Tashfeen, Ahmad
Dr. Qi Cheng
29 March 2023

General Examination

In Scriptum

1. Introduction

Cryptography thrives on easy problems that can be disguised as the hard ones. In other words, if a hard problem can be easily solved with a special information, it's likely found somewhere in a cryptographic system. *Hard* in such problems is usually defined as hardness due to the long standing status of a problem and not provably hard. Take for example the factoring problem contingent wherein is the security of the famous Rivest–Shamir–Adleman (RSA) encryption. While suspected, there is no proof of existence, or more importantly, a proof of non-existence of a polynomial time factoring algorithm [1], i. e., factoring being in $\mathcal{P}$. This puts factoring in at most the non-deterministic polynomial ($\mathcal{NP}$) complexity class of problems however *not*[1] in the $\mathcal{NP}$-complete[2]. Whether a cryptographic system can be as hard to break as an $\mathcal{NP}$-complete problem is an open problem itself.

Since a solution to any problem in the $\mathcal{NP}$-complete class solves every other problem in the same class as well, these $\mathcal{NP}$-complete problems are generally considered harder than the exclusively $\mathcal{NP}$ problems. It then makes sense to seek a cryptographic system that relies on the hardness of a $\mathcal{NP}$-complete problem. The first attempt to develop such a system was made by Merkle and Hellman in the 1970s [3] [4]. The particular $\mathcal{NP}$-complete problem they based their cryptosystem on is the *knapsack* or the subset sum problem. In this report, we will discuss the knapsack problem by considering the obvious and the surprising sources of its solutions.

2. Background

In this section we will introduce the mathematical definitions and notations in a way that is suitable to the formulation of different solutions to the knapsack problem.

2.1. **$\mathcal{NP}$-Imposter.** Some problems appear to be $\mathcal{NP}$-complete without any special information. But, the very fact that some special information exists because someone has it, nullifies the $\mathcal{NP}$-completeness. This is because–with the special information that exists, the problem can be solved easily. Even though the theoretical $\mathcal{NP}$-completeness is ruled out due to the existence of special information, i. e., a private key, someone missing this special information is forced to treat such an imposter problem just like any other $\mathcal{NP}$-complete problem without a major breakthrough. We will refer to these lookalike $\mathcal{NP}$-complete problems as the $\mathcal{NP}$-imposter problems.

2.2. **Linear Algebra.** Among the surprising sources of solutions to the knapsack problem are lattices. Succinctly, a lattice is a discrete subgroup of $\mathbb{R}^n$. Before we delve further into the operations and definitions of lattices, we start by defining a simple vector projection and orthogonality.

In figure 1, we have two linearly independent vectors a and b, both in $\mathbb{R}^2$. The projection of a onto b is given as the green vector a_1. We define the a_1 projection vector of a onto b as,

$$a_1 = \text{proj}_b(a) = \left(\frac{a \cdot b}{b \cdot b}\right) b$$

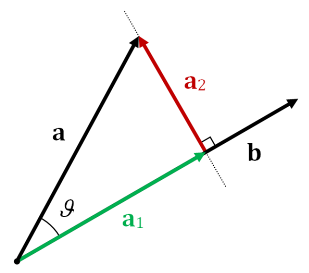

Figure 1. Projection of a onto b, i. e., $a_1 = \text{proj}_b(a)$.

[1]The scale of theoretical complexity from the easiest to the hardest is arranged as $\mathcal{P} \overset{?}{\subsetneq} \mathcal{NP} \subset \mathcal{NP}\text{-complete} \subset \mathcal{NP}\text{-hard} \subset$ ☠.

Note that $\left(\frac{a \cdot b}{b \cdot b}\right)$ is a scaler since it is a ratio of two dot (inner) products. Further note that while a and b form a basis of $\mathbb{R}^2$ on the account of their linear independence, they do not form the desired orthogonal basis. On the other hand, a_2 and b not only form the basis of $\mathbb{R}^2$ but they are also orthogonal. Therefore, we say that a_2, b form an orthogonal basis of $\mathbb{R}^2$. Since we already have b, observe that,

$$\begin{aligned}
a_1 + a_2 &= a && \text{Head-to-Tail Rule.} \\
a_2 &= a - a_1 && \text{Subtracting } a_1 \text{ on both sides.} \\
&= a - \operatorname{proj}_b(a) && \text{By figure 1.} \\
&= a - \left(\frac{a \cdot b}{b \cdot b}\right) b && \text{By the definition of } \operatorname{proj}_b(a) \text{ above.} \\
&= a - \mu b && \text{Let } \mu = \left(\frac{a \cdot b}{b \cdot b}\right).
\end{aligned}$$

The technique of projections to achieve an orthogonal basis is generalised as the *Gram-Schmidt* process [3]. Let $v_1, v_2, v_2, \ldots, v_n$ be a set of linearly independent vectors forming a basis of $\mathbb{R}^n$, then we define the orthogonal basis as $u_1, u_2, u_2, \ldots, u_n$.

$$\begin{aligned}
u_1 &= v_1 \\
u_2 &= v_2 - \operatorname{proj}_{u_1}(v_2) \\
u_3 &= v_3 - \operatorname{proj}_{u_1}(v_3) - \operatorname{proj}_{u_2}(v_3) \\
u_4 &= v_4 - \operatorname{proj}_{u_1}(v_4) - \operatorname{proj}_{u_2}(v_4) - \operatorname{proj}_{u_3}(v_4) \\
&\ddots \\
u_n &= v_n - \operatorname{proj}_{u_1}(v_n) - \operatorname{proj}_{u_2}(v_n) - \operatorname{proj}_{u_3}(v_n) - \cdots - \operatorname{proj}_{u_{n-1}}(v_n)
\end{aligned}$$

The above can be explicitly written to demonstrate the *Gram-Schmidt* coefficients $\left(\frac{v_i \cdot u_j}{u_j \cdot u_j}\right)$.

$$\begin{aligned}
u_1 &= v_1 \\
u_2 &= v_2 - \left(\frac{v_2 \cdot u_1}{u_1 \cdot u_1}\right) u_1 \\
u_3 &= v_3 - \left(\frac{v_3 \cdot u_1}{u_1 \cdot u_1}\right) u_1 - \left(\frac{v_3 \cdot u_2}{u_2 \cdot u_2}\right) u_2 \\
u_4 &= v_4 - \left(\frac{v_4 \cdot u_1}{u_1 \cdot u_1}\right) u_1 - \left(\frac{v_4 \cdot u_2}{u_2 \cdot u_2}\right) u_2 - \left(\frac{v_4 \cdot u_3}{u_3 \cdot u_3}\right) u_3 \\
&\ddots \\
u_n &= v_n - \left(\frac{v_n \cdot u_1}{u_1 \cdot u_1}\right) u_1 - \left(\frac{v_n \cdot u_2}{u_2 \cdot u_2}\right) u_2 - \left(\frac{v_n \cdot u_3}{u_3 \cdot u_3}\right) u_3 - \cdots - \left(\frac{v_n \cdot u_{n-1}}{u_{n-1} \cdot u_{n-1}}\right) u_{n-1}
\end{aligned}$$

Let $\mu_{i,j} = \left(\frac{v_i \cdot u_j}{u_j \cdot u_j}\right)$ for $i > j$ then,

$$\begin{aligned}
u_1 &= v_1 \\
u_2 &= v_2 - \mu_{2,1} u_1 \\
u_3 &= v_3 - \mu_{3,1} u_1 - \mu_{3,2} u_2 \\
u_4 &= v_4 - \mu_{4,1} u_1 - \mu_{4,2} u_2 - \mu_{4,3} u_3 \\
&\ddots \\
u_n &= v_n - \mu_{n,1} u_1 - \mu_{n,2} u_2 - \mu_{n,3} u_3 - \cdots - \mu_{n,n-1} u_{n-1}
\end{aligned}$$

Let $\boldsymbol{\mu}$ be an n by n lower-triangular *Gram-Schmidt* coefficients matrix defined as,

$$\boldsymbol{\mu} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ \mu_{2,1} & 0 & 0 & \cdots & 0 & 0 \\ \mu_{3,1} & \mu_{3,2} & 0 & \cdots & 0 & 0 \\ \mu_{4,1} & \mu_{4,2} & \mu_{4,3} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mu_{n,1} & \mu_{n,2} & \mu_{n,3} & \cdots & \mu_{n,n-1} & 0 \end{bmatrix}$$

Code listing 1 gives a Python implementation of the vector projections and the *Gram-Schmidt* orthogonalisation as we defined it.

```python
import numpy as np

def proj(u, v):  # projecting v onto u
    mu = (v @ u.T) / (u @ u.T)
    return mu * u, mu

def gram_schmidt(B):
    U, Mu = np.array(B, dtype=B.dtype), np.zeros(shape=B.shape, dtype=B.dtype)
    for i in range(1, B.shape[1]):
        for j in range(i):
            projection, Mu[i][j] = proj(U[:, j], B[:, i])
            U[:, i] -= projection
    return U, Mu
```

Listing 1. Vector projection $\text{proj}_u(v)$ and *Gram-Schmidt* orthogonalisation.

2.3. **Knapsack Problem.** If your house was on fire and you could only carry out a kilogram of weight with you, you might want to add the most valuable items up to a kilogram. How could that be done? You can take the power set of your household items, then sieve it to only contain the sets that weigh a kilogram and find the one with the maximum value. But of course, your house is on fire, so you hurry up!

We solve an instance of the knapsack problem every-time we make change. The most common coin denominations in the US are the twenty-five cents, ten cents, five cents and one cent, i. e., quarters, dimes, nickels and pence.

$$M' = \{1, 5, 10, 25\}$$

Let's say we had the above set and wanted to make 31 cents. You may come out with the subset $\{1, 5, 25\}$. We can encode this as a vector,

$$M' = [1, 5, 10, 25], x = [1, 1, 0, 1]$$

Note that,

$$M' \cdot x = 31$$

We define the *knapsack* problem as, for any $M \in \mathbb{N}^n$, $S \in N$ find $x \in \{0, 1\}^n$ such that,

$$M \cdot x = S$$

2.4. **Lattices.** Lattices are to euclidean spaces what modulo classes are to the real number line. Consider a basis matrix $\mathbf{B}$ with d linearly independent column vectors $\{1 \le i \le d : v_i \in \mathbb{R}^n\}$ such that v_i is the i^{th} column of $\mathbf{B}$. If $d = n$ we can write the entire $\mathbb{R}^n$ as,

$$\mathbb{R}^n = \{x \in \mathbb{R}^d : \mathbf{B}x\}$$

As a sanity check, we inspect the dimensions, $\dim(\mathbf{B}_{(n,d)}x_{(d,1)}) = (n, 1)$; makes sense. If we restrict $x \in \mathbb{Z}^d$ to the integers, i. e., $\mathbf{B}x$ to only the integral linear combinations and may allow $d \ne n$, we obtain a lattice,

$$\mathcal{L} = \{x \in \mathbb{Z}^d : \mathbf{B}x\}$$

The dimension of the lattice is $\dim(\mathcal{L}) = d$, i. e., the number of vectors in the basis matrix $\mathbf{B}$.

The following example bases illustrate a key difference in how bases behave differently in the euclidean space versus the lattice space.

$$\mathbf{B} = \begin{bmatrix} 47 & 95 \\ 215 & 460 \end{bmatrix}, \quad \mathbf{B'} = \begin{bmatrix} 0 & 50 \\ 50 & 0 \end{bmatrix}$$

Note that while both **B** and **B'** span the same space, i. e., $\mathbb{R}^2$ and **B'** is the trivial orthogonalisation of **B**. The two bases **B** and **B'** *do not span the same lattice*, as well as there is no trivial orthogonal version of **B** in the general case. Can you spot the trivial orthogonalisation of **B** in figure 2?

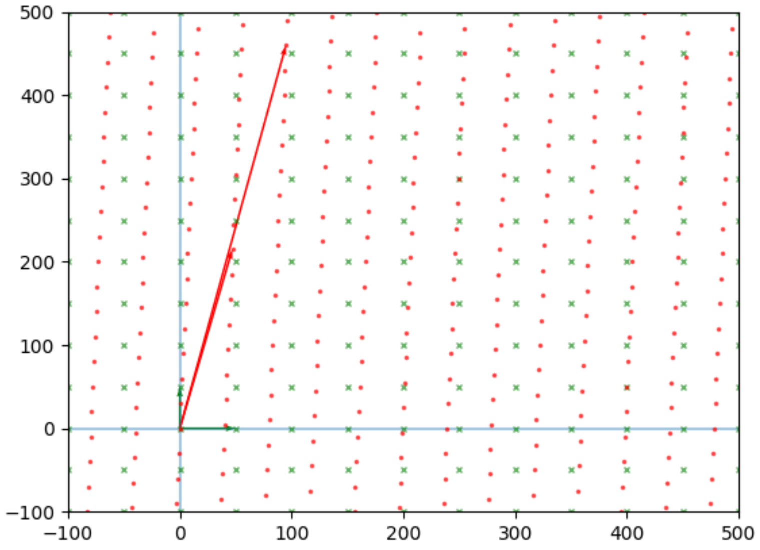

FIGURE 2. Lattice spanned by **B** and **B'**.

If we further restrict to **B** to $\mathbb{Z}^{n \times d}$, the resultant lattices are called the Lagarias-Odlyzko lattices [5]. We'll call these the *knapsack lattices.*

3. Merkle–Hellman Knapsack Cryptosystem

Revisiting the aforementioned coin denominations,

$$M' = \{1, 5, 10, 25\} = \{r'_1, r'_2, r'_3, r'_4\}$$

Note that,

$$r'_{i+1} \geq 2r'_i$$

Is that just a coincidence? No. The sets/sequences of the form $M' = \{1 \leq i \leq n : r'_i\}$ where $r'_{i+1} \geq 2r'_i$ are known as *super increasing* [3]. Knapsack problems for such sets can be solved in linear time by the algorithm given in listing 2. Furthermore, there is a bijection between the subsets of super increasing sequences and their sums.

```
x = [0 for i in M_]                 # Hoffstein Prop. 7.5 (pg. 379)
for i in range(len(M_)-1, -1, -1): # Loop i from n down to 1
    if S_ >= M_[i]:                 #     If S >= M'[i],
        x[i] = 1                    #         set x[i] = 1 and
        S_ = S_ - M_[i]             #         subtract M'[i] from S
    else:                           #     Else
        x[i] = 0                    #         set x[i] = 0
```

LISTING 2. Linear time algorithm to solve the knapsack problem for *super increasing* sets `M_` $= M'$.

If Alice can disguise a super increasing set M' as otherwise M then Bob can send her messages x (binary vectors) by sending $S = M \cdot x$ over a public channel. In such a way, Eve knows (M, S) and has to solve an $\mathcal{NP}$-imposter problem to obtain x whereas since Alice knows (M', S'), she can use the algorithm given in listing 2 to quickly obtain Bob's original message. This is chronologically shown in table 1.

Alice	Eve	Bob
Pick $M' = [r'_1, \dots, r'_n]$, such that $r'_1 > 2^n, r'_{i+1} \geq 2r'_i$. Pick A, B with $B > 2r'_n$ and $\gcd(A, B) = 1$.		
Let $r_i \equiv Ar'_i \mod B$ & $M = \{r'_i \in M' : r_i\} \longleftarrow$	M $\longrightarrow$	M
$S \longleftarrow$	S $\longleftarrow$	$S = Mx$
(M', S') is $\mathcal{O}(n)$. Let $S' \equiv A^{-1}S \mod B$. Solve $(M', S') \longrightarrow x$. We have $M'x = S'$.	(M, S) is $\mathcal{NP}$-imposter.	

Table 1. Merkle–Hellman knapsack cryptosystem [3].

Decryption works as follows,

$$\begin{aligned}
S' &\equiv A^{-1}S \mod B && \\
&\equiv A^{-1}Mx \mod B && \text{Bob's Encryption } S = Mx \\
&\equiv \sum_{i=1}^{n} A^{-1} r_i x_i \mod B && \text{Since } M = \{r'_i \in M' : r_i\} \\
&\equiv \sum_{i=1}^{n} A^{-1}(Ar'_i) x_i \mod B && \text{Since } r_i \equiv Ar'_i \\
&\equiv \sum_{i=1}^{n} r'_i x_i \mod B && \\
&\equiv M'x \mod B && \\
&= M'x && \text{Since } M'x \leq r'_1 + r'_2 + r'_3 +, \dots, r'_n < 2r'_n < B
\end{aligned}$$

Therefore, $M'x = S'$ if and only if $S = Mx$.

4. Cryptanalysis

Any given set M has $2^{|M|}$ number of subsets. Therefore, we can exhaust the subset space, checking their sums against S for the general knapsack problem as we defined it in section 2.3 in $\mathcal{O}(2^n)$ steps.

4.1. **Collision Algorithm.**

We prove a simple collision algorithm for $\mathcal{O}(2^{\frac{n}{2}})$.

Proof. Any general knapsack problem (M, S) can be solved in $\mathcal{O}(2^{\frac{n}{2}})$.

We start by splitting M into two halves,

$$\begin{aligned}
M_L &= \left\{1 \leq i < \left\lfloor \frac{n}{2} \right\rfloor + 1 : M_i \right\} && \text{The left half} \\
M_R &= \left\{ \left\lfloor \frac{n}{2} \right\rfloor + 1 \leq i \leq n : M_i \right\} && \text{The right half}
\end{aligned}$$

Note that if n is odd, M_L has fewer elements. Now we compute the subsets x_i and their corresponding sums. Let $b_j(i) = \left\lfloor \frac{i}{2^j} \right\rfloor \mod 2$. From the little end, $b_j(i)$ is the j^{th} bit in the binary representation of i.

$$L = \left\{ 0 \le i < 2^{\lfloor \frac{n}{2} \rfloor} : x_i = \{0 \le j \le \lfloor \lg i \rfloor : b_j(i)\}, \sum_{j=0}^{\lfloor \lg i \rfloor} x_{i,j} M_{L_j} \right\}$$

$$R = \left\{ 0 \le i < 2^{\lceil \frac{n}{2} \rceil} : x_i = \{0 \le j \le \lfloor \lg i \rfloor : b_j(i)\}, \sum_{j=0}^{\lfloor \lg i \rfloor} x_{i,j} M_{R_j} \right\}$$

Since $M_L \in \mathbb{N}^{\lfloor \frac{n}{2} \rfloor}$ and $M_R \in \mathbb{N}^{\lceil \frac{n}{2} \rceil}$ are both sets of positive integers, we may sort both L and R in $O(2^{n/2})$. Once they are both sorted, we may find a collision $L \star R = L_{\ell,1} + R_{r,1} = S$ like this,

```
$\ell \leftarrow \left\lfloor \frac{n}{2} \right\rfloor - 1$
$r \leftarrow 0$
while $(L_{\ell,1} + R_{r,1} \ne S)$   # while no collision
    if $(L_{\ell,1} + R_{r,1} < S)$ then $r \leftarrow r + 1$
    if $(L_{\ell,1} + R_{r,1} > S)$ then $\ell \leftarrow \ell - 1$
```

The above loop finds a collision in $O(n/2)$ then we may concatenate $L_{\ell,0}$ and $R_{r,0}$ for x in $Mx = S$. QUOD ERAT DEM■

We show an example where $M = \{2, 3, 5, 7, 11, 13\}$ and $S = 26$,

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} & 0 \\ & 2 \\ & 3 \\ & 5 \\ & 5 \\ & 7 \\ \ell \longrightarrow & 8 \\ \ell \longrightarrow & 10 \end{bmatrix} \star \begin{bmatrix} 0 & \longleftarrow r \\ 7 & \longleftarrow r \\ 11 & \longleftarrow r \\ 13 & \longleftarrow r \\ 18 & \longleftarrow r \\ 20 & \\ 24 & \\ 31 & \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 11 \\ 13 \end{bmatrix}$$

$$L_{\ell,1} + R_{r,1} \in \{10, 17, 21, 23, 28, 26\}$$

Therefore, we recovered $x = [0, 1, 1, 1, 1, 0]$ in less than $2^{6/2} = 8$ steps.

4.2. Lattice Reduction Algorithm.

Among the surprising sources of solutions to the knapsack problem are the lattice reduction algorithms. In section 2.4 we saw the following lattice basis,

$$\mathbf{B} = \begin{bmatrix} 47 & 95 \\ 215 & 460 \end{bmatrix}$$

If we try to orthogonalise the two almost parallel basis vectors via *Gram-Schmidt* as described in section 2.2 then we get,

$$\text{Gram-Schmidt}(\mathbf{B}) = \begin{bmatrix} 47 & -155875/48434 \\ 215 & 34075/48434 \end{bmatrix}$$

Unfortunately, as seen in figure 3, Gram-Schmidt($\mathbf{B}$) is not only non-integral, but neither does it span the same lattice (even if it spans the same space). A *lattice reduction algorithm* is the equivalent of *Gram-Schmidt* but for the lattice bases. It takes an integral basis and turns it into a new almost orthogonal basis that spans the same lattice.

As is usually the case with "firsts", Gauss [3] gave the first algorithm to reduce a two dimensional lattice. These are lattices with two basis vectors. A. K. Lenstra, H. W. Lenstra, L. Lovász published a more general lattice reduction algorithm (LLL) in 1982 [6] that reduces any general lattice in polynomial time of $O(n^2 \log n + n^2 \log \max(\mathbf{B}))$. Remember that n corresponds to the number of coordinates in any given lattice

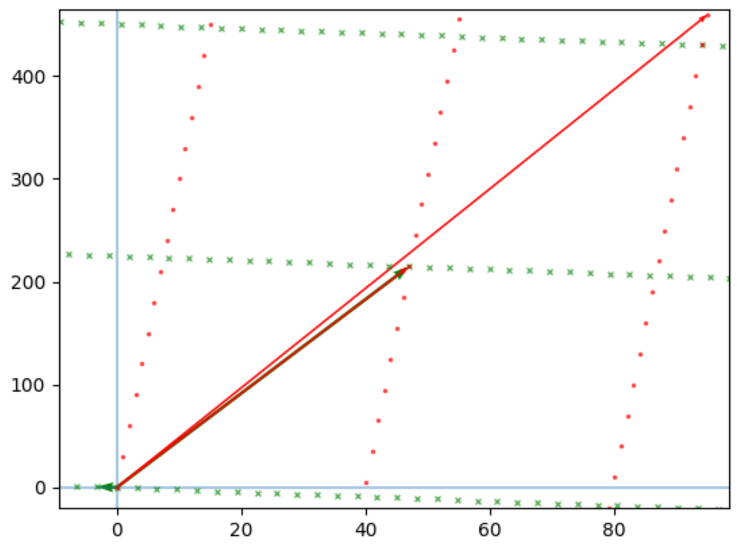

FIGURE 3. Lattice spanned by **B** and Gram-Schmidt(**B**).

vector which matches the cardinality of M. $\max \mathbf{B}$ is defined as the basis vector with the largest euclidean norm.

In *Remark 7.72* Hoffstein et al. [3] point out that "The problem of efficiently implementing the LLL algorithm presents many challenges." Therefore, the author decided to implement the LLL efficiently to ensure their comprehension of the algorithm and illustrate the lattice based attacks on the Merkle–Hellman knapsack cryptosystem in low dimensions. Note that the listing 3 uses functions previously given in listing 1.

```python
def lovasz_condition(G, Mu, k, delta):
    c = delta - Mu[k][k - 1]**2
    return G[:, k] @ G[:, k].T >= c * (G[:, k - 1] @ G[:, k - 1].T)

def lll(bad_basis, delta=0.75):
    B = np.array(bad_basis)
    G, Mu = gram_schmidt(B)  # G are the B*
    k, n = 1, B.shape[1] - 1
    while k <= n:
        for j in range(k - 1, -1, -1):
            if abs(Mu[k][j]) > 0.5:  # size condition not satisfied
                B[:, k] -= round(Mu[k][j]) * B[:, j]
                G, Mu = gram_schmidt(B)
        if lovasz_condition(G, Mu, k, delta):
            k = k + 1
        else:
            B[:, [k, k - 1]] = B[:, [k - 1, k]]  # swap
            G, Mu = gram_schmidt(B)
            k = max(k - 1, 1)
    return B
```

LISTING 3. Tashfeen's Python implementation of the general LLL lattice reduction algorithm.

It is here, where we use the specialised construction of the *Gram-Schmidt*, i. e., the *Gram-Schmidt* coefficients matrix,

$$\texttt{Mu} = \boldsymbol{\mu} \iff \texttt{Mu[k][j]} = \mu_{k,j} = \left(\frac{v_k \cdot u_j}{u_j \cdot u_j}\right) \text{ for } k > j.$$

Figure 4 shows that LLL(**B**) is not only integral but also spans the same lattice as **B**.

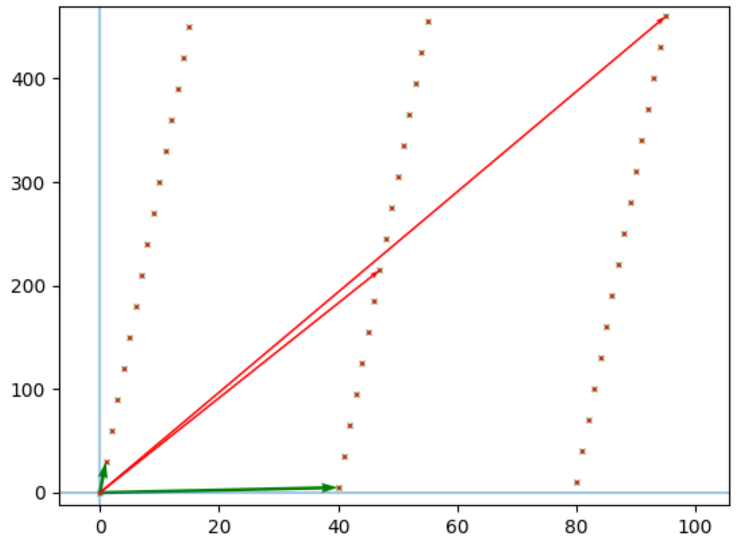

FIGURE 4. Lattice spanned by **B** and $\text{LLL}(\mathbf{B}) = \text{LLL}\left(\begin{bmatrix} 47 & 95 \\ 215 & 460 \end{bmatrix}\right) = \begin{bmatrix} 1 & 40 \\ 30 & 5 \end{bmatrix}$.

4.3. **Lattice Attack Setup.** Take any knapsack problem $M = \{r_1, r_2, r_3, \ldots, r_n\}$ with S and the relevant solution x such that $M \cdot x = S$. Recall that $r_i \in \Theta(2^{2n})$ due to requirements specified in table 1,

$$\Theta(2^{2n}) \ni 2^{2n} = \underbrace{(2 \cdot 2^{n-1} \cdot 2^n)}_{B} > \underbrace{(2^{n-1} \cdot 2^n)}_{r'_n} \geq \cdots \geq \underbrace{(2 \cdot 2 \cdot 2^n)}_{r'_3} \geq \underbrace{(2 \cdot 2^n)}_{r'_2} \geq r'_1 > 2^n$$

Now consider the following knapsack lattice basis in $\mathbb{N}^{d \times d}$ with $\dim(\boldsymbol{\kappa}) = d = n + 1$,

$$\boldsymbol{\kappa} = \begin{bmatrix} 2 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 2 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 2 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & 1 \\ r_1 & r_2 & r_3 & \cdots & r_n & S \end{bmatrix}$$

The lattice spanned by $\boldsymbol{\kappa}$ must have a vector that is the result of the following linear combination due to x,

$$t = \begin{bmatrix} 2 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 2 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 2 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & 1 \\ r_1 & r_2 & r_3 & \cdots & r_n & S \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \\ -1 \end{bmatrix} = \begin{bmatrix} 2x_1 - 1 \\ 2x_2 - 1 \\ 2x_3 - 1 \\ \vdots \\ 2x_n - 1 \\ M \cdot x - S \end{bmatrix} = \begin{bmatrix} 2x_1 - 1 \\ 2x_2 - 1 \\ 2x_3 - 1 \\ \vdots \\ 2x_n - 1 \\ 0 \end{bmatrix}$$

Since $x \in \{0, 1\}^n$ then $2x_i - 1 = \pm 1$. Therefore, $||t|| = \sqrt{n}$ which is at a stark contrast with the other vectors in the lattice spanned by $\boldsymbol{\kappa}$ due to the relative size of $r_i \in \Theta(2^{2n})$. Therefore, if we have a way of reducing the *bad* $\boldsymbol{\kappa}$ to an orthogonal and small, i. e., good $\boldsymbol{\kappa}'$ then t is very likely to be a part of the good basis. That way is the LLL algorithm and it's exponential time variations. In section 4.4, we demonstrate a real attack on the Merkle–Hellman cryptosystem.

4.4. **Lattice Attack Execution.** Tashfeen's friend Sam uses the Merkle–Hellman cryptosystem with the following public key,

$$\begin{aligned} M &= [r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, r_{17}, r_{18}, r_{19}, r_{20}, r_{21}, r_{22}, r_{23}, r_{24}, r_{25}] \\ &= [67108861, 134217725, 268435453, 536870909, 1073741821, 2147483645, \\ &\quad 4294967293, 8589934589, 17179869181, 34359738365, 68719476733, 137438953469, \\ &\quad 274877906941, 549755813885, 1099511627773, 2199023255549, 4398046511101, \\ &\quad 8796093022205, 17592186044413, 35184372088829, 70368744177661, \\ &\quad 140737488355325, 281474976710653, 562949953421309, 1125899906842621] \end{aligned}$$

And, the encoding given in table 2.

␣	A	B	C	D	E	···	T	U	V	W	X	Y	Z
0	1	2	3	4	5	···	20	21	22	23	24	25	26
00000	00001	00010	00011	00100	00101	···	10100	10101	10110	10111	11000	11001	11010

Table 2. Encoding Table.

Tashfeen sends Sam the name of his favourite Advisory Conference Report (ACR) committee member via,

$$S = 2002491457667039$$

Having read the *very* well-written *written* part of the Tashfeen's general exam, his advisor decides to figure out the plain-text x. He first constructs the bad basis $\boldsymbol{\kappa}$,

$$\boldsymbol{\kappa} = \begin{bmatrix}
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \\
r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 & r_8 & r_9 & r_{10} & r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} & r_{17} & r_{18} & r_{19} & r_{20} & r_{21} & r_{22} & r_{23} & r_{24} & r_{25} & S
\end{bmatrix}$$

Then, he uses the code implementing LLL in the listing 3 to obtain the reduced $\mathrm{LLL}(\boldsymbol{\kappa})$,

$$\left[\begin{array}{cccccccccccccc|c|ccccccccccc}
-4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1174258 \\
2 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & -1 & 0 & 0 & 0 & 0 & 1174260 \\
0 & 2 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1174260 \\
0 & 0 & 2 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1174258 \\
0 & 0 & 0 & 2 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & -3 & 0 & 0 & 0 & 0 & 1174262 \\
0 & 0 & 0 & 0 & 2 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1174264 \\
0 & 0 & 0 & 0 & 0 & 2 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1174266 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1174276 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 3 & 0 & 0 & 0 & 0 & 1174296 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1174328 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1174396 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1174534 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -4 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1174810 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -4 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1175364 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -1 & -4 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1176470 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & -4 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1178676 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 2 & -4 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1183098 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 2 & -4 & -3 & -3 & 0 & 0 & 0 & 0 & 1191934 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 2 & -1 & -1 & 0 & 0 & 0 & 0 & 1209608 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & -3 & 0 & 0 & 0 & 0 & 1244958 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & -4 & 0 & 0 & 0 & 1315654 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & 2 & -4 & 0 & 0 & 1457052 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 2 & -4 & 0 & 1739842 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 2 & -4 & 2305430 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 2 & 3436596 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 0 & 3 & 3 & 3 & 3 & 0 & 0 & 3 & 3 & 3 & 3 & 782839
\end{array}\right]$$

Sure enough, after some inspection he finds that the fourteenth column in $\mathrm{LLL}(\boldsymbol{\kappa})$ is the shortest vector t in the lattice spanned by $\boldsymbol{\kappa}$. He can further verify that,

$$||t|| = \sqrt{n} = \sqrt{25} = 5$$

He then lets,

$$x \equiv t - 1 \equiv [0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1] \mod 3$$

Breaking x per encoding table 2,

$$\begin{array}{ccccc}
00011 & 01000 & 00101 & 01110 & 00111 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
3 & 8 & 5 & 14 & 7 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{C} & \text{H} & \text{E} & \text{N} & \text{G}
\end{array}$$

5. Key Size

Merkle–Hellman knapsack cryptosystem is based on an $\mathcal{NP}$-complete problem but section 4.4 shows that the key size n matters. Otherwise, nitwits like Tashfeen can go around breaking post quantum cryptography with their shoddy implementation of some algorithm (listing 3) from the 80's [6]. Cleary, $n = 25$ is not large enough. How large should n be to be computationally secure? LLL approximates the shortest vector by finding another vector in the lattice $\mathcal{L}$ which is *no longer than* $\left(\frac{4}{3}\right)^{\frac{\dim(\mathcal{L})-1}{2}}$ times the length of the shortest vector [5]. We might use this theoretically provable upper-bound to judge an $n = 150$,

$$\left(\frac{4}{3}\right)^{\frac{151-1}{2}} < 2346417266$$

Which computes very high. Albeit, Gama et al. show a gap between theoretically provable and empirically predictable upper-bound [5].

$$(1.0219)^{\frac{151-1}{2}} < 5.1$$

This means that for a cryptosystem depending upon the hardness of approximating the shortest vector in a lattice to be secure, we need $\dim(\mathcal{L}) = n + 1 > 150$.

6. Lattice Problems

Under proper randomised assumptions, the *Shortest Vector Problem* (SVP) is not just $\mathcal{NP}$-complete but is also $\mathcal{NP}$-hard [3]. One of the contributing factors towards this hardness is the length of the shortest vector. Unlike that of knapsack lattices, in the general case we are not only unaware of the shortest vector but also how short it is, i. e., its length. Let $\lambda_i(\mathcal{L})$ denote the radius of the smallest zero-centred (hyper) ball containing i linearly independent vectors of $\mathcal{L}$ then we may use $\lambda_1(\mathcal{L})$ as notation for the length of the shortest (possibly non-unique) vector in $\mathcal{L}$. Additionally, we define the volume or the determinant of a lattice $\mathcal{L}$ as,

$$\det \mathcal{L} \le \prod_{i=1}^{d} ||b_i||$$

Where b_i are columns of a basis matrix $\mathbf{B}$. Note that the above inequality becomes equality when b_i are all orthogonal. According to the *Gaussian heuristic* for a randomly chosen lattice with a sufficiently large dimension d,

$$\lambda_1(\mathcal{L}) \approx \sqrt{\frac{d}{2\pi e}}(\det \mathcal{L})^{1/d}$$

Or, more generally due to the Hermite's constant γ_d,

$$\lambda_1(\mathcal{L}) \le \sqrt{\gamma_d}(\det \mathcal{L})^{1/d}$$

The exact values of the Hermite's constant γ_d for $d > 8$ are unknown. We state the three easier versions of the general SVP. Let v be a non zero vector,

Hermite-SVP: For $\alpha > 0$, find a vector $v \in \mathcal{L}$ such that $||v|| < \alpha \cdot (\det \mathcal{L})^{1/d}$.
Approx-SVP: For $\alpha > 0$, find a vector $v \in \mathcal{L}$ such that $||v|| < \alpha \cdot \lambda_1(\mathcal{L})$.
Unique-SVP: For $g > 1$, such that $\lambda_2(\mathcal{L})/\lambda_1(\mathcal{L}) \ge g$, find the unique shortest $v \in \mathcal{L}$.

Any algorithm that solves the Hermite-SVP with an approximation factor of α also solves the Approx-SVP with α^2 [7] [5]. We have seen one such algorithm, namely LLL which runs in polynomial time. There are exponential time variants of LLL known as BKZ and DEEP. While exponential, when these algorithms terminate, we can hope for a better approximation factor.

As hinted before, there is a gap between the theoretically proven and empirically predicted (by Gama et al [5]) upper-bounds on the approximation factors of the three of these algorithms. We summarise these gaps in table 3.

Hermite $\alpha^{1/d}$	LLL	BKZ	DEEP
Empirical	1.0219	1.0128	1.011
Theoretical	1.0754	1.0337	1.075

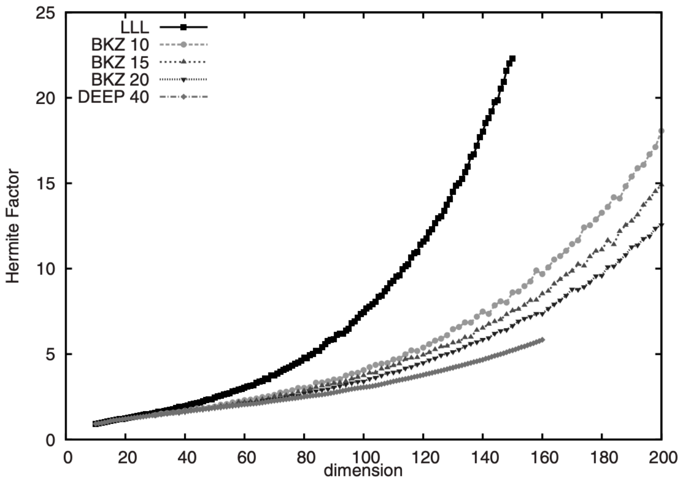

Table 3. Hermite factor gap for LLL, BKZ and DEEP where $1 \le d \le 200$ from by Gama et al [5].

We saw this gap playing out even in our usage of the LLL in section 4.4. We approximated the smallest vector in the knapsack lattice exactly, which is modelled more precisely via $1.0219^{2\times25} \approx 3$ than $1.0754^{2\times25} \approx 38$.

6.1. **Empirical Method.** With "empirically predicted" might come the question of the empirical method. Gama et al. use the "beautiful albeit mathematically sophisticated" probability distribution over the lattices defined by Goldstein et al. [8] in order to produce random lattices. They can then pick d linearly independent vectors to form a random basis. Note that they found the Unique-SVP to have a specially reduce-able case whenever $d \leq 70$, e. g., the knapsack lattices. In all other cases, they claimed the average case of lattice reduction via reduction algorithms to be equal to the worst case.

7. Conclusion

Unlike cryptosystems based on suspected $\mathcal{NP}$-complete problems, Merkle and Hellman based their scheme on a provable $\mathcal{NP}$-complete problem of the finding a subset that equals a desired sum. The knapsack problem therein is at at most as hard as the $\mathcal{NP}$-hard problem of finding the shortest vector in a lattice. Lattice reduction algorithms can be used to solve a given instance of a knapsack problem even in high dimensions $1 \leq d \leq 100$. This way of solution is en tandem with the gap between theoretically provable and empirically predicted d for which the SVP and its variants Hermite-SVP, Approx-SVP and Unique-SVP should be considered easy to solve.

References

[1] Steven George Krantz. *The proof is in the pudding: The changing nature of mathematical proof*, page 203. Springer, 2011.

[2] T. Gowers, J. Barrow-Green, and I. Leader. *The Princeton Companion to Mathematics*, page 583. Princeton University Press, 2010.

[3] Jeffrey Hoffstein, Jill Pipher, Joseph H Silverman, and Joseph H Silverman. *An introduction to mathematical cryptography*, volume 1, pages 377, 378, 387, 381, 437, 443, 395. Springer, 2008.

[4] Ralph C Merkle and Martin E Hellman. Hiding information and signatures in trapdoor knapsacks. In *Secure communications and asymmetric cryptosystems*, pages 197–215. Routledge, 2019.

[5] Nicolas Gama and Phong Q Nguyen. Predicting lattice reduction. In *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*, pages 31–51. Springer, 2008.

[6] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[7] László Lovász. An algorithmic theory of numbers, graphs and convexity, cbms-nsf reg. In *Conf. Ser. Appl. Math*, volume 50, page 91, 1986.

[8] Daniel Goldstein and Andrew Mayer. On the equidistribution of hecke points. *Forum Mathematicum*, 15:165–189, 2003.

University of Oklahoma, Computer Science