A Combinatorial Attack on Ternary Sparse Learning with Errors (sLWE)

Abul Kalam, Santanu Sarkar, and Willi Meier

Abstract

Sparse Learning With Errors (sLWE) is a novel problem introduced at Crypto 2024 by Jain et al., designed to enhance security in lattice-based cryptography against quantum attacks while maintaining computational efficiency. This paper presents the first third-party analysis of the ternary variant of sLWE, where both the secret and error vectors are constrained to ternary values. We introduce a combinatorial attack that employs a subsystem extraction technique followed by a Meet-in-the-Middle approach, effectively recovering the ternary secret vector. Our comprehensive analysis explores the attack's performance across various sparsity and modulus settings, revealing critical security limitations inherent in ternary sLWE.

Our analysis does not claim to present any attack on the proposal of Jain et al.; rather, it supports their assertion that sparse LWE is vulnerable for small secrets, particularly for ternary secrets and ternary errors. Notably, our findings indicate that the recommended parameters, which the developers claim provide security equivalent to LWE with a dimension of 1024, may not hold true for the ternary variant of sLWE. Our research highlights that, particularly with a modulus of 2^{64}, the secret key can be recovered in a practical timeframe, supporting the developers' claim of vulnerability in this case. Additionally, for configurations with moduli of 2^{32} and 2^{16}, we observe a significant reduction in the security margin. This suggests that the actual security level may be significantly weaker than intended. Overall, our work contributes crucial insights into the cryptographic robustness of ternary sLWE, emphasizing the need for further strengthening to protect against potential attacks and setting the stage for future research in this area.

Index Terms

Sparse Learning With Errors (sLWE), Ternary sLWE, Combinatorial Attack, Post-Quantum Cryptography

I. Introduction

In recent years, the study of lattice-based cryptography has become increasingly important, particularly in light of the growing concerns about quantum computing's potential to undermine traditional security methods. Lattice-based schemes, such as those based on the Learning With Errors (LWE) problem, are viewed as promising candidates for post-quantum cryptography due to their mathematical foundations that appear resistant to quantum attacks. Introduced by Regev in 2005 [1], the LWE problem serves as a cornerstone in this field, providing a robust framework for constructing secure cryptographic primitives. An LWE instance, represented as $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_M^{m\times n} \times \mathbb{Z}_M^m$, involves a matrix $\mathbf{A}$ generated randomly over $\mathbb{Z}_M$. The primary challenge lies in recovering a hidden secret vector $\mathbf{s} \in \mathbb{Z}_M^n$ from a noisy equation $\mathbf{A}\cdot\mathbf{s} = \mathbf{b} + \mathbf{e} \mod M$, where $\mathbf{e}$ represents a carefully introduced error term. The difficulty of this problem underpins many cryptographic tools, such as encryption schemes, digital signatures, and homomorphic encryption [2]–[8], making it vital for ensuring data confidentiality and integrity in various applications, from secure messaging to cloud computing.

The strength of the LWE problem is supported by its connections to some of the most challenging problems in lattice theory, which boosts confidence in the security of LWE-based systems. The reductions from these hard problems provide a theoretical foundation that allows cryptographers to build systems with provable security. However, this proven security

Abul Kalam and Santanu Sarkar are with the Department of Mathematics, Indian Institute of Technology Madras, Chennai, Tamil Nadu 600036, India (e-mail: abulkalam.sunny@gmail.com; sarkar.santanu.bir1@gmail.com).

University of Applied Sciences and Arts Northwestern Switzerland (FHNW), Windisch, Switzerland. e-mail: willimeier48@gmail.com

often applies only within specific parameter ranges, leading to concerns about practical efficiency in real-world applications. Consequently, many implementations of LWE choose parameters that go beyond these proven ranges to enhance performance, a trade-off that must be carefully managed. A significant factor in achieving this balance is the size and structure of the secret key, which influences not only the security of the system but also its computational efficiency. This has led to the adoption of binary, ternary, or sparse vector forms in various LWE systems [9]–[15]. By exploring these alternatives, researchers aim to optimize both security and efficiency, ultimately enhancing the viability of LWE in practical scenarios.

Moreover, the significance of LWE and its variants extends beyond their theoretical foundations; they serve as critical building blocks for a wide range of cryptographic primitives essential in our digital society. As applications of cryptography expand, spanning secure communications, privacy-preserving data sharing, and blockchain technologies, the need for robust and efficient cryptographic methods becomes paramount. These advancements not only ensure the security of individual systems but also contribute to the overall resilience of the digital ecosystem against various attack vectors. Ongoing research into LWE's properties and potential applications is vital for addressing the challenges posed by evolving technological threats and for ensuring the long-term integrity of cryptographic practices.

Despite the development of various methods to address the LWE problem, the cryptographic security of LWE with sparse or ternary secrets remains an area of ongoing inquiry. One promising method is combinatorial attacks, which take advantage of smaller secrets to lower the complexity of attacks, especially when combined with lattice reduction techniques. Among these, the Meet-in-the-Middle (MitM) attack, proposed by Odlyzko [16], stands out for its effectiveness, achieving an attack complexity of $S^{0.5}$, where S represents the search space size for the secret key.

Building on this foundation, May introduced a new MitM combinatorial attack called *Meet-LWE*, which significantly improves upon previous methods with an asymptotic complexity of $S^{0.25}$ [17]. This innovative approach combines Howgrave-Graham's representation technique [18] with a tree-based structure that efficiently organizes secrets and their hash values. May's method divides the LWE secret $\mathbf{s} \in \{0, \pm 1\}^n$ into two parts, $\mathbf{s_1}$ and $\mathbf{s_2}$, also in $\{0, \pm 1\}^n$. By making educated guesses about certain parts of the error vector $\mathbf{e}$, this method uses multiple representations of $\mathbf{s}$ to simplify the list construction process.

A recent study [19] introduces an exciting new variant called the k-Sparse Learning With Errors (k-sLWE) problem, which examines the computational challenges of using sparse coefficient matrices. Unlike traditional LWE, which uses dense matrices, k-sLWE employs random k-sparse matrices $\mathbf{A} \in \mathbb{Z}_M^{m \times n}$, where each row has exactly k non-zero elements, drawn uniformly from $\mathbb{Z}_M$. This fresh approach greatly improves efficiency in both storage and computation, as each k-sLWE sample consists of only $k+1$ elements, resulting in a computation time of $O(k \log M)$. In comparison, standard LWE samples require $n+1$ elements and take $O(n \log M)$ time. The parameter k is key to maintaining the difficulty of the k-sLWE problem. An interesting part of this research is the investigation of different settings for k, including constant, logarithmic, polylogarithmic, and sublinear values relative to the dimension n. The findings suggest that choosing k to be logarithmic or polylogarithmic strikes an ideal balance for cryptographic applications, offering both efficiency and strong security. While a constant k can provide theoretical insights under certain sample constraints, it does not achieve the necessary level of security when the sample size exceeds the linear dimension.

Our Contributions. In this work, we conduct a thorough analysis of the ternary variant of Sparse Learning With Errors (sLWE). Specifically, we propose a novel combinatorial attack that leverages a subsystem extraction technique to recover the secret vector efficiently. Our investigation explores the performance of this attack under various sparsity levels and modulus settings, revealing potential vulnerabilities in the proposed parameters. We show that, for practical moduli such as 2^{64}, the secret can be recovered within a feasible timeframe, casting doubt on the claimed security equivalence to LWE with a dimension of 1024. Additionally, for smaller moduli such as 2^{32} and 2^{16}, we observe a significant reduction in the security margin, further emphasizing the need for careful parameter selection.

Organization of our paper. The paper is organized as follows. In Section-II, we briefly introduce the Sparse Learning With Errors (sLWE) problem. Section-III introduces the ternary variant of sLWE. Section-IV describes our method for extracting a

Good Subsystem. In Section-V, we estimate the expected number of variables in a system of equations. Section-VI presents our efficient algorithm for solving an instance of ternary sLWE. Section-VII provides the results of our analysis on ternary sLWE, while Section-VIII details the experimental verification of the algorithm. Finally, we conclude the paper in Section-IX.

II. Mathematical Preliminaries and Notation

II-A Notation

Let $\mathbb{N} = \{1, 2, ...\}$ be the natural numbers and $\mathbb{Z}$ be the set of integers. Define $[n] = \{1, 2, \cdots, n\} \subset \mathbb{N}$. For a positive integer M, $\mathbb{Z}_M$ denotes the ring of integers modulo M. We will assume that $\mathbb{Z}_M = \{-\lfloor \frac{M-1}{2} \rfloor, -\lfloor \frac{M-1}{2} \rfloor + 1, \cdots, -1, 0, 1, \cdots, \lfloor \frac{M}{2} \rfloor - 1, \lfloor \frac{M}{2} \rfloor\}$. We use $\mathbb{Z}_M^n$ to represent the n-dimensional vector space over $\mathbb{Z}_M$. Capital bold letters (e.g., $\mathbf{A}$) are used to denote matrices, while small bold letters (e.g., $\mathbf{v}$) represent vectors. Additionally, we denote probability by $\mathbb{P}$, variance by $\mathbb{V}$, and covariance by $\mathbb{COV}$.

II-B Sparse Learning With Errors

In this study, we investigate a variant of the Learning With Errors (LWE) problem introduced by Jain et al. in CRYPTO 2024 [19]. We begin by formally defining the variant known as Sparse LWE.

Definition II.1. Let $n, m, M \in \mathbb{Z}$, and let k be a constant or a polylogarithmic function of n. We consider a random matrix $\mathbf{A} \in \mathbb{Z}_M^{m \times n}$, where each row contains exactly k non-zero elements uniformly sampled from $\mathbb{Z}_M$. Additionally, let $\mathbf{b} \in \mathbb{Z}_M^m$. The Sparse Learning With Errors ($\text{sLWE}_{n,k,m,M,\sigma}$) problem is defined as the task of finding a secret vector $\mathbf{s} \in \mathbb{Z}_M^n$ that satisfies the equation

$$\mathbf{As} = \mathbf{b} + \mathbf{e},$$

where $\mathbf{e} \in \mathbb{Z}_M^m$ is an error vector drawn from a discrete Gaussian distribution with mean 0 and noise parameter σ.

Definition II.2. For a modulus M and a vector $\mathbf{v} \in \mathbb{Z}_M^n$, the locality set $\text{LocSet}(\mathbf{v})$ is defined as the indices $S \subset [n]$ where $\mathbf{v}[i] \neq 0$ for all $i \in S$.

We demonstrate that if we have an oracle capable of solving the sparse LWE problem for dimension n and sparsity k with m samples, we can use it to solve the standard LWE problem for dimension k with the same m samples, provided the modulus p is prime.

Given an input pair $(\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e})$, we will transform it into a new pair $(\mathbf{A}', \mathbf{b}' = \mathbf{A}'\mathbf{s}' + \mathbf{e})$ that retains the same error properties. If $\mathbf{b}$ were sampled randomly instead of generated from the LWE distribution, this transformation would maintain that randomness. The resulting distribution of $(\mathbf{A}', \mathbf{b}')$ will match that of average-case sparse LWE. Importantly, this method does not alter the error vector $\mathbf{e}$, allowing it to also apply in scenarios like learning parity with noise.

Conversion Procedure. The conversion process works as follows: we start with a dense matrix $\mathbf{A}$ of size $m \times k$ and aim to convert each column into a corresponding sample in a new sparse matrix $\mathbf{A}'$ of size $m \times n$, where each row has exactly k non-zero entries. The vector $\mathbf{b}$ will remain unchanged for now, allowing us to express it as $\mathbf{b} = \mathbf{Az}' + \mathbf{e}$. At this point, the secret $\mathbf{z}$ is not uniformly random. To address this, we can add a random term $\mathbf{Av}'$ to $\mathbf{b}$ using a randomly selected vector $\mathbf{v}$. This ensures that the new secret $\mathbf{v} + \mathbf{z}$ is random and independent of $\mathbf{A}'$.

Next, we need to establish a relationship between $\mathbf{A}$ and $\mathbf{A}'$. Specifically, we want to ensure that there exists a matrix $\mathbf{C}$ of size $n \times k$ such that the equation $\mathbf{A}'\mathbf{C} = \mathbf{A}$ holds. This allows us to rewrite $\mathbf{b}$ as $\mathbf{b} = \mathbf{A}'(\mathbf{Cs}) + \mathbf{e}$.

To construct $\mathbf{A}'$, we first fix the matrix $\mathbf{C}$. We then randomly sample locality patterns $S_1, \ldots, S_m$ from $\binom{n}{k}$. For each i, we select the non-zero coordinates according to S_i so that $\mathbf{A}'[i]\mathbf{C} = \mathbf{A}[i]$. This is feasible because we can view this as a system of linear equations over $\mathbb{Z}_p$, where k variables correspond to the non-zero entries of $\mathbf{A}'[i]$ and k equations relate to the columns of $\mathbf{A}[i]$. We choose $\mathbf{C}$ to be a Vandermonde matrix, ensuring that any k rows are linearly independent, which guarantees that our system is solvable.

Randomness of $\mathbf{A}'$. To confirm that $\mathbf{A}'$ is random, we note that each row $\mathbf{A}'[i]$ can be described by its locality pattern and the random non-zero entries corresponding to this pattern. Since the locality patterns are independently chosen at random, the columns of $\mathbf{A}'$ will be uniformly distributed and independent of one another.

We state the following theorem:

Theorem: [19] If there exists a probabilistic polynomial-time (PPT) algorithm that can solve average-case $\text{sLWE}_{n,k,m,p,\sigma}$ instances in time T with success probability ξ, where the modulus p is a prime and the number of samples $m < p$, then we can solve average-case $\text{LWE}_{k,m,p,\sigma}$ instances in time $T + O(n^3 m \log p)$ with the same success probability ξ.

III. Ternary Variant of sLWE

In [19], the authors posed an open problem regarding the security of small secret sparse LWE. In this work, we introduce and analyze a novel variant of sparse LWE (sLWE) that incorporates ternary secrets and errors. This variant, which we call ternary sLWE, is formally defined as follows:

Definition III.1. Consider dimension n, sparsity parameter k, a sample complexity m, modulus M. Let $\mathbf{A} \in \mathbb{Z}_M^{m\times n}$ be a k-sparse random matrix whose every row has exactly k number of non-zero elements and $\mathbf{b} \in \mathbb{Z}_M^m$. A ternary sLWE problem asks to find a ternary secret vector $\mathbf{s} \in \{-1, 0, 1\}^n$ that holds the identity $\mathbf{A}\mathbf{s} = \mathbf{b} + \mathbf{e}$, where $\mathbf{e} \in \{-1, 0, 1\}^m$ is an arbitrary ternary vector.

We now proceed to analyze an instance of the ternary k-sLWE problem, where $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_M^{m\times n} \times \mathbb{Z}_M^m$, the dimension is n, and the number of samples is m. In this context, each equation derived from the samples involves precisely k variables, and the size of the search space for solution to any individual equation is 3^{k+1}, as the variables involved in the equation are ternary and the error in the equation is also ternary. Given the random nature of $\mathbf{A}$ and $\mathbf{b}$, a randomly chosen element from the search space satisfies a given equation with probability $\frac{1}{M}$. Consequently, the expected number of solutions to a single equation is $\frac{3^{k+1}}{M}$.

If k is sufficiently small, each equation is expected to have approximately one solution. In this regime, the equations are independent enough to be solved separately, and the solution to the ternary sLWE problem can be constructed coordinate by coordinate. We formalize this notion by defining a threshold for the sparsity parameter k:

Definition III.2. For an instance of ternary $\text{sLWE}_{n,k,m,M}$, we define the **threshold limit** $\delta \in \mathbb{N}$ as the largest integer such that $3^{\delta+1} \leq M$.

In instances where the sparsity parameter k is less than or equal to this threshold limit δ, each equation is expected to have a unique solution. In this case, using Algorithm-2, the solution for each equation can be recovered efficiently.

However, when k exceeds the threshold limit δ, solving the entire system of equations directly becomes computationally infeasible due to the exponential growth of the search space. To address this, we propose decomposing the system into smaller subsystems, where each subsystem can be solved independently.

Consider a subsystem with μ equations and v variables. The search space for the solution of such a subsystem is $3^{v+\mu}$, as there are v ternary variables and μ ternary errors to account for. The probability that a randomly chosen element from this search space satisfies all μ equations is $\frac{1}{M^\mu}$. Hence, the expected number of solutions for the subsystem is $\frac{3^{v+\mu}}{M^\mu}$.

If we can extract subsystems where $\frac{3^{v+\mu}}{M^\mu} \approx 1$, the solutions to these subsystems can be combined to recover the full secret vector $\mathbf{s}$. This decomposition strategy leverages the sparsity and structure of the problem, allowing us to tackle larger instances of ternary sLWE.

IV. Extracting an Effective Subsystem for Secret Recovery

Our initial approach is to identify a subsystem of sLWE$_{n,k,m,M}$ and solve it. We will continue this process until we determine all the unknown coordinates of the secret vector $\mathbf{s}$. However, if each subsystem yields exponentially many solutions, pinpointing the exact value of the secret vector will become quite challenging. To address this, we will introduce the following definition.

Definition IV.1. A subsystem of ternary sLWE$_{n,k,m,M}$ with v number of variables and μ number of equations is called a **Good Subsystem** of the instance if $3^{v+\mu} \leq M^{\mu}$.

Clearly, a Good Subsystem for a ternary sLWE$_{n,k,m,M}$ is expected to have exactly one solution. Thus, our goal will be to identify these Good Subsystems and subsequently solve them. But again, we have a challenge to find a Good Subsystem. So first we will reduce our problem to the following problem.

Problem IV.2. Let $A_0, A_1, \ldots, A_{\mu-1}$ be random subsets of $[n]$ such that $|A_i| = k$ for $i = 0, 1, \ldots, \mu-1$. We aim to analyze the probability of selecting these μ random subsets such that the cardinality of the union $A_0 \cup A_1 \cup \cdots \cup A_{\mu-1}$ does not exceed v.

To address this problem, we will aim to establish the following theorem.

Theorem IV.3. *Let $A_0, A_1, \ldots, A_{\mu-1}$ be random subsets of the set $[n]$ such that $|A_i| = k_i$ for each $i = 0, 1, \ldots, \mu-1$. Define the random variable X as the size of the union $A_0 \cup A_1 \cup \cdots \cup A_{\mu-1}$. Then, the expected value of X is given by*

$$\mathbb{E}(X) = n - \sum_{i=1}^{n} \prod_{j=0}^{\mu-1} \left(1 - \frac{k_j}{n}\right),$$

and the variance of X can be expressed as:

$$\mathbb{V}(X) = n \prod_{j=0}^{\mu-1} \left(1 - \frac{k_j}{n}\right) - n^2 \prod_{j=0}^{\mu-1} \left(1 - \frac{k_j}{n}\right)^2 + n(n-1) \prod_{j=0}^{\mu-1} \left(1 - \frac{2k_j}{n} + \frac{k_j(k_j-1)}{n(n-1)}\right).$$

Proof. Let $A = A_0 \cup A_1 \cup \cdots \cup A_{\mu-1}$. To analyze the union, we introduce indicator random variables X_{ij} defined as follows:

$$X_{ij} = \begin{cases} 1 & \text{if } i \in A_j, \\ 0 & \text{otherwise} \end{cases}$$

for each $i = 1, 2, \ldots, n$ and $j = 0, 1, \ldots, \mu-1$. It follows that for fixed i, the variables X_{ij} are independent, and we can compute their expected values as $\mathbb{E}(X_{ij}) = \frac{k_j}{n}$. Additionally, we define another indicator variable X_i for each i:

$$X_i = \begin{cases} 1 & \text{if } i \in A, \\ 0 & \text{otherwise} \end{cases}$$

Also, we have the following relations:

$$X_i = 1 - \prod_{j=0}^{\mu-1} (1 - X_{ij}) \text{ and } X = \sum_{i=1}^{n} X_i.$$

To find $\mathbb{E}(X)$, we calculate:

$$\mathbb{E}(X) = \sum_{i=1}^{n} \mathbb{E}(X_i) = n - \sum_{i=1}^{n} \mathbb{E}\left(\prod_{j=0}^{\mu-1} (1 - X_{ij})\right).$$

Applying the independence of the X_{ij} for a fixed i yields:

$$\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})\right)=\prod_{j=0}^{\mu-1}\mathbb{E}(1-X_{ij})=\prod_{j=0}^{\mu-1}\left(1-\frac{k_j}{n}\right).$$

Thus, we have the expected value of X:

$$\mathbb{E}(X)=n-\sum_{i=1}^{n}\prod_{j=0}^{\mu-1}\left(1-\frac{k_j}{n}\right).$$

Next, we will compute the variance $\mathbb{V}(X)$:

$$\begin{aligned}
\mathbb{V}(X)&=\mathbb{V}\left(\sum_{i=1}^{n}X_i\right)\\
&=\mathbb{V}\left(n-\sum_{i=1}^{n}\prod_{j=0}^{\mu-1}(1-X_{ij})\right)\\
&=\mathbb{V}\left(\sum_{i=1}^{n}\prod_{j=0}^{\mu-1}(1-X_{ij})\right)\\
&=\sum_{i=1}^{n}\mathbb{V}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})\right)+2\sum_{i=1}^{n}\sum_{r=i+1}^{n}\mathbb{COV}\left(\prod_{j=0}^{\mu-1}(1-X_{ij}),\prod_{j=0}^{\mu-1}(1-X_{rj})\right)\\
&=\sum_{i=1}^{n}\left[\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})^2\right)-\left\{\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})\right)\right\}^2\right]\\
&\quad+2\sum_{i=1}^{n}\sum_{r=i+1}^{n}\left[\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})(1-X_{rj})\right)-\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})\right)\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{rj})\right)\right].
\end{aligned}$$

From the independence of X_{ij} for a fixed i, we have

$$\begin{aligned}
\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})^2\right)-\left\{\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})\right)\right\}^2&=\prod_{j=0}^{\mu-1}\mathbb{E}\left((1-X_{ij})^2\right)-\prod_{j=0}^{\mu-1}\left\{\mathbb{E}\left(1-X_{ij}\right)\right\}^2\\
&=\prod_{j=0}^{\mu-1}\mathbb{E}\left(1-X_{ij}\right)-\prod_{j=0}^{\mu-1}\left\{\mathbb{E}\left(1-X_{ij}\right)\right\}^2\\
&=\prod_{j=0}^{\mu-1}\left(1-\frac{k_j}{n}\right)-\prod_{j=0}^{\mu-1}\left(1-\frac{k_j}{n}\right)^2
\end{aligned}$$

Similarly, we can get

$$\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})\right)\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{rj})\right)=\prod_{j=0}^{\mu-1}\mathbb{E}\left(1-X_{ij}\right)\prod_{j=0}^{\mu-1}\mathbb{E}\left(1-X_{rj}\right)=\prod_{j=0}^{\mu-1}\left(1-\frac{k_j}{n}\right)^2$$

Also,

$$\mathbb{E}\left(\prod_{j=0}^{\mu-1}(1-X_{ij})(1-X_{rj})\right)=\prod_{j=0}^{\mu-1}\mathbb{E}\left((1-X_{ij})(1-X_{rj})\right)$$

$$= \prod_{j=0}^{\mu-1} \mathbb{E}\left(1 - X_{ij} - X_{rj} + X_{ij}X_{rj}\right)$$

$$= \prod_{j=0}^{\mu-1} \left(1 - \mathbb{E}(X_{ij}) - \mathbb{E}(X_{rj}) + \mathbb{E}(X_{ij}X_{rj})\right)$$

$$= \prod_{j=0}^{\mu-1} \left(1 - \frac{2k_j}{n} + \frac{k_j(k_j-1)}{n(n-1)}\right)$$

After substituting all these values into our variance expression, we obtain:

$$\mathbb{V}(X) = n\prod_{j=0}^{\mu-1}\left(1-\frac{k_j}{n}\right) - n^2\prod_{j=0}^{\mu-1}\left(1-\frac{k_j}{n}\right)^2 + n(n-1)\prod_{j=0}^{\mu-1}\left(1-\frac{2k_j}{n}+\frac{k_j(k_j-1)}{n(n-1)}\right)$$

This completes the proof of the theorem, establishing both the expected value and variance of the random variable X. $\square$

From the above theorem, we have the following corollary:

Corollary IV.4. *Let $A_0, A_1, \ldots, A_{\mu-1}$ be random subsets of the set $[n]$ such that $|A_i| = k$ for each $i = 0, 1, \ldots, \mu-1$. Define the random variable X as the size of the union $A_0 \cup A_1 \cup \cdots \cup A_{\mu-1}$. Then, the expected value of X is given by*

$$\mathbb{E}(X) = n\left[1 - \left(1-\frac{k}{n}\right)^{\mu}\right],$$

and the variance of X can be expressed as:

$$\mathbb{V}(X) = n\left(1-\frac{k}{n}\right)^{\mu} - n^2\left(1-\frac{k}{n}\right)^{2\mu} + n(n-1)\left[1-\frac{2k}{n}+\frac{k(k-1)}{n(n-1)}\right]^{\mu}.$$

Thus, based on the above result, we can derive the mean and variance for the random variable X. We can then apply the Normal Distribution to determine the probability of selecting μ random subsets of S such that the size of their union is at most v, which is equivalent to $\mathbb{P}(X \leq v)$. Consequently, this probability can be expressed as $\mathbb{P}\left(\frac{v-\text{mean}}{\text{standard deviation}} \leq 1\right)$. This value can be obtained from the Z-table of the Standard Normal Distribution.

In this context, we must address one additional crucial point. Suppose we have identified a Good Subsystem containing at most v variables and μ equations. However, what happens if there are one or two equations within the subsystem that do not overlap with the others? In this scenario, while the large overlapping class may yield a unique solution, the isolated equations could generate exponentially many solutions, thereby exponentially increasing the overall number of solutions for the subsystem. Therefore, we will remove those isolated equations from the subsystem, and as a consequence, the expected number of solutions to this refined subsystem will be one.

To address this problem, let us introduce the following relation.

Definition IV.5. Let S be a collection of sets. We define a relation σ on S as follows:

For any $A_1, A_2 \in S$, we say $A_1 \; \sigma \; A_2$ if and only if there exists a sequence of sets $A_3, A_4, \ldots, A_n \in S$ such that the following conditions hold:

- $A_1 \cap A_3 \neq \varnothing$,
- $A_3 \cap A_4 \neq \varnothing$,
- $A_4 \cap A_5 \neq \varnothing$,
- $\vdots$
- $A_n \cap A_2 \neq \varnothing$.

Clearly, the relation σ on S is an equivalence relation, which allows us to derive a partition from this relation on S. As an initial intuition, our motive will be to extract the largest equivalence class. We present the following algorithm:

Algorithm 1: GetSubsystem(v, μ)

```
Input: sLWE$_{n,k,m,M}$, $v$, $\mu$
Output: A good refined subsystem of sLWE$_{n,k,m,M}$
1  repeat
2      repeat
3          Randomly select $\mu$ equations from the sample;
4          $S_{sub} \leftarrow$ Subsystem formed by the selected $\mu$ equations;
5      until;
6      The number of distinct variables in $S_{sub} \leq v$;
7      $S \leftarrow$ Collection of all index sets of variables associated with the $\mu$ equations in $S_{sub}$;
8      Compute partition on $S$ using equivalence relation $\sigma$;
9      $C_1 \leftarrow$ Largest equivalence class;
10     $S'_{sub} \leftarrow$ Collection of equations corresponding to $C_1$;
11     $\mu' = |S'_{sub}|$, $v' = \left|\bigcup_{A \in S'_{sub}} A\right|$;
12     if $3^{v'+\mu'} \leq M^{\mu'}$ then
13         return $S'_{sub}$;
14 until;
15 A Good Subsystem is found;
```

The time complexity of the above algorithm is entirely dependent on finding a subsystem with at most v variables and μ equations. This is because we will select v and μ such that the condition $3^{v+\mu} \leq M^{\mu}$ holds. This condition enhances the overlap between the equations, which suggests a high probability that the refined subsystem with v' variables and μ' equations will also satisfy the condition $3^{v'+\mu'} \leq M^{\mu'}$.

V. Estimating Number of Variables in γ Equations from the Subsystem

In the worst-case scenario, let's consider a subsystem with exactly v variables and μ equations. We aim to determine the expected number of variables present in γ equations from this subsystem. We can translate our problem into the following:

We are considering μ subsets $A_0, A_1, \ldots, A_{\mu-1}$ of the set $[n]$, where each subset consists of k elements. The task is to find the expected size of the union $\bigcup_{i=0}^{\gamma-1} A_i$, given that the total number of distinct elements in the union of all μ subsets, $\bigcup_{i=0}^{\mu-1} A_i$, is equal to v. This can be expressed mathematically as

$$\mathbb{E}\left(\left|\bigcup_{i=0}^{\gamma-1} A_i\right| \mid \left|\bigcup_{i=0}^{\mu-1} A_i\right| = v\right).$$

To compute this expectation, we need to evaluate the conditional probability

$$\mathbb{P}\left(\left|\bigcup_{i=0}^{\gamma-1} A_i\right| = \ell \mid \left|\bigcup_{i=0}^{\mu-1} A_i\right| = v\right)$$

for all possible sizes ℓ. According to Bayes' theorem, we can express this conditional probability as

$$\mathbb{P}\left(\left|\bigcup_{i=0}^{\gamma-1} A_i\right| = \ell \mid \left|\bigcup_{i=0}^{\mu-1} A_i\right| = v\right) = \frac{\mathbb{P}\left(\left|\bigcup_{i=0}^{\mu-1} A_i\right| = v \mid \left|\bigcup_{i=0}^{\gamma-1} A_i\right| = \ell\right) \cdot \mathbb{P}\left(|\bigcup_{i=0}^{\gamma-1} A_i| = \ell\right)}{\sum_j \mathbb{P}\left(\left|\bigcup_{i=0}^{\mu-1} A_i\right| = v \mid \left|\bigcup_{i=0}^{\gamma-1} A_i\right| = j\right) \cdot \mathbb{P}\left(|\bigcup_{i=0}^{\gamma-1} A_i| = j\right)}.$$

To find $\mathbb{P}\left(|\bigcup_{i=0}^{\gamma-1} A_i| = j\right)$, we need to consider how the elements are drawn from a total of n distinct items when selecting k items for each of the γ subsets. The probability of obtaining exactly j distinct outcomes can be modeled using the Hypergeometric Distribution, which is appropriate for situations where we are sampling without replacement. This distribution provides the necessary framework to calculate the likelihood of achieving various distinct counts after γ draws.

When $\gamma = 1$, the calculation becomes simpler because there is only one subset to consider. For this case,

$$\mathbb{P}\left(\left|\bigcup_{i=0}^{\gamma-1} A_i\right| = j\right) = \mathbb{P}(|A_0| = j) = \begin{cases} 1, & \text{if } j = k, \\ 0, & \text{otherwise.} \end{cases}$$

To efficiently track the probabilities of different union sizes after γ draws, we introduce an array denoted as Λ_γ, which has a size of n such that $\Lambda_\gamma[j] = \mathbb{P}\left(|\bigcup_{i=0}^{\gamma-1} A_i| = j\right)$ for each j from 1 to n. This array serves as a compact representation of the probabilities associated with each possible size of the union $\bigcup_{i=0}^{\gamma-1} A_i$.

We begin by initializing the array Λ_γ to represent the state after a single draw. Since the first subset contains exactly k elements, so we have k initial elements and thus we set $\Lambda_\gamma[k] = 1$, indicating that after one draw, the size of the union is exactly k. All other values in the array are initially set to zero.

The process then iterates for each of the additional $\gamma - 1$ draws. Each iteration updates Λ_γ to reflect the new possible sizes of the union after adding another subset. Let us assume that, after the i^{th} iteration, the size of $A_0 \cup A_1 \cup \cdots \cup A_i$ is j. Then we calculate the probability of reaching all possible new sizes j_1 after adding the next subset A_{i+1}. The probability of transitioning from a union of size j to size j_1 is computed using Hypergeometric Distribution and update the probabilities stored in Λ_γ. This process repeats for each subsequent draw, updating the array Λ_γ to reflect the distribution of possible union sizes after each subset is added. When all γ draws are completed, the final array Λ_γ contains the probabilities for all possible sizes of the union $\bigcup_{i=0}^{\gamma-1} A_i$.

Once the array Λ_γ has been computed for the first γ draws, the next step involves calculating the probability distribution for the size of $\bigcup_{i=0}^{\mu-1} A_i$, conditioned on the size of the union $\bigcup_{i=0}^{\gamma-1} A_i$. This is done by building a matrix $\mathcal{M}$, where each row represents the possible sizes of the union $\bigcup_{i=0}^{\gamma-1} A_i$, and each column represents the possible sizes of the union $\bigcup_{i=0}^{\mu-1} A_i$.

For each possible size j of $\bigcup_{i=0}^{\gamma-1} A_i$, we compute the distribution of union sizes $\bigcup_{i=0}^{\mu-1} A_i$ by invoking the rest $\mu - \gamma$ draws with an initial size of j. This creates a probability distribution for how the union can grow from size j to different sizes after the remaining $\mu - \gamma$ subsets are added. The result is stored in the j^{th} row of the matrix $\mathcal{M}$.

Once the matrix $\mathcal{M}$ is constructed, we focus on the v^{th} column, which contains the probabilities of obtaining $|\bigcup_{i=0}^{\mu-1} A_i| = v$, for each possible initial size of $\bigcup_{i=0}^{\gamma-1} A_i$, that is $\mathcal{M}[j][v] = \mathbb{P}\left(\left|\bigcup_{i=0}^{\mu-1} A_i\right| = v \mid \left|\bigcup_{i=0}^{\gamma-1} A_i\right| = j\right)$ for all $j = 1, 2, \cdots, n$. Utilizing Bayes' theorem, we can now compute $\mathbb{P}\left(\left|\bigcup_{i=0}^{\gamma-1} A_i\right| = \ell \mid \left|\bigcup_{i=0}^{\mu-1} A_i\right| = v\right)$ for all possible size ℓ and consequently we can compute the expected size of $\bigcup_{i=0}^{\gamma-1} A_i$ when it is given that $\left|\bigcup_{i=0}^{\mu-1} A_i\right| = v$.

VI. Our Algorithm To Solve Ternary sLWE

Case-1: Sparsity is less than or equal to the threshold limit

Consider an instance of the ternary sLWE problem with dimension n, sample complexity m, and a sparsity parameter k that is less than or equal to the threshold limit δ for this instance. In this case, we will select μ equations from the samples such that they involve all the variables of the secret vector $\mathbf{s}$. This can be done by randomly selecting μ equations and checking whether all variables are present.

Once such equations are identified, we solve them individually using Algorithm-2 to recover the secret vector $\mathbf{s}$. Algorithm-2 efficiently handles a single equation of the form

$$\sum_{i=0}^{k-1} a_i x_i = b + e,$$

where the unknowns x_i are ternary values in $\{-1, 0, 1\}$. This algorithm employs a Meet-in-the-Middle strategy by splitting the equation into two parts and searching for solutions in smaller spaces. Let $\ell = \lceil \frac{k}{2} \rceil$. We first focus on the left half of the equation, which involves the first ℓ terms:

$$T_{\text{left}} = \left\{ \left(y, \sum_{i=0}^{\ell-1} a_i y_i \right) : y \in \{-1, 0, 1\}^{\ell} \right\}$$

and for the right half of the equation, which involves the remaining $k - \ell$ terms, we compute:

$$T_{\text{right}} = \left\{ \left(y, b - \sum_{i=\ell}^{k-1} a_i y_{i-\ell} \right) : y \in \{-1, 0, 1\}^{k-\ell} \right\}.$$

Both T_{left} and T_{right} are sorted based on the second element of each pair, which is the sum computed for that half of the equation. The goal is to match pairs of partial solutions from T_{left} and T_{right} whose sums differ by a small value, within the range allowed by the noise e, that is

$$z_1 - z_2 \in \{-1, 0, 1\},$$

where z_1 is the sum from T_{left} and z_2 is the sum from T_{right}.

Once matching pairs are found, the corresponding vectors y_1 from T_{left} and y_2 from T_{right} are concatenated to form a complete solution vector $(y_1 || y_2)$. This vector is a valid solution for the original equation and stored in the table T.

Algorithm 2: Solve

```
Input: An equation $\sum_{i=0}^{k-1} a_i x_i = b + e$
Output: A table consisting of solutions for the given equation
1 $T \leftarrow \varnothing$;
2 $\ell \leftarrow \lceil \frac{k}{2} \rceil$ // Initialize empty table and half-length
3 $T_{\text{left}} \leftarrow \{(y, \sum_{i=0}^{\ell-1} a_i y_i) \mid y \in \{-1, 0, 1\}^{\ell}\}$ // Left partial sums
4 $T_{\text{right}} \leftarrow \{(y, b - \sum_{i=\ell}^{k-1} a_i y_{i-\ell}) \mid y \in \{-1, 0, 1\}^{k-\ell}\}$ // Right partial sums
5 Sort $T_{\text{left}}$ and $T_{\text{right}}$ by their last coordinate;
6 foreach $(y_1, z_1) \in T_{left}$ and $(y_2, z_2) \in T_{right}$ do
7     if $z_1 - z_2 \in \{-1, 0, 1\}$ then
8         $T \leftarrow (y_1 \parallel y_2)$ // Concatenate solutions
9 return $T$;
```

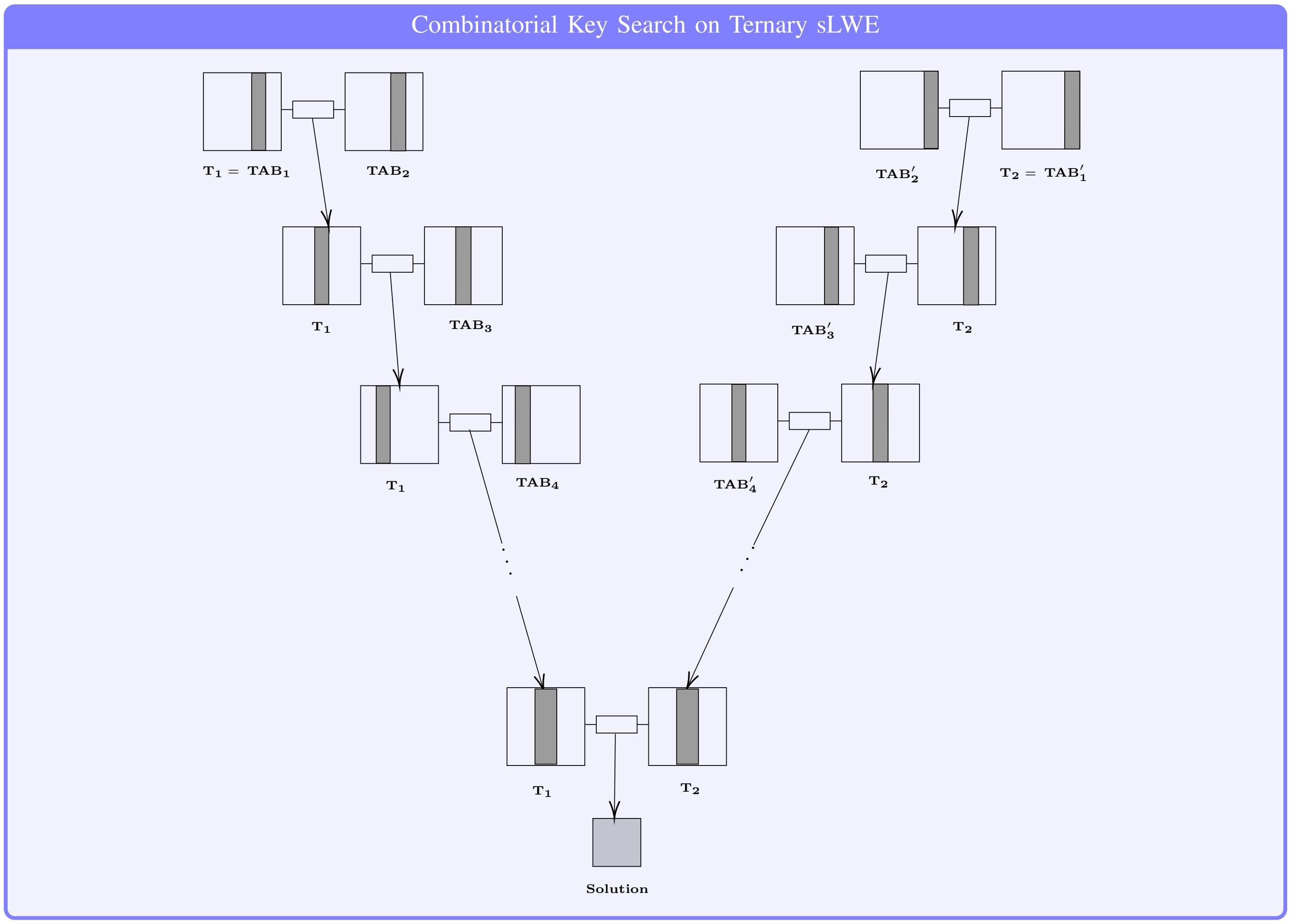
Case-2: Sparsity is greater than threshold limit

In this case, first we select optimal parameters v and μ and then we begin by obtaining a Good Subsystem that consists of at most v variables and μ equations. This subsystem forms the core of our combinatorial approach to solving the ternary k-sLWE problem.

To efficiently manage the solution space of the selected subsystem, we organize the solutions of the μ equations into two primary tables. These tables facilitate a structured merging process that progressively narrows down the possible solutions. To construct the first table T_1, we focus on the first $\lceil \mu/2 \rceil$ equations. For each equation, say the i^{th} equation $(i \in [\lceil \mu/2 \rceil])$ we construct a table TAB_i that contains all its solutions using Algorithm-2.

Next we will merge all these tables using Algorithm-3. The **MergeTable** algorithm combines solutions from $\lceil \mu/2 \rceil$ equations into a single table by identifying and merging matching solutions based on their shared variables.

The algorithm starts by taking the first table TAB_1 as the initial table T_1, and the variables involved in first equation are



stored in a variable set called ‘var’. For every subsequent equation (starting from the second one), the following steps are performed:

First, the common variables between the current set of variables (‘var’) and the variables of the current equation are identified. These common variables are important because solutions from different tables can only be merged if they agree on the values of these variables. Next, both the current table T_1 and the solution table TAB_i for the new equation are sorted according to the common variables. Sorting ensures that we can efficiently compare solutions by aligning them based on these shared variables. Then, the algorithm checks each pair of elements X from T_1 and Y from TAB_i. If the solutions for the common variables match, the two solutions are merged, and the combined solution is added to a new temporary table, ‘Tab’. Once all pairs of solutions have been checked, the variable set ‘var’ is updated to include the variables of the current equation, and the temporary table ‘Tab’ replaces the current table T_1. The algorithm returns the final table T_1, which contains solutions consistent across all $\lceil \mu/2 \rceil$ equations, and the corresponding set of variables ‘var’.

A similar procedure is followed for the second set of equations, which constitutes the last $\lfloor \mu/2 \rfloor$ equations, to construct table T_2. After obtaining both tables, T_1 and T_2, we compare and merge their solutions based on the common variables between them.

At this stage, in Algorithm-4 we perform another merging step, where the elements of T_1 and T_2 are sorted according to the values of their common variables. For each pair of elements from T_1 and T_2, if they share the same values for these variables, the solutions are merged into a final table T. If multiple solutions remain in T, further intersections with previously stored solutions (in the set $STORE$) are performed. If the resulting table T contains exactly one solution after the intersection, the corresponding values of the secret vector $\mathbf{s}$ are extracted. In the case where T contains multiple solutions, these are added to the set $STORE$ for future intersections. The subsystem equations are then removed from the sample, and the process repeats until the number of unrecovered variables is less than or equal to the threshold limit δ.

Finally, we iteratively solve for the remaining unrecovered variables by substituting the known values into the remaining equations and solving each equation independently. This process continues until the entire secret vector $\mathbf{s}$ is recovered.

Algorithm 3: MergeTable

```
Input: Collection of equations ($\mathcal{C}$ )
Output: A table of solutions corresponding to equations in ($\mathcal{C}$ )
// Extract variables and solve each equation
1  foreach equation in ($\mathcal{C}$ ) do
2      $\text{var}_i \leftarrow$ variables in the equation;
3      $TAB_i \leftarrow$ Solve(equation);
// Initialize the solution table
4  $T \leftarrow TAB_1$, $\text{var} \leftarrow \text{var}_1$;
// Merge solutions from all equations
5  for $i \geq 2$ do
6      $\text{Tab} \leftarrow \varnothing$;
7      $\text{CommonVar} \leftarrow \text{var} \cap \text{var}_i$;
8      Sort $T$ and $TAB_i$ with respect to the coordinates of CommonVar;
9      foreach $X \in T$ and $Y \in TAB_i$ do
10         if $X[CommonVar] = Y[CommonVar])$ then
11             $\text{Tab} \leftarrow \text{Merge}(X, Y)$;
       // Update the set of variables and solution table
12     $\text{var} \leftarrow \text{var} \cup \text{var}_i$;
13     $T \leftarrow \text{Tab}$;
14 return $T, var$;
```

VI-A *Correctness and Complexity Analysis*

Case-1: Sparsity is less than or equal to the threshold limit

The memory complexity of Algorithm-2 is $\mathcal{O}(\max\{|T_{\text{left}}|, |T_{\text{right}}|\})$, which evaluates to $\mathcal{O}\left(3^{\lceil \frac{k}{2} \rceil}\right)$. The time complexity of Algorithm-2 is determined by the sorting of the tables T_{left} and T_{right}. Therefore, the time complexity of Algorithm-2 is $\mathcal{O}\left(\frac{k}{2} \cdot 3^{\lceil \frac{k}{2} \rceil}\right)$.

Since there are μ equations, Algorithm-2 must be repeated μ times. Thus, the overall time complexity for the entire process becomes $\mathcal{O}\left(\frac{\mu k}{2} \cdot 3^{\lceil \frac{k}{2} \rceil}\right)$. Also, the probability of involving all variables in μ equations can be approximated using the Normal Distribution, with the mean and variance as calculated in Section-IV. Let this probability be denoted as $\tilde{p}$. Therefore, solving the instance requires finding μ equations that involve all variables and solving those equations, resulting in an overall time complexity of $\mathcal{O}\left(\max\{\frac{\mu k}{2} \cdot 3^{\lceil \frac{k}{2} \rceil}, \frac{1}{\tilde{p}}\}\right)$.

Furthermore, to achieve optimal time complexity, we select an optimal value of μ by imposing the following condition:

$$\frac{\mu k}{2} \cdot 3^{\lceil \frac{k}{2} \rceil} \approx \frac{1}{\tilde{p}}.$$

Case-2: Sparsity is greater than the threshold limit

The overall time complexity of the procedure is primarily determined by two factors: the search for a subsystem containing at most v variables and μ equations, and the efficiency of the **MergeTable** algorithm.

We have previously discussed in Section-IV the probability of locating such a subsystem, let us denote this probability as $\Pr_{\text{ssy}}$. Therefore, if we randomly select μ equations from a sample of size m, the expected number of attempts required to successfully find one subsystem with at most v variables and μ equations can be expressed as:

Algorithm 4: Algorithm for Solving Ternary k-sLWE

```
Input: $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_M^{m\times n} \times \mathbb{Z}_M^m$, sparsity $k \in \mathbb{N}$
Output: Ternary vector $\mathbf{s} \in \mathbb{Z}_M^n$ such that $\mathbf{e} = \mathbf{A}\mathbf{s} - \mathbf{b} \bmod M$ is ternary
1  Choose optimal parameters $v$ and $\mu$ for subsystem;
2  Compute threshold limit $\delta$ as the largest integer such that $\frac{3^{\delta+1}}{M} \leq 1$;
3  Initialize empty set: $STORE \leftarrow \varnothing$;
4  repeat
       ▷ Extract a subsystem and process it
5      GetSubsystem($v, \mu$);
6      Update values of recovered variables;
7      $T_1, \mathrm{var}_1 \leftarrow$ MergeTable(first $\lceil \mu/2 \rceil$ equations);
8      $T_2, \mathrm{var}_2 \leftarrow$ MergeTable(last $\lfloor \mu/2 \rfloor$ equations);
9      Initialize empty set: $T \leftarrow \varnothing$;
10     CommonVar $\leftarrow \mathrm{var}_1 \cap \mathrm{var}_2$;
       ▷ Sort tables by shared variables
11     Sort $T_1$ and $T_2$ according to CommonVar;
12     foreach $X \in T_1$ and $Y \in T_2$ do
13         if $X[CommonVar] = Y[CommonVar]$ then
14             $T \leftarrow$ Merge$(X, Y)$;
       ▷ Handle multiple solutions if necessary
15     if size of $T > 1$ then
16         Perform intersection with elements in $STORE$ sharing variables with $T$;
17     if size of $T = 1$ after intersection then
18         Collect values of the secret vector coordinates $\mathbf{s}$;
19     else
20         Insert elements into $STORE$;
21     Remove the equations of the subsystem from the sample;
22 until number of unrecovered variables $\leq \delta$;
   ▷ Solve any remaining unrecovered variables
23 repeat
24     Find an equation EQN involving unrecovered variables;
25     Substitute values of recovered variables in EQN;
26     Solve(EQN);
27     Remove EQN from sample;
28 until all variables are recovered;
29 return secret vector $\mathbf{s}$;
```

$$\mathcal{T}_{\text{find}} = \mathcal{O}\left(\frac{1}{\text{Pr}_{\text{ssy}}}\right).$$

Additionally, it is very crucial to analyze the constraints to find a subsystem:

- **Uniqueness of Solutions**: For a subsystem consisting of at most v variables and μ equations, the total number of variables including the error terms is $v + \mu$. This indicates that the size of the search space for ternary solutions and ternary errors is $3^{v+\mu}$. We can examine the probability that a randomly selected solution from this search space satisfies all μ equations simultaneously. In the ring $\mathbb{Z}_M$, each arbitrary element from the search space will satisfy all μ equations with a probability of:

$$\frac{1}{M^\mu}.$$

As a result, the expected number of valid solutions for the subsystem can be calculated as follows:

$$\text{Expected Number of Solutions} = \frac{3^{v+\mu}}{M^{\mu}}.$$

To ensure that the subsystem has a unique solution, we need to satisfy the following condition:

$$3^{v+\mu} \leq M^{\mu}.$$

From this inequality, we can derive a formula for μ:

$$\mu \geq \frac{v \log(3)}{\log(M) - \log(3)}.$$

This derived relationship serves an essential purpose: it quantifies how the number of equations μ must scale with respect to the number of variables v in the subsystem.

- **Existence of the Subsystem**: We can choose μ equations from a sample of size m in $\binom{m}{\mu}$ different ways. This combinatorial expression represents the number of possible combinations of μ equations that can be selected from the total available m equations. Each selection offers a unique set of equations that may or may not lead to the desired subsystem.

  To find a subsystem containing at most v variables and μ equations, we need to repeat the selection process $\mathcal{T}_{\text{find}}$ times. Each iteration corresponds to a new attempt at identifying a valid subsystem based on the chosen equations. Given the nature of the problem, this repeated selection is essential for increasing our chances of success in locating a suitable subsystem that satisfies the constraints of having at most v variables and μ equations.

  To ensure that we can reliably find such a subsystem within the available sample, we impose the following constraint:

$$\mathcal{T}_{\text{find}} \leq \binom{m}{\mu}.$$

  This inequality indicates that the number of attempts $\mathcal{T}_{\text{find}}$ should not exceed the total number of ways to choose μ equations from m equations.

Now, let us analyze the time and memory complexity of the **MergeTable** algorithm. The **MergeTable** algorithm processes a collection of equations, consider first $\lceil \frac{\mu}{2} \rceil$ equations, to generate a table of solutions corresponding to those equations. Below, we break down the time and memory complexities involved in this process.

Memory Complexity: During i^{th} iteration of the loop, the algorithm constructs a table T^i that contains the solutions for the first i equations. Let us denote the size of this table as $\mathcal{T}_i$. The memory required for the algorithm is determined by the largest table created throughout the iterations. Thus, the overall memory complexity is expressed as:

$$\mathcal{O}(\max_i\{\mathcal{T}_i\}).$$

Time Complexity for Table Construction: After the initial tables TAB_is are constructed, the algorithm merges these tables based on the common variables between them. To facilitate this merging in the i^{th} iteration, the algorithm needs to sort the current table with size $\mathcal{T}_i$ and the initial table TAB_i with respect to their common variables. The sorting process in every iteration incurs a time complexity of:

$$\mathcal{O}(\log(\max_i\{\mathcal{T}_i\}) \cdot \max_i\{\mathcal{T}_i\}).$$

The nested loop that merges the tables involves checking for common variable values and merging pairs of solutions (X, Y) when their common variable values match. The time complexity for this merging process in the i^{th} iteration depends on $\max\{\mathcal{T}_i, \mathcal{T}_{i+1}\}$. Therefore, the total time complexity of the algorithm **MergeTable** is

$$\mathcal{O}(\log(\max_i\{\mathcal{T}_i\}) \cdot \max_i\{\mathcal{T}_i\}).$$

Now we will find the expected value $\mathcal{T}_i$ that is the expected size of the table that consists the solutions of first i equations. If those i equations involve ν number of variables then the expected value of $\mathcal{T}_i$ will be $\frac{3^{\nu+i}}{M^i}$. Therefore, it suffices to compute the expected value for ν, which can be derived using the results from Section-V.

Thus, the time complexity for one iteration that is finding a subsystem and solving that subsystem is $\mathcal{O}(\max\{\log(\max_i \mathcal{T}_i)\cdot \max_i \mathcal{T}_i, \mathcal{T}_{\text{find}}\})$, while the memory complexity is $\mathcal{O}(\max_i \mathcal{T}_i)$. To determine the optimal complexity, we introduce an additional constraint on v and μ, such that $\mathcal{T}_{\text{find}} \approx \log(\max_i \mathcal{T}_i) \cdot \max_i \mathcal{T}_i$.

Let us assume that we repeat the process r times, and let V_i represent the set of variables in the subsystem corresponding to the i^{th} iteration. The expected size of $\cup_{i=1}^{r} V_i$ is given by $n\,[1 - (1 - v/n)^r]$. We will terminate the procedure when the number of remaining variables falls below the threshold limit δ. Therefore, we need to find the minimum r such that $n - n\,[1 - (1 - v/n)^r] \leq \delta$, or equivalently, $n(1 - v/n)^r \leq \delta$.

Hence, the overall time complexity for solving a ternary sLWE instance is $\mathcal{O}(r \cdot \max\{\log(\max_i \mathcal{T}_i) \cdot \max_i \mathcal{T}_i, \mathcal{T}_{\text{find}}\})$ and the memory complexity is $\mathcal{O}(\max_i \mathcal{T}_i)$.

VII. Our Result

Table I
Recommended Dimensions for sLWE with Security Guarantees Equivalent to LWE with Dimension 1024

Sparsity Parameter k \ Number of Samples m	2^{13}	2^{17}	2^{21}
20	1218	1425	1656
30	1143	1265	1395
40	1110	1195	1285
50	1090	1158	1234

The authors claim that the recommended parameters with modulus $2^{64}, 2^{32}$ or 2^{16} provide security equivalent to LWE with a dimension of 1024. However, our analysis, presented in the Table-II, Table-III and Table-IV, reveals significant vulnerabilities in ternary sLWE using the parameters in Table-I.

When the modulus is set to 2^{64}, we found that the secret key can be recovered in a reasonable timeframe, pointing to a serious security flaw. Moreover, in many configurations with moduli of 2^{32} and 2^{16}, our results indicate a significant reduction in the security margin, although a number of instances would resist our analysis, if a 128-bit security level is targeted. Our results raise important concerns about the overall strength of the system under these settings, suggesting that the actual security may be considerably weaker than expected, which aligns with the developers' concerns.

VIII. Experimental verification

We present a proof-of-concept attack to validate the effectiveness of our technique. To ensure efficiency, we select small parameters for our sLWE instance. Specifically, we use a dimension of $n = 200$, a sparsity parameter $k = 14$, a sample complexity of $m = 2^8$, and a modulus $M = 2^{18}$. With these parameters, we construct an instance of ternary sLWE.

To tackle this instance, we define a subsystem with 60 variables and 6 equations. The theoretical expected number of variables in 6 equations is 70.6, with a variance of 7.9, which can be derived from the formulas in Section-IV. Using a

Table II
SUMMARY OF MEMORY AND TIME COMPLEXITY RELATIVE TO PARAMETERS (v, μ) FOR THE SUBSYSTEM WITH MODULUS 2^{64}

k	$m = 2^{13}$					$m = 2^{17}$					$m = 2^{21}$				
	n	v	μ	Time (bits)	Memory (bits)	n	v	μ	Time (bits)	Memory (bits)	n	v	μ	Time (bits)	Memory (bits)
20	1218	20	1	28	16	1425	20	1	28	16	1656	20	1	29	16
30	1143	30	1	35	24	1265	30	1	36	24	1395	30	1	36	24
40	1110	74	2	43	32	1195	74	2	43	32	1285	74	2	43	32
50	1090	232	6	49	40	1158	235	6	49	40	1234	236	6	49	40

Table III
SUMMARY OF MEMORY AND TIME COMPLEXITY RELATIVE TO PARAMETERS (v, μ) FOR THE SUBSYSTEM WITH MODULUS 2^{32}

k	$m = 2^{13}$					$m = 2^{17}$					$m = 2^{21}$				
	n	v	μ	Time (bits)	Memory (bits)	n	v	μ	Time (bits)	Memory (bits)	n	v	μ	Time (bits)	Memory (bits)
20	1218	38	2	28	16	1425	38	2	28	16	1656	38	2	28	16
30	1143	383	20	94	79	1265	422	22	101	86	1395	460	24	113	93
40	1110	556	29	201	190	1195	594	31	218	203	1285	633	33	236	215
50	1090	670	35	316	304	1158	710	37	334	322	1234	762	40	356	344

Table IV
SUMMARY OF MEMORY AND TIME COMPLEXITY RELATIVE TO PARAMETERS (v, μ) FOR THE SUBSYSTEM WITH MODULUS 2^{16}

k	$m = 2^{13}$					$m = 2^{17}$					$m = 2^{21}$				
	n	v	μ	Time (bits)	Memory (bits)	n	v	μ	Time (bits)	Memory (bits)	n	v	μ	Time (bits)	Memory (bits)
20	1218	636	70	240	229	1425	742	82	285	268	1656	864	95	321	309
30	1143	827	91	479	468	1265	909	100	535	513	1395	1007	111	581	567
40	1110	942	104	686	672	1195	1013	112	746	724	1285	1089	120	786	774
50	1090	1003	111	837	826	1158	1065	118	888	876	1234	1133	125	945	931

Normal Distribution with mean 70.6 and variance 7.9, we estimated the probability of finding a subsystem with at most 60 variables and 6 equations as approximately $\frac{1}{2^{13}}$.

From the 2^8 samples, we arbitrarily select 6 equations and count the total number of variables involved. We conduct this experiment $3,812$ times, ultimately identifying a subsystem consisting of 60 variables and 6 equations. Next, we divide the 6 equations into two groups: the first three and the last three. The first group contains 37 variables, while the second contains 36. These values align well with the theoretical expectation of 36.53 variables, as discussed in Section-V. We solve each equation in the first group individually, focusing on the relevant variables, and record the solutions in a table. The size of the table corresponding to one equation is 52, which aligns closely with the theoretical value of $\frac{3^{14+1}}{2^{18}} \approx 55$.

We then merge the tables for the first three equations into a single table (T_1), which has a size of 495. This also matches the theoretical estimate of $\frac{3^{37+3}}{2^{18\times3}} \approx 678$. We repeat this process for the last three equations, creating a second table (T_2) with a size of 235, which is consistent with the theoretical value of $\frac{3^{36+3}}{2^{18\times3}} \approx 225$.

Finally, we merge the tables T_1 and T_2 to obtain the solution for all 6 equations, resulting in exactly one solution for the 60 variables. The entire process was repeated 10 times, reducing the number of unrecovered variables to below the threshold limit of 10.

IX. Conclusion

In this work, we have conducted the first third-party analysis of the ternary Sparse Learning With Errors (sLWE) problem, emerging from the recent advancements presented at Crypto 2024. By leveraging a novel combination of subsystem extraction and Meet-in-the-Middle strategies, we have effectively mounted a combinatorial attack on ternary sLWE instances. Our theoretical and experimental analyses support the developers' claims that sLWE, especially in its ternary variant, exhibits vulnerabilities when used with small secrets. In particular, when the modulus is set to 2^{64}, we were able to recover the secret key within a practical timeframe, demonstrating a clear security weakness. For configurations using smaller moduli such as 2^{32} and 2^{16}, our analysis revealed a further reduction in the security margin, reinforcing the developers' concerns regarding the robustness of ternary sLWE under these parameter choices.

These results confirm that sLWE, particularly in the ternary case, is susceptible to attacks when small secrets are employed, and emphasize the importance of choosing secure parameter sets. Although we focused on ternary secrets and ternary errors, our methodology can be extended to other scenarios involving small max-norm secrets. Our work provides additional evidence of the need for cryptographic strengthening in such configurations and serves as a foundation for future efforts to enhance the security of this problem.

X. References

[1] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *STOC*, 2005, p. 84–93.

[2] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal, "NTRU prime: Reducing attack surface at low cost," in *SAC 2017*, ser. LNCS, C. Adams and J. Camenisch, Eds. Heidelberg: Springer, 2017, vol. 10719, p. 235–260.

[3] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber: a cca-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, p. 353–367.

[4] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *41st ACM STOC*, M. Mitzenmacher, Ed. ACM Press, 2009, p. 169–178.

[5] V. Lyubashevsky, "Lattice signatures without trapdoors," in *EUROCRYPT 2012*, ser. LNCS, D. Pointcheval and T. Johansson, Eds. Heidelberg: Springer, 2012, vol. 7237, p. 738–755.

[6] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem: extended abstract," in *41st ACM STOC*, M. Mitzenmacher, Ed. ACM Press, 2009, p. 333–342.

[7] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *37th ACM STOC*, H. N. Gabow and R. Fagin, Eds. ACM Press, 2005, p. 84–93.

[8] D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa, "Efficient public key encryption based on ideal lattices," in *ASIACRYPT 2009*, ser. LNCS, M. Matsui, Ed. Heidelberg: Springer, 2009, vol. 5912, p. 617–635.

[9] C. Chen, O. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, J. M. Schanck, T. Saito, P. Schwabe, W. Whyte, K. Xagawa, T. Yamakawa, and Z. Zhang, "PQC round-3 candidate: NTRU," Tech. Rep., 2020, accessed: 2022-05-20.

[10] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *Annual Cryptology Conference*. Springer, 2013, p. 40–56.

[11] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann, "Practical lattice-based cryptography: A signature scheme for embedded systems," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, p. 530–547.

[12] H. Baan, S. Bhattacharya, S. Fluhrer, O. Garcia-Morchon, T. Laarhoven, R. Rietman, M.-J. O. Saarinen, L. Tolhuizen, and Z. Zhang, "Round5: Compact and fast post-quantum public-key encryption," in *International Conference on Post-Quantum Cryptography*. Springer, 2019, p. 83–102.

[13] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, "Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, p. 587–617.

[14] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, "High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, p. 618–647.

[15] Y. Lee, J.-W. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and H. Kang, "High-precision bootstrapping for approximate homomorphic encryption by error variance minimization," in *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part I*. Springer, 2022, p. 551–580.

[16] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory*, J. P. Buhler, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, p. 267–288.

[17] A. May, "How to meet ternary LWE keys," in *Advances in Cryptology – CRYPTO 2021*, T. Malkin and C. Peikert, Eds. Cham: Springer International Publishing, 2021, p. 701–731.

[18] N. Howgrave-Graham, "A hybrid lattice-reduction and meet-in-the-middle attack against NTRU," in *Advances in Cryptology - CRYPTO 2007*, A. Menezes, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, p. 150–169.

[19] A. Jain, H. Lin, and S. Saha, "A systematic study of sparse LWE," in *Advances in Cryptology – CRYPTO 2024*, ser. Lecture Notes in Computer Science, L. Reyzin and D. Stebila, Eds. Cham: Springer, 2024, vol. 14922, p. to appear.

Abul Kalam received the M.Sc degree in Mathematics from Visva Bharati University, India, in 2022. He is currently a Ph.D student, holding the post of Junior Research Fellow at Indian Institute of Technology Madras under the supervision of Prof. Santanu Sarkar. His research interests include cryptology and security.

Santanu Sarkar received the PhD degree in mathematics from the Indian Statistical Institute, Kolkata, India, in 2011. He is currently professor at IIT Madras, India. He was a guest researcher at the National Institute of Standards and Technology (NIST), Gaithersburg, Maryland. His research interests include cryptology and number theory.

Willi Meier received the Diploma degree in mathematics and the doctoral degree in mathematics from the Swiss Federal Institute of Technology Zurich (ETHZ), Zürich, Switzerland, in 1972 and 1975, respectively. He has been a Guest Researcher with Oxford University and Heidelberg University; and a Research Assistant with the University of Siegen, Germany. Since 1985, he has been a Professor of mathematics and computer science with the University of Applied Sciences and Arts Northwestern Switzerland, Windisch, Switzerland. His current interests include analysis and design of cryptographic primitives like stream ciphers and hash functions.