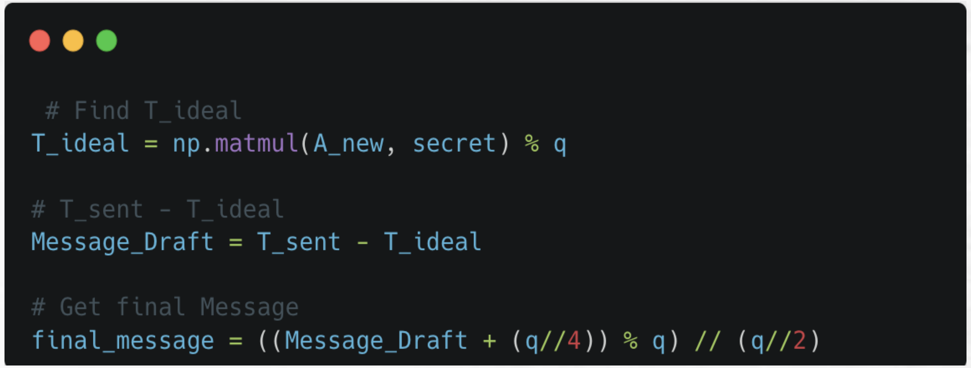
1) Calculate T_{ideal}

$$T_{ideal} = (A_{new} \times S) \bmod q$$

2) Calculate Message Bit m

$$m = \left\lfloor \frac{(T_{sent} - T_{ideal} + \left\lfloor \frac{q}{4} \right\rfloor) \bmod q}{\left\lfloor \frac{q}{2} \right\rfloor} \right\rfloor$$

LWE PUBLIC KEY CRYPTO: DECRYPTION



```
 # Find T_ideal
T_ideal = np.matmul(A_new, secret) % q

# T_sent - T_ideal
Message_Draft = T_sent - T_ideal

# Get final Message
final_message = ((Message_Draft + (q//4)) % q) // (q//2)
```

1) Calculate T_{ideal}

$$T_{ideal} = (A_{new} \times S) \, mod \, q$$

2) Calculate Message Bit m

$$m = \left\lfloor \frac{(T_{sent} - T_{ideal} + \left\lfloor \frac{q}{4} \right\rfloor) \, mod \, q}{\left\lfloor \frac{q}{2} \right\rfloor} \right\rfloor$$

RING LEARNING WITH ERRORS (RLWE)

$$\begin{array}{lll} & 6x^2 + 8x + 3 & : A_3 \\ \times & x^2 + 4x + 9 & : S_3 \\ \hline \% & x^3 + 1 & : \Phi_4 \\ \hline + & x^2 + 0x + 1 & : E_3 \\ \hline \% & x^3 + 1 & : \Phi_4 \\ \hline & 10x^2 + x + 9 & : T_3 \end{array}$$

1) All numbers must be in mod q

2) Given q, A, Φ, T. Find S !

3) All intermediate polynomials must be in the corresponding Galois Field. If they exceed it divide by Φ and take remainder.

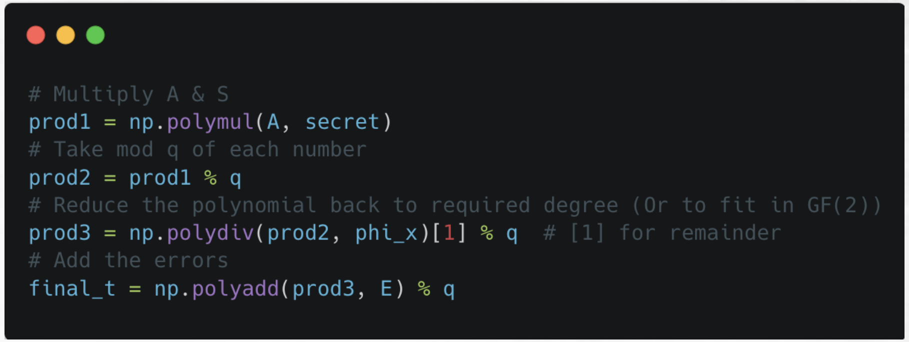
RLWE PUBLIC KEY CRYPTO: KEY GENERATION

	$6x^2 + 8x + 3$	$: A_3$
×	$x^2 + 4x + 9$	$: S_3$
%	$x^3 + 1$	$: \Phi_4$
+	$x^2 + 0x + 1$	$: E_3$
%	$x^3 + 1$	$: \Phi_4$
	$10x^2 + x + 9$	$: T_3$

1) Choose n, q, list_size
2) Choose Φ_{n+1}
3) Randomly generate S_n
4) Randomly generate $A_{\text{list\_size x n}}$
5) Randomly generate $E_{\text{list\_size x n}}$

Note: $A_{\text{list\_size x n}}$ & $E_{\text{list\_size x n}}$ are lists containing list_size polynomials of degree n

RLWE PUBLIC KEY CRYPTO: KEY GENERATION



```
# Multiply A & S
prod1 = np.polymul(A, secret)
# Take mod q of each number
prod2 = prod1 % q
# Reduce the polynomial back to required degree (Or to fit in GF(2))
prod3 = np.polydiv(prod2, phi_x)[1] % q  # [1] for remainder
# Add the errors
final_t = np.polyadd(prod3, E) % q
```

1) Choose n, q, list_size
2) Choose Φ_{n+1}
3) Randomly generate S_n
4) Randomly generate $A_{\text{list\_size x n}}$
5) Randomly generate $E_{\text{list\_size x n}}$
6) Calculate $T_{\text{list\_size x n}}$

Note: $A_{\text{list\_size x n}}$ & $E_{\text{list\_size x n}}$ are lists containing list_size polynomials of degree n

RLWE PUBLIC KEY CRYPTO: KEY GENERATION

$$\begin{bmatrix} 2x^2 + 7x + 11 \\ 6x^2 + 8x + 3 \\ 12x^2 + 6x \end{bmatrix} \begin{bmatrix} 6x^2 + x + 5 \\ 10x^2 + x + 9 \\ 5x^2 + 2x + 1 \end{bmatrix} \begin{bmatrix} x^3 + 1 \end{bmatrix} \quad 13 \quad 1 \qquad \begin{bmatrix} x^2 + 4x + 9 \end{bmatrix}$$

$$A_{\text{list\_size x n}}, \quad T_{\text{list\_size x 1}}, \quad \Phi, \quad q, \ \text{max\_error} \qquad S_{nx1}$$

Public Key **Private Key**

RLWE PUBLIC KEY CRYPTO: ENCRYPTION

$$\boxed{T_{send} = [T_{new} + (m \ * \ \lfloor \frac{q}{2} \rfloor)] \ mod \ q}$$

$$\begin{bmatrix} 2x^2 + 7x + 11 \\ 6x^2 + 8x + 3 \\ 12x^2 + 6x \end{bmatrix} \begin{bmatrix} 6x^2 + x + 5 \\ 10x^2 + x + 9 \\ 5x^2 + 2x + 1 \end{bmatrix}$$

$$A_{list\_size \times n}, \qquad T_{list\_size \times 1},$$

$$+ \rightarrow \begin{array}{l} 8x^2 + 2x + 1 : \mathbf{A_{new}} \\ 3x^2 + 2x + 1 : T_{new} \end{array}$$

$$\text{message} = [1, 1, 0] \rightarrow m = x^2 + x$$

$$\downarrow$$

$$9x^2 + 8x + 1 : \mathbf{T_{send}}$$

RLWE PUBLIC KEY CRYPTO: DECRYPTION

$$\begin{array}{lll}
 & 8x^2 + 2x + 1 & : A_{new} \\
\times & x^2 + 4x + 9 & : S \\
\hline
\% & x^3 + 1 & : \Phi \\
\hline
 & 3x^2 + x + 3 & : T_{ideal}
\end{array}$$

$$\begin{tikzcd}
3x^2 + x + 3 \arrow[dr] & & \\
 & - \arrow[dr] & \\
T_{send}: 9x^2 + 8x + 1 \arrow[ur] & & \text{message} = x^2 + x \arrow[r] & {[1, 1, 0]}
\end{tikzcd}$$

$$\boxed{m = \left\lfloor \frac{\left(T_{sent} - T_{ideal} + \left\lfloor \frac{q}{4} \right\rfloor\right) \bmod q}{\left\lfloor \frac{q}{2} \right\rfloor} \right\rfloor}$$

RLWE can send n bits a pass while LWE can only send 1 bit per pass

SECURITY PARAMS: Public Key Entropy

1) Refers to the number of ways of generating $\mathrm{A}_{\mathrm{new}}$ and $\mathrm{T}_{\mathrm{new}}$

2) If attacker can discover how to generate $\mathrm{A}_{\mathrm{new}}$ they can generate $\mathrm{T}_{\mathrm{new}}$ and retrieve the message.

$$Entropy = {}^{list\_size + eq\_selected - 1}C_{eq\_selected}$$

eq_selected →

list_size ↓

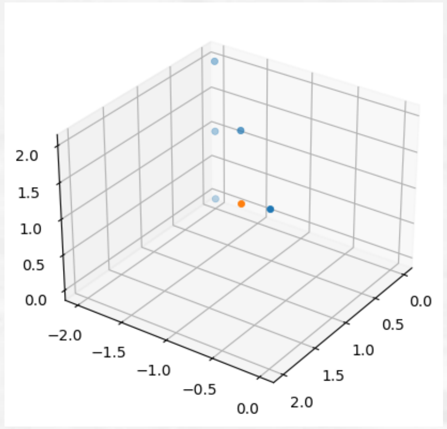
	4	8	16	32	64	128
16	11	18	28	39	52	66
32	15	25	40	59	83	109
64	19	33	54	84	123	170
128	23	41	69	111	171	250
256	27	48	84	141	226	347
512	31	56	100	171	285	457

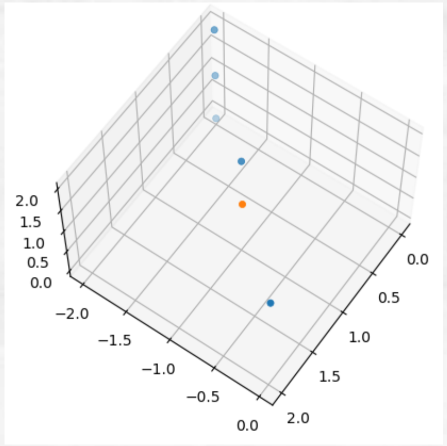
log_2 Entropy for Public Key in LWE

(Selection with repetition = ${}^{n+r-1}C_r$)

SECURITY PARAMS: Bad RNG

1) RNG must generate numbers of range [0, q).
2) A smaller range increases the vulnerability to a noise reduction attack.
3) The attack works by solving different sets of the equations and estimating S
4) Don’t need to find exact value to decrypt the message





SECURITY PARAMS: Large q

- Setting q too large opens yourself to polynomial time Lattice Reduction Attacks.
- r is a lower bound.

n	r (log2(q))	Time (estimate)
512	76	25 days
1024	147	4 years
2048	303	275 years
4096	640	19700 years

Source: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/LWEattackPreprint.pdf

SECURITY PARAMS: Finite Set of Errors

- In practical implementations, the distribution from which the errors are drawn from are usually discrete.
- They are also bounded by *max_error* to ensure that the decryption to occur without any mistakes.
- However, this allows for a simple but clever algebraic attack that can recover the secret **s** using the public keys.

Arora-Ge Algebraic Attack

We have A_list and T_list (public keys) which are made up of **m** individual samples **<a, b>**

a := vector of coefficients
b := $\mathbf{a}^T\underline{\mathbf{s}} + \mathbf{e}$ where errors, $\mathbf{e} \in \mathbf{E} \subseteq \mathbf{Z}_q$

$$f_{\mathbf{a},b}(\mathbf{s}) = \prod_{x \in E} (b - \mathbf{a}^T\underline{\mathbf{s}} - x) \bmod q$$

Trick 1: is to construct a polynomial equation from each sample that will clearly be = 0 (mod **q**)

Arora-Ge Algebraic Attack (continued)

We can construct **m** polynomials of degree **|E|** and solving the system of polynomial equations will yield **s**

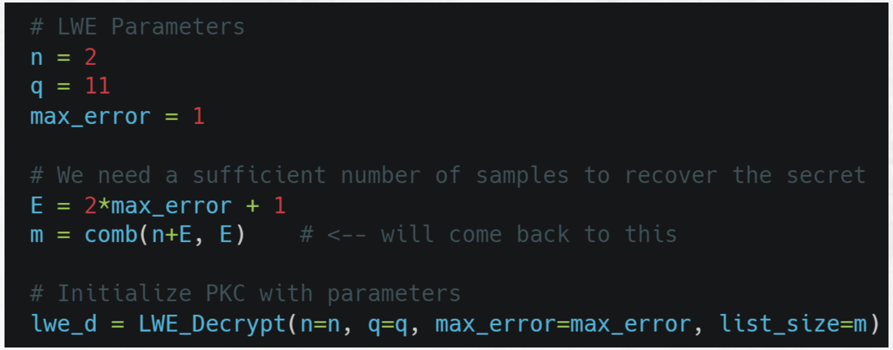
$$\{f_{\mathbf{a}_i,b_i}(\mathbf{s}) = 0 \bmod q\}_{i=1}^{m}$$

However, solving systems of polynomial equations (even of degree 2) is NP-hard.

Trick 2: is the observation that if we have enough samples, we can **linearize** the polynomials equations. Then solve in O(n^3) and we obtain **s** with high probability.

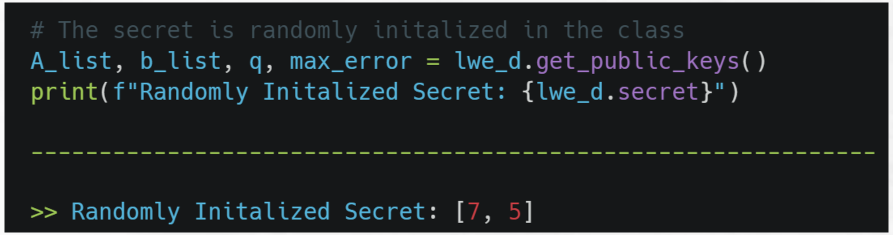
Example: Setup

Let us understand by attacking a LWE with a 2D secret vector and errors ∈ {-1, 0, +1}



```
# LWE Parameters
n = 2
q = 11
max_error = 1

# We need a sufficient number of samples to recover the secret
E = 2*max_error + 1
m = comb(n+E, E)    # <-- will come back to this

# Initialize PKC with parameters
lwe_d = LWE_Decrypt(n=n, q=q, max_error=max_error, list_size=m)
```

We have $^5C_3 = 10$ samples

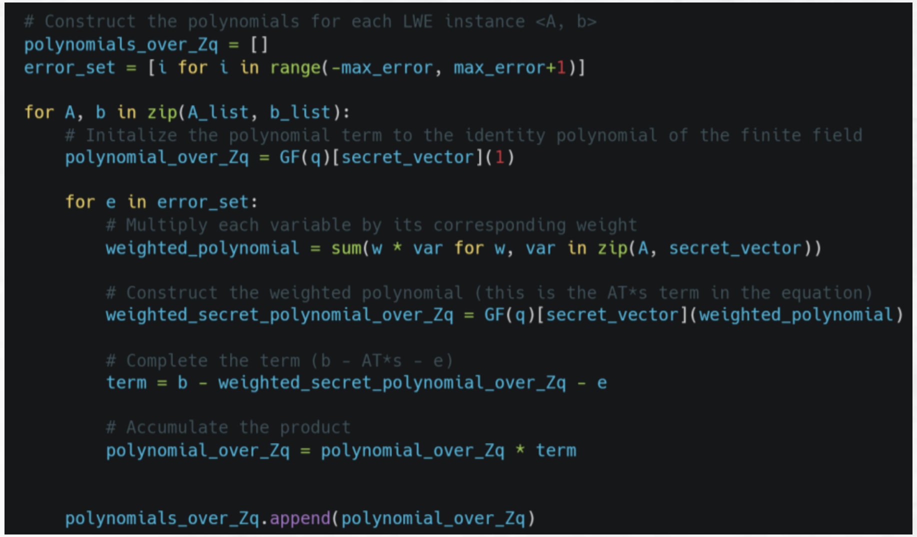


```
# The secret is randomly initalized in the class
A_list, b_list, q, max_error = lwe_d.get_public_keys()
print(f"Randomly Initalized Secret: {lwe_d.secret}")

--------------------------------------------------------------

>> Randomly Initalized Secret: [7, 5]
```

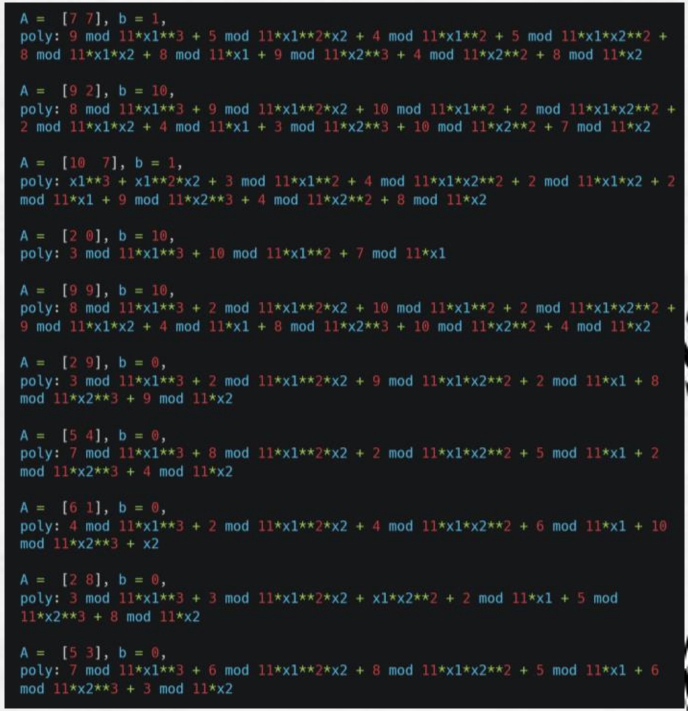
The secret vector **s** is [7, 5]

Example: Constructing Polynomials

Leveraging the *sympy* library we can construct the system of polynomials.



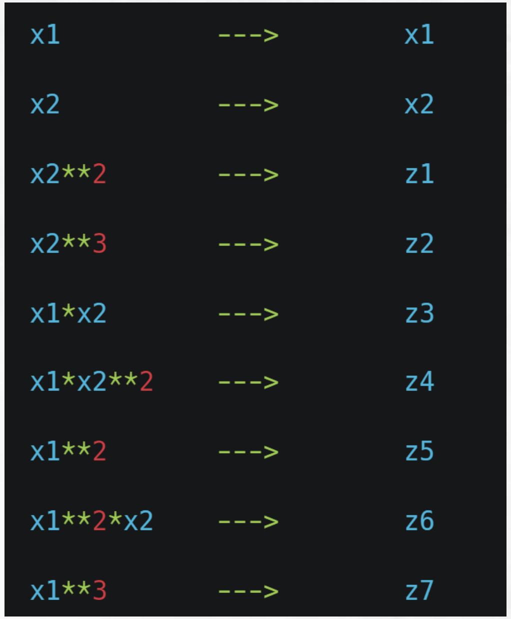
```python
# Construct the polynomials for each LWE instance <A, b>
polynomials_over_Zq = []
error_set = [i for i in range(-max_error, max_error+1)]

for A, b in zip(A_list, b_list):
    # Initalize the polynomial term to the identity polynomial of the finite field
    polynomial_over_Zq = GF(q)[secret_vector](1)

    for e in error_set:
        # Multiply each variable by its corresponding weight
        weighted_polynomial = sum(w * var for w, var in zip(A, secret_vector))

        # Construct the weighted polynomial (this is the AT*s term in the equation)
        weighted_secret_polynomial_over_Zq = GF(q)[secret_vector](weighted_polynomial)

        # Complete the term (b - AT*s - e)
        term = b - weighted_secret_polynomial_over_Zq - e

        # Accumulate the product
        polynomial_over_Zq = polynomial_over_Zq * term


    polynomials_over_Zq.append(polynomial_over_Zq)
```



```
A =  [7 7], b = 1,
poly: 9 mod 11*x1**3 + 5 mod 11*x1**2*x2 + 4 mod 11*x1**2 + 5 mod 11*x1*x2**2 +
8 mod 11*x1*x2 + 8 mod 11*x1 + 9 mod 11*x2**3 + 4 mod 11*x2**2 + 8 mod 11*x2

A =  [9 2], b = 10,
poly: 8 mod 11*x1**3 + 9 mod 11*x1**2*x2 + 10 mod 11*x1**2 + 2 mod 11*x1*x2**2 +
2 mod 11*x1*x2 + 4 mod 11*x1 + 3 mod 11*x2**3 + 10 mod 11*x2**2 + 7 mod 11*x2

A =  [10  7], b = 1,
poly: x1**3 + x1**2*x2 + 3 mod 11*x1**2 + 4 mod 11*x1*x2**2 + 2 mod 11*x1*x2 + 2
mod 11*x1 + 9 mod 11*x2**3 + 4 mod 11*x2**2 + 8 mod 11*x2

A =  [2 0], b = 10,
poly: 3 mod 11*x1**3 + 10 mod 11*x1**2 + 7 mod 11*x1

A =  [9 9], b = 10,
poly: 8 mod 11*x1**3 + 2 mod 11*x1**2*x2 + 10 mod 11*x1**2 + 2 mod 11*x1*x2**2 +
9 mod 11*x1*x2 + 4 mod 11*x1 + 8 mod 11*x2**3 + 10 mod 11*x2**2 + 4 mod 11*x2

A =  [2 9], b = 0,
poly: 3 mod 11*x1**3 + 2 mod 11*x1**2*x2 + 9 mod 11*x1*x2**2 + 2 mod 11*x1 + 8
mod 11*x2**3 + 9 mod 11*x2

A =  [5 4], b = 0,
poly: 7 mod 11*x1**3 + 8 mod 11*x1**2*x2 + 2 mod 11*x1*x2**2 + 5 mod 11*x1 + 2
mod 11*x2**3 + 4 mod 11*x2

A =  [6 1], b = 0,
poly: 4 mod 11*x1**3 + 2 mod 11*x1**2*x2 + 4 mod 11*x1*x2**2 + 6 mod 11*x1 + 10
mod 11*x2**3 + x2

A =  [2 8], b = 0,
poly: 3 mod 11*x1**3 + 3 mod 11*x1**2*x2 + x1*x2**2 + 2 mod 11*x1 + 5 mod
11*x2**3 + 8 mod 11*x2

A =  [5 3], b = 0,
poly: 7 mod 11*x1**3 + 6 mod 11*x1**2*x2 + 8 mod 11*x1*x2**2 + 5 mod 11*x1 + 6
mod 11*x2**3 + 3 mod 11*x2
```

Example: Linearization

In the simplest of terms, linearization is the substitution of non-linear terms with independent variables.



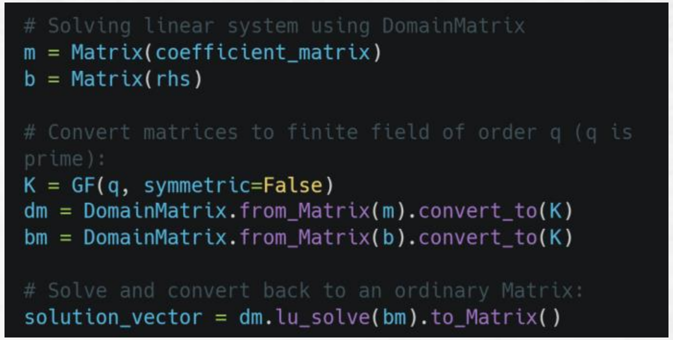
```
x1          --->        x1

x2          --->        x2

x2**2       --->        z1

x2**3       --->        z2

x1*x2       --->        z3

x1*x2**2    --->        z4

x1**2       --->        z5

x1**2*x2    --->        z6

x1**3       --->        z7
```

Terms of the secret vector are at the top. Same order is followed when constructing the matrix.

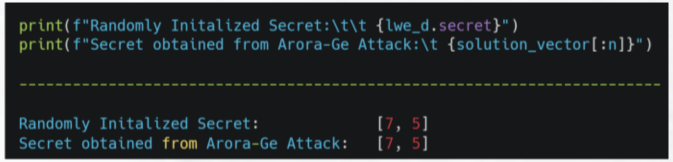
- However, this increases the number of unknown variables (for n=2 we now have 10 variables).
- Hence, we need atleast 10 samples.
- We can solve the linear system using LU decomposition in the finite field.

Example: Solving Linear System

We can use *sympy*'s solver to solve the system in $\mathbf{Z_q}$. In our setup the secret vector is the first n entries of the solution vector.



```python
# Solving linear system using DomainMatrix
m = Matrix(coefficient_matrix)
b = Matrix(rhs)

# Convert matrices to finite field of order q (q is
prime):
K = GF(q, symmetric=False)
dm = DomainMatrix.from_Matrix(m).convert_to(K)
bm = DomainMatrix.from_Matrix(b).convert_to(K)

# Solve and convert back to an ordinary Matrix:
solution_vector = dm.lu_solve(bm).to_Matrix()
```



```
print(f"Randomly Initalized Secret:\t\t {lwe_d.secret}")
print(f"Secret obtained from Arora-Ge Attack:\t {solution_vector[:n]}")

---------------------------------------------------------------------

Randomly Initalized Secret:             [7, 5]
Secret obtained from Arora-Ge Attack:   [7, 5]
```

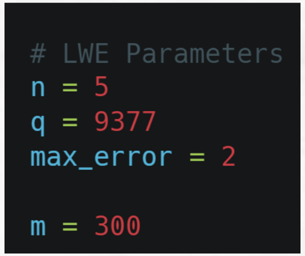
Success!

$$\begin{bmatrix} 2 & 10 & 9 & 4 & 8 & 9 & 3 & 4 & 1 \\ 7 & 7 & 3 & 7 & 6 & 10 & 3 & 10 & 7 \\ 1 & 6 & 7 & 8 & 6 & 4 & 6 & 8 & 9 \\ 0 & 0 & 7 & 4 & 9 & 3 & 8 & 9 & 9 \\ 6 & 10 & 6 & 2 & 5 & 8 & 7 & 7 & 8 \\ 4 & 1 & 0 & 10 & 0 & 10 & 0 & 7 & 2 \\ 6 & 10 & 9 & 9 & 2 & 3 & 5 & 4 & 3 \\ 3 & 5 & 4 & 3 & 7 & 1 & 1 & 5 & 1 \\ 6 & 9 & 9 & 5 & 1 & 10 & 4 & 3 & 8 \\ 6 & 3 & 1 & 1 & 4 & 6 & 4 & 1 & 8 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 6 \\ 5 \\ 5 \\ 5 \\ 0 \\ 1 \\ 6 \\ 6 \\ 6 \end{bmatrix}$$

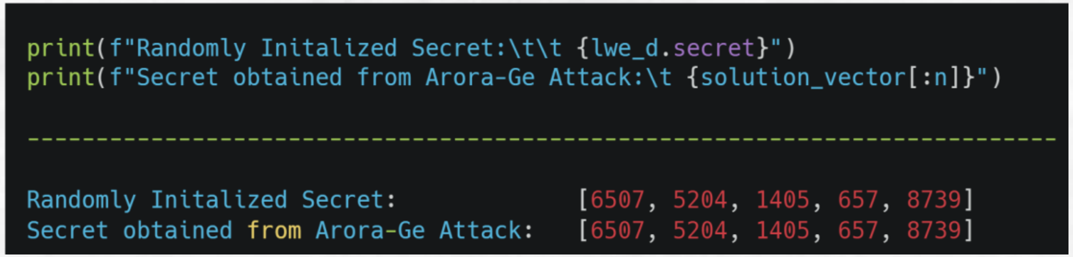
The coefficient matrix (m) and b vector.

More Examples

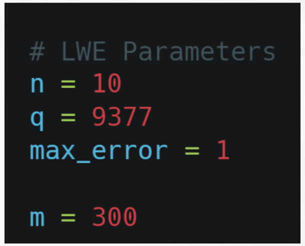
What about increasing **n, q** and **E**?



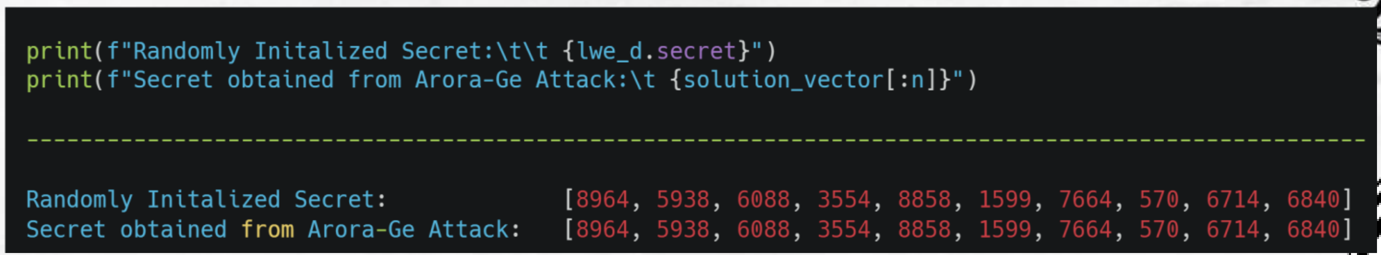
```
# LWE Parameters
n = 5
q = 9377
max_error = 2

m = 300
```



```
print(f"Randomly Initalized Secret:\t\t {lwe_d.secret}")
print(f"Secret obtained from Arora-Ge Attack:\t {solution_vector[:n]}")

----------------------------------------------------------------------------

Randomly Initalized Secret:             [6507, 5204, 1405, 657, 8739]
Secret obtained from Arora-Ge Attack:   [6507, 5204, 1405, 657, 8739]
```



```
# LWE Parameters
n = 10
q = 9377
max_error = 1

m = 300
```



```
print(f"Randomly Initalized Secret:\t\t {lwe_d.secret}")
print(f"Secret obtained from Arora-Ge Attack:\t {solution_vector[:n]}")

--------------------------------------------------------------------------------------------------

Randomly Initalized Secret:             [8964, 5938, 6088, 3554, 8858, 1599, 7664, 570, 6714, 6840]
Secret obtained from Arora-Ge Attack:   [8964, 5938, 6088, 3554, 8858, 1599, 7664, 570, 6714, 6840]
```

Both needed the same number of samples to crack.

Number of Samples Required

Let us revisit the theory,

$$f_{\mathbf{a},b}(\mathbf{s}) = \prod_{x \in E} (b - \mathbf{a}^T \underline{\mathbf{s}} - x) \bmod q$$

$\mathbf{a}$:= vector of coefficients

$\mathbf{b} := \mathbf{a}^T \underline{\mathbf{s}} + \mathbf{e}$ where errors, $\mathbf{e} \in \mathbf{E} \subseteq \mathbf{Z_q}$

- Each polynomial is of degree **|E|** and there are **n** variables in the secret vector **s**.
- This implies the number of monomials in the expansion is $^{(n + |E|)}\mathbf{C}_{|E|}$
- After linearization, all of them would be mapped to independent variables, thus we would require at least the same number of samples to solve for **s**. More samples would gurantee unique solution.

Further Analysis

- Thus, if the errors are bounded by **T**, i.e., **e** ∈ {-T, …, 0, …, T} each polynomial **f(s)** would be of degree **2T + 1**.
- We would require **$O(n^{T+1})$** samples to successfully recover **s**.
- This does not scale well, T = 1 already requires $O(n^3)$ samples.
- Improved versions of the attack require a sublinear amount of samples ($m = n + O(n/\log(n))$) for the Binary LWE problem. Hence it is still a relevant attack.

RELEVANCE OF LWE & RLWE

1) Quantum Safe Algorithms

i) Can be reduced to Shortest Vector Problem in Lattice Cryptography.

2) Fully Homomorphic Crypto

i) BGV cipher based on RLWE

ii) Needs high value of q such that:

$$no_{selected} \times \text{max\_}error \leq \frac{1}{x}\left\lfloor\frac{q}{4}\right\rfloor$$: For Addition of x ciphers

$$no_{selected} \times \text{max\_}error \leq \left\lfloor\frac{q}{4}\right\rfloor^{1/x}$$: For Multiplication of x ciphers

Contributions

Both of us contributed equally ☺ and managed to work on all aspects of the project.

Name	Cryptosystems	Attacks
Dhanyamraju Harsh Rao	Implemented encryption and decryption of LWE & RLWE	Noise Reduction Attack (Not shown)
George Rahul	Implemented padding for above	Arora-Ge Attack

Thanks!

Any questions?