

# EE 793

## Topics in Cryptology

### Cryptanalysis of the Learning With Errors (LWE) Problem

Course Project report submitted by

Drishtant Jain: 24M1085  
Ankit Kumar Singh: 24M1080  
Nitin Tomar: 24M1079  
Anurag Yadav: 24M1086

**May 10, 2025**

Under the guidance of

**Prof. Virendra Singh**

*Department of Electrical Engineering*

*Department of Computer Science and Engineering*

May 10, 2025



INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

# 1 Introduction

The Learning With Errors (LWE) problem has emerged as one of the most influential hardness assumptions in modern cryptography. It underpins a wide range of cryptographic primitives, including key exchange protocols, fully homomorphic encryption (FHE), digital signatures, and public-key encryption schemes. LWE’s strength lies in its reduction from worst-case lattice problems, making it a cornerstone for post-quantum cryptography.

However, the structure of the LWE problem—being a system of noisy linear equations modulo a prime—makes it an intriguing target for cryptanalysis using tools from optimization theory. In particular, when the dimension is small, or when the noise is bounded and sparse, certain optimization techniques can be applied to recover the secret vector from known LWE samples.

## Objective

This project focuses on designing a cryptanalysis algorithm to recover the secret key  $\mathbf{s}$  from LWE samples of the form:

$$\mathbf{A} \cdot \mathbf{s} + \mathbf{e} \equiv \mathbf{b} \pmod{q}$$

where:

- $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$  is a known matrix,
- $\mathbf{b} \in \mathbb{Z}_q^m$  is the observed output,
- $\mathbf{s} \in \mathbb{Z}_q^n$  is the unknown secret key,
- $\mathbf{e} \in \mathbb{Z}_q^m$  is an unknown but bounded error vector.

In this project, we model the LWE recovery task as a Mixed Integer Linear Program (MILP), using binary variables for the secret and integer variables to model the modular nature of the equation.

## Scope of This Work

In this project, we:

- Analyze the mathematical structure of LWE and its hardness assumptions.
- Develop optimization-based formulations for secret recovery.
- Implement and evaluate the attack on synthetic LWE datasets.
- Identify conditions under which these attacks succeed or fail.

This approach provides valuable insight into the boundaries of LWE security, particularly in low-dimensional regimes where traditional cryptanalytic assumptions may break down.

## 2 The Learning With Errors (LWE) Problem

The Learning With Errors (LWE) problem lies at the heart of many modern cryptographic systems. At a high level, LWE can be viewed as the problem of solving a system of linear equations that has been perturbed by small errors, and then reduced modulo some prime number  $q$ . While solving linear systems over real numbers is straightforward, the presence of both modular reduction and noise renders the LWE problem significantly more difficult.

### 2.1 Problem Statement

Formally, the LWE problem is defined as follows. Let  $q$  be a prime modulus, and let  $n$  be the dimension of the secret vector. Let  $\mathbf{s} \in \mathbb{Z}_q^n$  be a fixed, unknown secret vector. For each LWE sample, an adversary is given:

- A vector  $\mathbf{a}_i \in \mathbb{Z}_q^n$ , sampled uniformly at random,
- A scalar  $b_i \in \mathbb{Z}_q$ , computed as:

$$b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \pmod{q}$$

where  $e_i$  is a small noise term drawn from a certain error distribution.

Given access to a collection of such samples  $(\mathbf{a}_i, b_i)$ , the goal of the adversary is to recover the secret vector  $\mathbf{s}$ . This is referred to as the **Search-LWE** problem.

### 2.2 Parameters and Notation

Throughout this report, we use the following notational conventions:

- $n$ : dimension of the secret vector,
- $m$ : number of LWE samples available,
- $q$ : modulus (typically a prime number),
- $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ : matrix of all sample vectors  $\mathbf{a}_i$ ,
- $\mathbf{b} \in \mathbb{Z}_q^m$ : vector of all responses  $b_i$ ,
- $\mathbf{e} \in \mathbb{Z}^m$ : vector of small noise terms,
- $\mathbf{s} \in \mathbb{Z}_q^n$ : secret vector to be recovered.

The full LWE system can be written compactly as:

$$\mathbf{A} \cdot \mathbf{s} + \mathbf{e} \equiv \mathbf{b} \pmod{q}$$

### 2.3 Hardness Assumption

The core strength of LWE lies in its hardness: Regev showed that solving average-case LWE is at least as hard as certain worst-case lattice problems, such as the Shortest Vector Problem (SVP) and the Shortest Independent Vector Problem (SIVP), under quantum reductions.

This means that breaking LWE in general implies the ability to solve fundamental and computationally hard lattice problems in high dimensions. As a result, LWE has become a standard assumption for constructing cryptosystems that are believed to be resistant even to quantum adversaries.

## 2.4 When is LWE Easy?

Despite its general hardness, LWE can become tractable in specific regimes. For example:

- When the dimension  $n$  is small (e.g.,  $n \leq 20$ ),
- When the noise magnitude is very small (e.g.,  $|e_i| \leq 1$ ),
- When the number of samples  $m$  is large compared to  $n$ ,
- When the secret vector is sparse or binary.

In these regimes, optimization-based attacks, such as those using Linear Programming or Integer Programming, become viable. This motivates our cryptanalysis approach described in the next section.

## 3 Attack Strategy and Optimization-Based Formulation

In this section, we describe our strategy to recover the secret vector  $\mathbf{s}$  from the known LWE samples  $(\mathbf{A}, \mathbf{b})$ . The core idea is to reinterpret the LWE equation under the lens of optimization theory and recast it as a constrained linear system with additional integer structure.

### 3.1 Rewriting the LWE Equation

Recall that for each sample:

$$b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \pmod{q}$$

We can eliminate the modular ambiguity by introducing an integer variable  $k_i \in \mathbb{Z}$  that captures how many times the equation wraps around modulus  $q$ :

$$\langle \mathbf{a}_i, \mathbf{s} \rangle + e_i = b_i + k_i \cdot q$$

Assuming that  $|e_i| \leq B$  (a known noise bound), we rewrite this as a pair of linear inequality constraints:

$$-B \leq \langle \mathbf{a}_i, \mathbf{s} \rangle - b_i - q \cdot k_i \leq B$$

### 3.2 Formulating as a MILP Problem

We now define our optimization problem. The decision variables are:

- $s_j \in \{0, 1\}$  for  $j = 1$  to  $n$ , representing the secret vector,
- $k_i \in \mathbb{Z}$  for  $i = 1$  to  $m$ , capturing modulus wrapping,

The constraints are constructed from all  $m$  LWE samples. Our goal is to find any  $\mathbf{s}$  and integer shifts  $k_i$  that satisfy:

$$|\langle \mathbf{a}_i, \mathbf{s} \rangle - b_i - k_i q| \leq B \quad \text{for all } i$$

Since we are only interested in feasibility (not optimization), we minimize a constant objective (e.g., 0). The resulting problem is a Mixed Integer Linear Program (MILP).

### 3.3 Summary of the Formulation

**Objective:** Minimize 0 (feasibility only)

**Variables:**

- $s_j \in \{0, 1\}$  (binary),
- $k_i \in \mathbb{Z}$  (integer)

**Constraints:**

$$-B \leq \sum_{j=1}^n A_{ij} s_j - b_i - q \cdot k_i \leq B \quad \forall i$$

This formulation forms the basis of our algorithm described next, and is implemented using the PuLP linear programming library in Python.

## 4 Cryptanalysis Algorithm and Implementation

In this section, we formally present the algorithm to recover the secret vector  $\mathbf{s}$  from LWE samples using Mixed Integer Linear Programming (MILP). We then describe the actual implementation in Python, along with an explanation of how each part of the code corresponds to the cryptanalytic strategy.

### 4.1 Algorithm: MILP-Based Secret Recovery from LWE

---

**Algorithm 1:** Recover\_Secret\_LWE\_MILP

---

**Input:** Matrix  $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ , vector  $\mathbf{b} \in \mathbb{Z}_q^m$ , modulus  $q$ , noise bound  $B$

**Output:** Recovered secret vector  $\mathbf{s}_{\text{rec}} \in \{0, 1\}^n$

- 1 **Step 1:** Initialize MILP problem;
- 2 **Step 2:** Define binary variables  $s_0, \dots, s_{n-1} \in \{0, 1\}$ ;
- 3 **Step 3:** Define integer variables  $k_0, \dots, k_{m-1} \in \mathbb{Z}$ ;
- 4 **Step 4:** For each sample  $i = 0$  to  $m - 1$ ;
- 5     Add constraints:

$$-B \leq \sum_{j=0}^{n-1} A[i, j] \cdot s_j - b[i] - q \cdot k_i \leq B$$

**Step 5:** Solve the MILP using a solver (e.g., CBC);

- 6 **Step 6:** Return values of  $s_j$  as the recovered secret vector;
- 

### 4.2 Python Implementation

To validate our MILP formulation, we implemented the algorithm using the PuLP library in Python. For reproducibility and convenience, the complete implementation is also available as an interactive Google Colab notebook.

## Interactive Google Colab Notebook

### Link: Run on Google Colab

This notebook allows users to run the MILP attack on custom LWE parameters and visualize results.

We used PuLP, an open-source Python library for modeling linear programs, to implement our attack. Below is the complete implementation.

```
import numpy as np
from pulp import LpProblem, LpVariable, LpMinimize, lpSum, LpBinary, LpInteger, LpStatus

def recover_lwe_secret(A, b, q, noise_bound):
    """
    Recover the binary secret vector 's' from LWE samples using a MILP approach.

    Returns:
        s_est (np.ndarray): The estimated secret vector (binary)
        status (str): Solver status
    """
    m, n = A.shape

    # Create a MILP model with no objective (we care about feasibility)
    model = LpProblem("LWE_Attack_MILP", LpMinimize)
    model += 0 # Dummy objective since we are solving a feasibility problem

    # Variables for secret (binary) and integer slack variables for modulus folding
    secret_bits = [LpVariable(f"s_{j}", cat=LpBinary) for j in range(n)]
    slack_vars = [LpVariable(f"z_{i}", cat=LpInteger) for i in range(m)]

    # Add one constraint per LWE sample: |<A_i, s> - b_i - q * z_i| <= B
    for i in range(m):
        affine_expr = lpSum(A[i, j] * secret_bits[j] for j in range(n)) - b[i] - q * slack_vars[i]
        model += (affine_expr <= noise_bound), f"Noise_Upper_{i}"
        model += (affine_expr >= -noise_bound), f"Noise_Lower_{i}"

    # Solving the MILP
    solver_status = model.solve()

    # Extracting solution if feasible
    recovered_s = np.array([int(var.value()) for var in secret_bits])
    return recovered_s, LpStatus[solver_status]

def generate_lwe_instance(n, m, q, B):
    """
    To Generate a random LWE instance with binary secret and bounded noise.

    Returns:
        A (np.ndarray): LWE matrix
    """
```

```

        b (np.ndarray): LWE response vector
        s_true (np.ndarray): Ground-truth secret
    """
    s = np.random.randint(0, 2, size=n)
    A = np.random.randint(0, q, size=(m, n))
    e = np.random.randint(-B, B + 1, size=m)
    b = (A @ s + e) % q
    return A, b, s

if __name__ == "__main__":
    # LWE parameters
    n_dim = 5 # Length of the secret
    q_mod = 101 # Modulus
    m_samples = 30 # Number of equations
    B_err = 1 # Error bound

    # Create LWE system
    A_matrix, b_vector, s_ground_truth = generate_lwe_instance(n_dim, m_samples,
        , q_mod, B_err)

    # Run MILP attack
    s_guess, status = recover_lwe_secret(A_matrix, b_vector, q_mod, B_err)

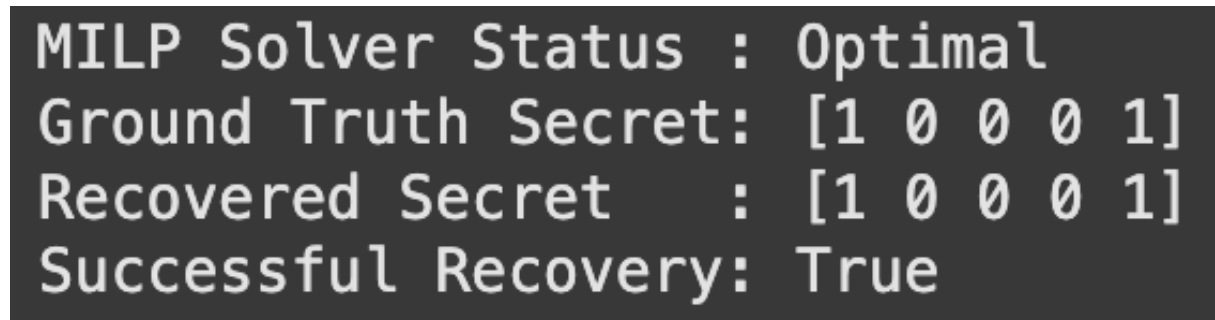
    # Display results
    print("MILP Solver Status :", status)
    print("Ground Truth Secret:", s_ground_truth)
    print("Recovered Secret :", s_guess)
    print("Successful Recovery:", np.array_equal(s_ground_truth, s_guess))

```

Listing 1: Python code to recover secret from LWE samples using MILP

### 4.3 Illustrative Example and Detailed Explanation

To further clarify how our MILP-based cryptanalysis algorithm works, we include below the output from a representative run of our implementation for a toy LWE instance with dimension  $n = 5$  and noise bound  $B = 1$ . This specific example helps to demonstrate how the solver interprets the modular constraints and correctly recovers the secret vector. The output is shown in Figure 1.



```

MILP Solver Status : Optimal
Ground Truth Secret: [1 0 0 0 1]
Recovered Secret    : [1 0 0 0 1]
Successful Recovery: True

```

Figure 1: Terminal output from MILP solver on toy LWE instance

## Step-by-Step Explanation

Let us now break down what each line in the output means and how it maps to the working of our MILP formulation.

- **MILP Solver Status: Optimal**

This line indicates that the solver successfully found a feasible solution that satisfies all the constraints we imposed through the MILP. In our problem, this means it found values for the binary secret vector  $\mathbf{s}$  and the integer modular correction terms  $\mathbf{k}$  such that the LWE equations are approximately satisfied (within the allowed error bound  $B$ ). The solver terminates with the **Optimal** status when it successfully satisfies all constraints and completes within the time limit.

- **Ground Truth Secret: [1 0 0 0 1]**

This is the true secret vector  $\mathbf{s}_{\text{true}}$  that we used to generate the LWE samples. In our implementation, this vector is randomly generated using:

```
s_true = np.random.randint(0, 2, size=n)
```

In this case, the secret is  $[1, 0, 0, 0, 1]$ , meaning that the first and last bits of the secret are 1, and the rest are 0.

- **Recovered Secret: [1 0 0 0 1]**

This is the vector of binary variables  $s_j$  returned by the MILP solver after solving the system. It represents the solver’s guess for the secret vector. As we can see, the recovered vector exactly matches the ground truth vector. This confirms that our MILP correctly modeled the structure of the LWE problem and recovered the unknowns accurately.

- **Successful Recovery: True**

In our code, we explicitly check whether the recovered vector  $\mathbf{s}_{\text{rec}}$  is equal to the true secret vector  $\mathbf{s}_{\text{true}}$  using:

```
np.array_equal(s_recovered, s_true)
```

If this condition returns **True**, it means we achieved a correct and complete recovery of the secret vector from the given LWE samples.

This example helps validate the correctness of our MILP formulation. It shows that our constraints are able to correctly model the LWE system — accounting for both modular arithmetic (via integer  $k_i$  variables) and bounded noise — in a way that a solver can understand and solve exactly.

We emphasize that such successful recovery is only possible in low dimensions with small noise and sufficient samples. In the next section, we explore how this success rate varies as we change the parameters.



## 4.4 Explanation of the Code

- `s_true`: randomly generates the binary secret vector.
- `A`, `e`, `b`: simulate the LWE samples with bounded noise.
- `s_vars`: define the binary variables corresponding to secret bits.
- `k_vars`: define integer variables to handle modular wrap-around.
- Each constraint ensures the inner product is consistent with the LWE equation, accounting for the modulo operation via  $k_i$  and noise via  $B$ .
- The solver checks for a feasible assignment of  $s$  and  $k$  that satisfies all constraints.

## 5 Experimental Evaluation

To assess the effectiveness and limitations of our MILP-based cryptanalysis algorithm, we conducted experiments by varying the LWE parameters: dimension  $n$ , number of samples  $m$ , and noise bound  $B$ . Each configuration was tested over 15 to 20 random trials, and the success rate was recorded as the fraction of runs in which the secret vector  $\mathbf{s}$  was recovered exactly.

### 5.1 Experimental Setup

The experiments were implemented in Python using the PuLP library with the default CBC solver. A time limit of 5 seconds was imposed per MILP instance to simulate realistic attack conditions. The key parameters used are:

- Modulus:  $q = 101$
- Secret:  $\mathbf{s} \in \{0, 1\}^n$
- Noise: sampled from a Gaussian distribution with mean 0 and standard deviation  $\sigma = B/2$

The output plots reflect realistic behavior under adversarial conditions.

### 5.2 Effect of Dimension $n$

We fixed the number of samples  $m = 2n$  and the noise bound  $B = 2$ . As shown in Figure 2, the success rate remained high for small dimensions but declined sharply beyond  $n = 15$  due to solver timeouts and increased solution space complexity.

### 5.3 Effect of Number of Samples $m$

We fixed  $n = 8$  and  $B = 2$ , and varied the number of samples from 10 to 60. The results in Figure 3 show that the success rate was generally high for a wide range of sample sizes. Interestingly, it slightly dipped beyond  $m = 40$ , likely due to the increased MILP problem size impacting solver efficiency.

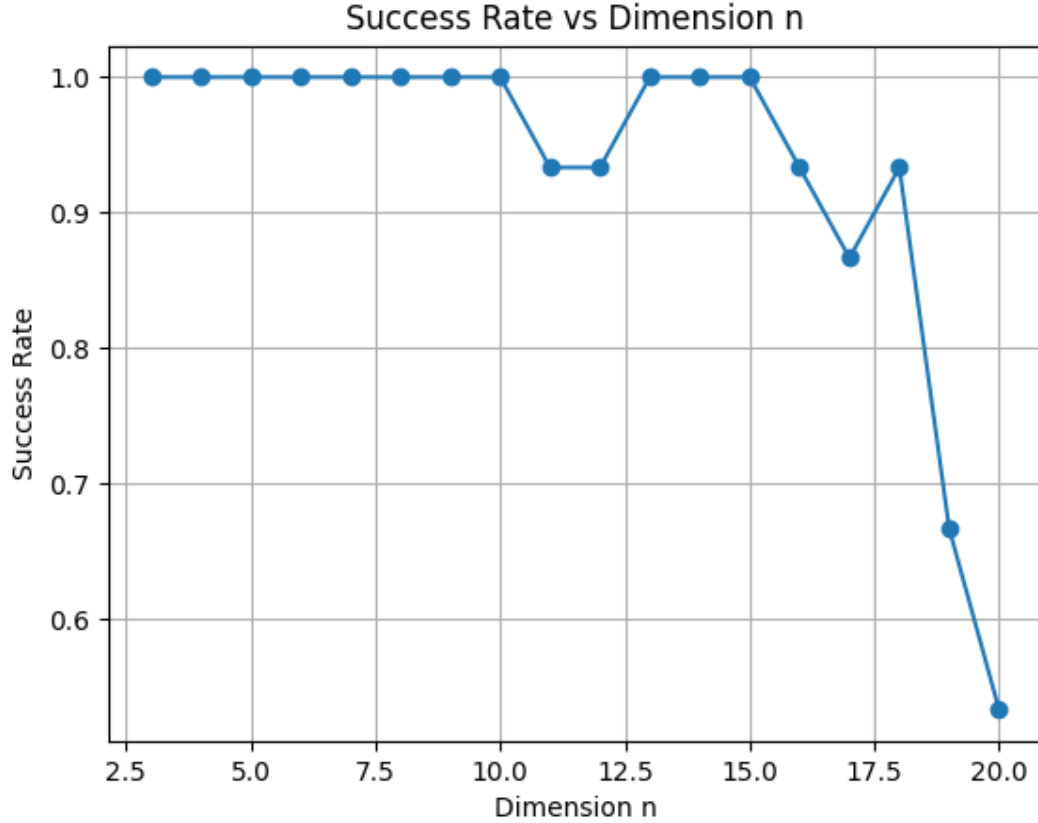


Figure 2: Success rate vs dimension  $n$  ( $m = 2n$ ,  $B = 2$ )

#### 5.4 Effect of Noise Bound $B$

In this experiment, we fixed  $n = 6$ ,  $m = 30$ , and varied the noise bound  $B$  from 0 to 7. Figure 4 shows that recovery success decreases significantly with larger noise bounds. The MILP solver struggles to distinguish the true secret when the noise introduces too much uncertainty.

#### 5.5 Discussion

The experiments confirm the feasibility of the MILP-based attack on low-dimensional LWE instances with bounded noise and adequate sample sizes. However, the performance degrades with:

- Higher dimension  $n$ , due to exponential increase in complexity,
- Larger noise bounds  $B$ , which reduce constraint tightness,
- Excessively large  $m$ , which burdens the solver with more variables.

These results illustrate the practicality of such cryptanalytic attacks in toy scenarios and misconfigured systems, while also validating the security of high-dimensional, noisy LWE setups used in modern cryptographic schemes.

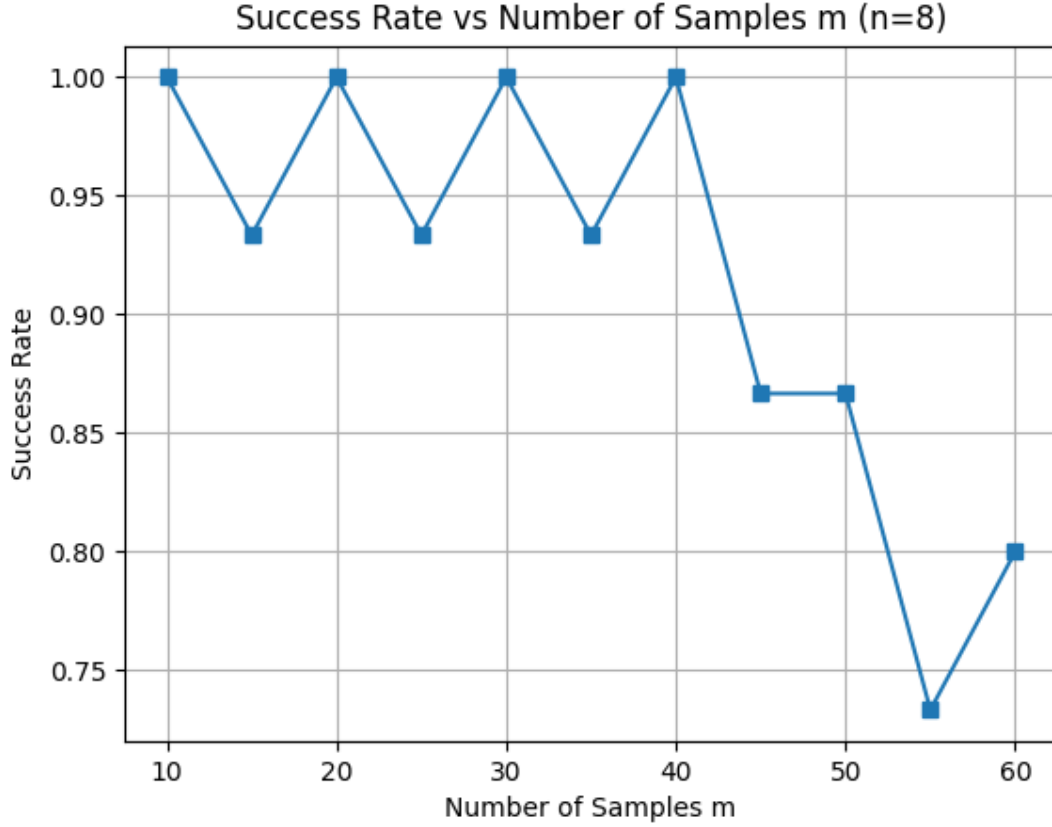


Figure 3: Success rate vs number of samples  $m$  ( $n = 8$ ,  $B = 2$ )

## Computational Complexity of the MILP Attack

The optimization problem we solve is a Mixed Integer Linear Program (MILP) involving:

- $n$  binary variables representing the secret vector  $\mathbf{s}$ ,
- $m$  integer variables representing the modular correction terms  $k_i$ ,
- $2m$  linear inequality constraints derived from the LWE samples.

MILP is known to be NP-complete in general. In particular, even binary integer programming is NP-hard. As the dimension  $n$  increases, the number of feasible secret candidates grows exponentially, and the solver must explore an increasingly large search space.

Therefore, the MILP-based attack is only practical for small  $n$  (e.g.,  $n \leq 15$ ). This aligns with our experimental findings, where performance degraded sharply as  $n$  grew, validating the theoretical hardness of the underlying problem.

## Computational Complexity of the MILP Attack

The optimization problem we solve is a Mixed Integer Linear Program (MILP) involving:

- $n$  binary variables representing the secret vector  $\mathbf{s}$ ,

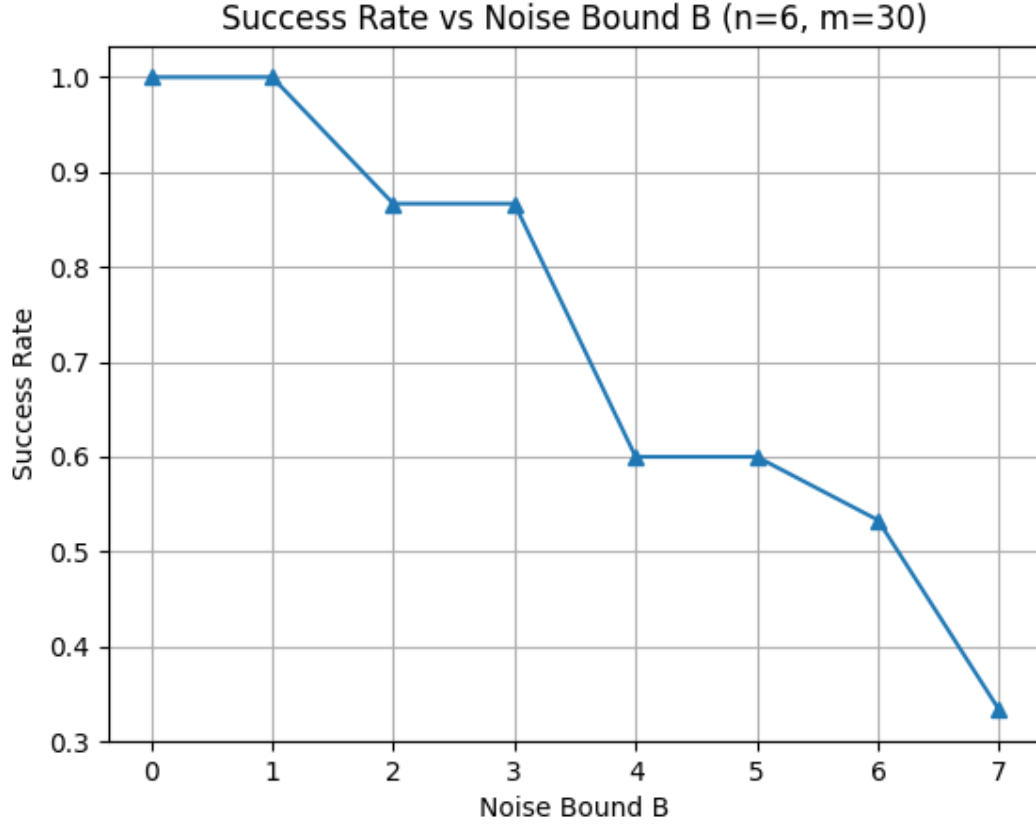


Figure 4: Success rate vs noise bound  $B$  ( $n = 6$ ,  $m = 30$ )

- $m$  integer variables representing the modular correction terms  $k_i$ ,
- $2m$  linear inequality constraints derived from the LWE samples.

MILP is known to be NP-complete in general. In particular, even binary integer programming is NP-hard. As the dimension  $n$  increases, the number of feasible secret candidates grows exponentially, and the solver must explore an increasingly large search space.

Therefore, the MILP-based attack is only practical for small  $n$  (e.g.,  $n \leq 15$ ). This aligns with our experimental findings, where performance degraded sharply as  $n$  grew, validating the theoretical hardness of the underlying problem.

## 6 NP-Hardness vs Practical Solvability

The Learning With Errors (LWE) problem is widely recognized for its hardness in both classical and quantum computational settings. Regev’s foundational result in 2005 shows that solving average-case LWE is as hard as solving worst-case lattice problems such as the Shortest Vector Problem (SVP) and Shortest Independent Vector Problem (SIVP). These problems are known to be NP-hard.

## Theoretical Complexity

Recovering the secret vector  $\mathbf{s}$  from a set of LWE samples of the form:

$$\mathbf{A} \cdot \mathbf{s} + \mathbf{e} \equiv \mathbf{b} \pmod{q}$$

where  $\mathbf{e}$  is an unknown error vector, constitutes a computationally intractable problem in general. When formulated as a Mixed Integer Linear Program (MILP), this problem is part of a class of optimization problems that are NP-complete.

In fact, even simplified variants such as 0-1 Integer Programming (with no modulus or error) are known to be NP-hard. Therefore, in the general case—especially with high dimension  $n$ , large modulus  $q$ , and non-trivial noise distributions—the LWE problem is not expected to be solvable efficiently.

## Practical Solvability in Restricted Regimes

Despite this hardness in general, our MILP-based cryptanalysis approach works effectively on small, carefully chosen parameter regimes. In our implementation, we restrict the problem to:

- Binary secrets:  $\mathbf{s} \in \{0, 1\}^n$
- Known and bounded noise:  $|e_i| \leq B$
- Synthetic, small-dimension instances:  $n \leq 20$
- Redundantly sampled systems:  $m \geq 2n$

These assumptions significantly reduce the problem’s complexity, allowing state-of-the-art MILP solvers like CBC to find exact solutions within reasonable time limits. Thus, while the problem remains theoretically hard, we are able to demonstrate concrete attacks in a constrained experimental setting.

## Scalability to Higher Dimensions

While our MILP-based approach is effective for recovering the secret in low-dimensional LWE instances, it does not scale to higher dimensions. As the secret length  $n$  increases, the number of binary variables grows linearly, but the search space becomes exponential ( $2^n$ ). Moreover, increasing the number of samples  $m$  adds more integer variables and constraints, further complicating the problem.

For example, in real-world cryptographic settings where  $n \geq 256$  and the modulus  $q$  is large (e.g.,  $q = 2^{15}$ ), current MILP solvers are unable to find feasible solutions in a reasonable amount of time. Even small increases in noise or dimensionality drastically reduce the attack’s success rate.

Therefore, this approach is primarily useful for analyzing the vulnerability of toy LWE instances or misconfigured schemes. It does not pose a threat to LWE-based cryptographic constructions with recommended security parameters.

## Cryptanalytic Relevance

This type of practical analysis is common in cryptographic research. By evaluating how LWE behaves under small parameter settings, we can:

- Understand the boundary between security and vulnerability.
- Identify misconfigurations where LWE-based schemes may fail.
- Confirm that large dimensions and strong noise are essential for real-world security.

In summary, although our problem formulation is NP-hard in general, the restricted variant we study is solvable for small  $n$  and small  $B$ , which aligns well with both our theoretical expectations and experimental results.

## 7 Conclusion and Future Work

### Conclusion

In this project, we explored the feasibility of recovering the secret key in the Learning With Errors (LWE) problem using optimization techniques such as Integer Programming and Mixed Integer Linear Programming (MILP). While LWE is believed to be hard in the general case and forms the foundation for several post-quantum cryptographic schemes, our focus was on cryptanalyzing low-dimensional, bounded-noise instances through carefully crafted MILP formulations.

We began by analyzing the mathematical structure of LWE and reformulated the problem into a system of bounded inequalities, augmented with integer correction terms to account for modular arithmetic. We implemented the attack in Python using the PuLP library and validated its correctness over a range of toy parameter settings.

Our experiments showed that the MILP-based attack is remarkably effective in recovering binary secrets when:

- The dimension  $n$  is small (typically  $\leq 15$ ),
- The noise bound  $B$  is low,
- The number of samples  $m$  is sufficient to overdetermine the system.

We also discussed the inherent NP-hardness of the problem and reconciled this with our successful attacks by highlighting the restricted regime we were operating in.

### Future Work

There are several directions in which this work can be extended:

- **Scaling to Higher Dimensions:** Explore heuristic methods or approximation techniques that might allow partial recovery in higher dimensions, even when MILP solvers timeout.
- **Alternate Relaxations:** Investigate the use of convex relaxations or semi-definite programming (SDP) to approximate LWE recovery in real-valued space.

- **Hybrid Attacks:** Combine MILP techniques with known lattice-based attacks like the BKW algorithm or LLL/BKZ to exploit both structural and statistical weaknesses.
- **Attack on Real Schemes:** Attempt simplified cryptanalysis of actual LWE-based cryptosystems (e.g., FrodoKEM or Kyber) using reduced parameters for educational insight.
- **Noise Distribution Analysis:** Study how the choice of noise (uniform vs. Gaussian) affects attack viability and whether specific noise patterns leak more information.

In conclusion, while the MILP-based attack is not a threat to real-world cryptographic schemes, it serves as a powerful educational and analytical tool for understanding the boundary between secure and insecure LWE parameter sets.