Lattice Basis Reduction and Knapsack Cryptosystems

Jonathan E. Maltz
Faculty Mentor: Dr. Kevin O'Bryant

June 9, 2010

Abstract

One of the earliest attempts at public key cryptography used what are now called knapsacks. Based on the subset sum problem, knapsack cryptosystems have consistently been shown vulnerable to attack by lattice basis reduction. This thesis introduces the Merkle–Hellman and Chor–Rivest knapsack cryptosystems, lattices, and the LLL lattice basis reduction algorithm.

1 Introduction

Public key cryptography (PKC) involves the receiving party of a secure communication keeping an unpublished *private key*, which will be used to easily decrypt a secure message sent to them by a sender, who encrypts their message using the receiver's published *public key*. From a security standpoint, this requires two properties to be true for a cryptosystem to be reliably secure: First, the private key must not be easily recovered from the public information, or an eavesdropper will be able to decrypt the message as though they are authorized; and second, the plaintext must be difficult to retrieve using only a ciphertext and the public key. The second aspect is particularly interesting because a ciphertext should additionally be easy to compute for the sender, or the cryptosystem will be too cumbersome and impractical for widespread use. To rephrase the challenge of PKC, a ciphertext must be easy for the sender to build and easy for the receiver to decrypt, while remaining difficult to disassemble by an eavesdropper who knows all of the ciphertext's possible building blocks.

One type of PKC cryptosystem is built upon the subset sum problem. These *knapsack cryptosystems* consist of a private set generated with some mathematical relationship amongst the elements in order to facilitate "easy" decryption by the authorized receiver. To impede decryption from an eavesdropper, operations are performed on the elements of the set to make them appear random. Because encryption is generally easy for the sender, who performs elementary operations to combine a subset of elements, an eavesdropper without the private decryption data has 2^n possible combinations to consider, where n is the (usually large) number of elements in the knapsack.

In this paper, we consider two such knapsack cryptosystems. First, the Merkle–Hellman cryptosystem, based on super-increasing sets and modular arithmetic, is introduced. Then we discuss the more secure Chor–Rivest cryptosystem, which is based on finite field arithmetic. Our goal is show the vulnerabilities of these systems by identifying attacks to recover a plaintext message from only the public information (the public knapsack and ciphertext).

The attack vector that we use is lattice basis reduction. After presenting background information on lattices, we discuss the Lenstra–Lenstra–Lovász (LLL) algorithm for lattice basis reduction. Using that algorithm, it is "almost always" easy to recover the plaintext from a lattice constructed of the public information in the Merkle–Hellman cryptosystem. Because of the in-

creased security of the Chor–Rivest cryptosystem, such an attack against that cryptosystem is not as straightforward. We end with a discussion of the lattice reduction-based attacks against the Chor–Rivest system.

2 Knapsack Schemes

Knapsack cryptographic problems involve an ordered set of n numbers and a message encoded as n bits, breaking up the message into several parts and padding as necessary. The ciphertext is generated by selecting elements of the set based on the message bits and combining them through a mathematical operation (usually addition). To be secure, the set must be considerably large, so that an eavesdropper attempting to crack the ciphertext by brute-force will have a tremendous amount of computational time resulting from the 2^n possible solutions. To be a realistic cryptographic system, however, the intended recipient must be able to decrypt the message in a relatively short amount of time.

To accommodate this requirement, knapsack cryptosystems include so-called "trapdoors" as a part of their private keys. A trapdoor, in this context, is usually a mathematical relationship amongst the elements of the knapsack which ensures that every ciphertext has a unique and easy-to-perform decryption. To provide security against an eavesdropper, this trapdoor must be concealed through a private randomization function performed on the knapsack prior to its publishing. The function usually includes algebraic operations which sufficiently alter the elements, causing their apparent relationship to be obscured. A knapsack cryptosystem which implements this type of authorized decryption method is called a "trapdoor knapsack."

Because the trapdoor facilitates the decryption of a ciphertext, and generally involves less than 2^n possible solutions, the security of the trapdoor (and its concealment) becomes paramount in determining the security of the specific knapsack cryptosystem implementation. Likewise, many attacks on knapsack cryptosystems revolve around weaknesses in their trapdoors. To gain a practical understanding of trapdoor knapsack cryptosystems, a description of two such systems follow.

2.1 Merkle–Hellman Knapsack Cryptosystem

The Merkle–Hellman knapsack cryptosystem [6] was one of the earliest knapsack cryptosystems. It relies on a super-increasing set to form a knapsack with a trapdoor, and modular arithmetic to conceal the trapdoor. We will first examine the initialization and encryption/decryption algorithms of this system, followed by a brief practical example of the system in use.

2.1.1 Key Generation

We choose a knapsack vector $\mathbf{a}' = \langle a'_1, a'_2, \ldots, a'_n \rangle$ generated in a way such that for all i,

$$a'_i > \sum_{j=1}^{i-1} a'_j, \tag{1}$$

and two large coprime numbers m and w, where

$$m > \sum a'_i. \tag{2}$$

Defining "mod" to be the least positive residue, we then transform $\mathbf{a}'$ into the trapdoor vector $\mathbf{a}$ by performing the following computation on each element

$$a_i = w \cdot a'_i \bmod m, \tag{3}$$

and publish the resulting "random" vector $\mathbf{a}$ as the public key. The remaining data (m, w, and $\mathbf{a}'$) is kept as the private key.

2.1.2 Encryption and Decryption

With a message encoded as a bit string $\mathbf{x}$, the encrypted message is computed:

$$S = \sum x_i a_i \tag{4}$$

To decrypt, we must calculate $w^{-1} \pmod m$, which exists by design, and then perform the following computation:

$$\begin{aligned} S' &= w^{-1} \cdot S \bmod m & (5)\\ &= w^{-1} \sum x_i a_i \bmod m & (6)\\ &= w^{-1} \sum x_i \cdot (w a'_i) \bmod m & (7)\\ &= \sum x_i \cdot a'_i \bmod m & (8) \end{aligned}$$

To find $\mathbf{x}$, we can use a greedy algorithm such that we set $x_n = 1$ if and only if $S' > a'_n$, and for $i = n-1, n-2, \ldots, 1$, we set $x_i = 1$ if and only if

$$S' - \sum_{j=i+1}^{n} x_j \cdot a'_j \geq a'_i. \tag{9}$$

2.1.3 Example

To clarify the nature of this system, a small example is given for a knapsack with five elements. We choose

$$\begin{aligned} \mathbf{a}' &= \langle 171, 196, 457, 1191, 2410 \rangle \\ m &= 8443 \\ w &= 2550 \text{ (so } w^{-1} = 3950), \end{aligned}$$

and keep that information private. We then calculate

$$\mathbf{a} = \langle 5457, 1663, 216, 6013, 7439 \rangle$$

and publish $\mathbf{a}$ as the public knapsack.

A user wishing to send $\mathbf{x} = \langle 0, 1, 0, 1, 1 \rangle$ will send

$$S = 1663 + 6013 + 7439 = 15115.$$

To obtain $\mathbf{x}$ from S, we compute

$$\begin{aligned} S' &= w^{-1} \cdot S \bmod m \\ &= 3950 \cdot 15115 \bmod 8443 \\ &= 3797. \end{aligned}$$

Because $S' > a'_5$ ($3797 > 2410$), we set $x_5 = 1$. Using (9), we find that $x_4 = 1$, $x_3 = 0$, $x_2 = 1$, and $x_1 = 0$, which is the initial message $\mathbf{x}$.

If the message is intercepted, the eavesdropper will only have the information made public. They will have to find the combination of elements in $\mathbf{a} = \langle 5457, 1663, 216, 6013, 7439 \rangle$ which, when added together, result in the sum $S = 15115$. Although that challenge is trivial in this small example, it could require up to 2^5 possible calculations by brute-force.

2.2 Chor–Rivest

Designed to improve the security of knapsack cryptosystems, the Chor–Rivest knapsack cryptosystem [2] utilizes finite field arithmetic during the construction phase. It also uses the work of Bose and Chowla [1] to restrict the *hamming weight* of a secret message to a size h (large messages can be split into a series of small messages if necessary). This restriction plays an important role in the cryptosystem's construction and trapdoor generation.

2.2.1 Key Generation

1. Pick a prime power p, and an integer $h \leq p$ such that discrete logarithms in $GF(p^h)$ can be efficiently computed. To resist an attack by brute-force, typical magnitudes for these values are $p \approx 200$ and $h \approx 25$.

2. Pick a random $t \in GF(p^h)$ that is of algebraic degree h over $GF(p)$. This is done by finding $f(t)$, a random irreducible monic polynomial of degree h in $GF(p)[t]$, and representing $GF(p^h)$ arithmetic by $GF(p)[t]/\langle f(t)\rangle$. This enables every element of $GF(p^h)$ to be written uniquely in the form $c_0 + c_1 t + c_2 t^2 + \cdots + c_{n-1} t^{n-1}$, $c_i \in GF(p)$.

3. Pick a multiplicative generator $g \in GF(p^h)$ of $GF(p^h)$ at "random" by picking a random $r \in GF(p^h)$ until one which satisfies $r^{(p^h-1)/s} \neq 1$ (for all prime factors s of $p^h - 1$) is found.

4. Compute $a_i = \log_g(t + \alpha_i)$ for all $\alpha_i \in GF(p)$. This forces g^{a_i} to be a polynomial of the the form $g^{a_i} = \alpha_i + 1t + 0t^2 + \cdots + 0t^{h-1}$, and since g is a generator of $GF(p^h)^\times$, we know that $0 < a_i < p^h$.

5. Scramble the a_i's by randomly choosing a permutation $\pi : \{0, 1, \ldots, p-1\} \to \{0, 1, \ldots, p-1\}$, and set $b_i = a_{\pi(i)}$.

6. Pick $0 \leq d \leq p^h - 2$ at random. Set $c_i = b_i + d$. This adds noise to the system.

The vector $\mathbf{c} = \langle c_0, c_1, \ldots, c_{p-1}\rangle$, with p and h are published as the public key. The values t, g, π^{-1}, and d are kept as the private key.

2.2.2 Encryption and Decryption

With a message encoded as a bit string $\mathbf{x}$ of length p and weight (the number of 1's) *exactly* h, the encrypted message is computed:

$$S = \sum_{i=0}^{p-1} x_i c_i \pmod{p^h - 1}. \tag{10}$$

To decrypt, the following computations are performed:

1. Let $r(t) = t^h \bmod f(t)$, a polynomial of degree $\leq h - 1$. This is computed once during the system generation.
2. Compute $S' = S - hd \pmod{p^h - 1}$.
3. Compute $q(t) = g^{S'} \bmod f(t)$, a polynomial of degree $h - 1$ in the formal variable t.
4. Add $t^h - r(t)$ to $q(t)$ to get $s(t) = t^h + q(t) - r(t)$, a polynomial of degree h in $GF(p)[t]$.
5. This results in

$$s(t) = (t + \alpha_{i_1}) \cdot (t + \alpha_{i_2}) \cdots (t + \alpha_{i_h}), \tag{11}$$

where $s(t)$ factors to *linear terms* over $GF(p)$. By successive substitutions, we find the h roots α_{i_j}'s using at most p substitutions. Applying π^{-1} recovers the coordinates of the bits in the original $\mathbf{x}$ that are set to 1.

2.2.3 Example

This example was generated using the Finite Fields Package in Mathematica.

Key Generation

1. We choose $p = 11$ and $h = 5$.
2. Using p and h, we can find $f(t) = 1 + 7t^2 + 4t^3 + 8t^5$, which gives us a $t \in GF(11^5)$ of degree h.

3. Now, we choose a multiplicative generator of $GF(11^5)$, $g = 10 + 7t^2 + 5t^3 + 8t^4 + 2t^5$.

4. The computation $a_i = \log_g(t + \alpha_i)$ for all $\alpha_i \in GF(11)$ yields $\mathbf{a} = \langle 95254, 101623, 37272, 53340, 79700, 113016, 23201, 141170, 74278, 136906, 2586\rangle$.

5. We choose a random permutation $\pi = \{6, 7, 9, 10, 0, 5, 1, 4, 3, 8, 2\}$, and calculate its inverse, $\pi^{-1} = \{4, 6, 10, 8, 7, 5, 0, 1, 9, 2, 3\}$. Next, we rearrange the elements of $\mathbf{a}$ as per the permutation, and store the result in a vector $\mathbf{b}$.

6. A random integer $d = 79317$ is chosen. By performing the computation $c_i = b_i + d$, noise is added to the system.

7. We publish

$$\begin{aligned} \mathbf{c} &= \langle 102518, 220487, 216223, 81903, 174571, 192333, \\ &\quad 180940, 159017, 132657, 153595, 116589\rangle \\ p &= 11 \\ h &= 5 \end{aligned}$$

   as the public key.

8. We keep

$$\begin{aligned} f(t) &= 1 + 7t^2 + 4t^3 + 8t^5 \\ g &= \langle 10, 7, 5, 8, 2\rangle \\ \pi^{-1} &= \{4, 6, 10, 8, 7, 5, 0, 1, 9, 2, 3\} \\ d &= 79317 \end{aligned}$$

   as the private key.

Encryption We choose a message to encrypt, and perform (10) to find the ciphertext S.

$$\begin{aligned} \mathbf{x} &= \langle 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0\rangle \\ S &= 52382 \end{aligned}$$

Decryption To obtain $\mathbf{x}$ from S, we compute:

$$\begin{aligned} r(t) &= 4 + 6t^2 + 5t^3 \\ S' &= 138947 \\ q(t) &= 6 + 10t + 3t^2 + 9t^3 + 8t^4 \\ s(t) &= 2 + 10t - 3t^2 + 4t^3 + 8t^4 + t^5 \end{aligned}$$

And we find, through successive substitutions, the 5 roots α_{i_j}'s, $s(t) = (3+t)(4+t)(6+t)(7+t)(10+t) = 5040+5004t+1900t^2+345t^3+30t^4+t^5$, which can be written as $\langle 0, 0, 1, 1, 0, 1, 1, 0, 0, 1\rangle$. By applying π^{-1}, we get the result $\langle 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0\rangle$ which is the initial message $\mathbf{x}$.

3 What is a Lattice?

A property common to many knapsack cryptosystems is that encryption for the message-sender is computationally easy, as many of these cryptosystems require only the addition of numbers from a provided set, the receiver's public key. This property provides an underlying structure to knapsacks which can be seen by mapping every element a_i in the public key to a vector that points in the i^{th} direction. Under this system, an encrypted message S can also be mapped to a vector which is the sum of the previously-defined vectors. A system with this geometry is called a *lattice*, and the remainder of this section will formally define the concept of a lattice, and introduce some properties of lattices relevant to the cryptanalysis of knapsack cryptosystems.

Definition 1. The *unit vector* along a direction $\mathbf{u}$, where $\mathbf{u} \in \mathbb{R}^n \setminus \{0\}$, has length 1 and is defined by

$$\hat{\boldsymbol{u}} := \frac{\mathbf{u}}{\|\mathbf{u}\|}.$$

Definition 2. The *projection* of a vector $\mathbf{v}$ onto a vector $\mathbf{u}$ is the $\mathbf{u}$ component of $\mathbf{v}$, and is defined by

$$\mathrm{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{(\mathbf{v}, \mathbf{u})}{(\mathbf{u}, \mathbf{u})}\mathbf{u},$$

where $(,)$ denotes the ordinary dot product.

Definition 3. Let $S \subseteq \mathbb{R}^d$, where $d \geq 1$, be a non-empty finite set. The *lattice* Λ generated by S is the set of integer linear combinations of the elements in S,

$$\Lambda = \Lambda(S) := \{m_1u_1 + m_2u_2 + \cdots + m_ku_k : k \geq 1, u_i \in S, m_i \in \mathbb{Z}\}.$$

Definition 4. If S has the minimum cardinality among generating sets for Λ, we call S a *basis* of Λ. The cardinality of a basis of Λ is the *dimension*, $\dim \Lambda$, of Λ.

In this paper, we will look only at the cases where a basis for Λ is a set of k linearly independent vectors, and $1 \leq k \leq d$, as is customary in these applications. We then look at the ordered column vectors $(u_1, \ldots, u_k)$ and define a matrix A:

$$A = [u_1, \ldots, u_k] \in \mathbb{R}^{d \times k}$$

and by the same convention, the matrix A has rank k, and we address this lattice as $\Lambda(A)$ as opposed to $\Lambda(S)$.

Definition 5. A square matrix U is *unimodular* if $\det U = \pm 1$. The inverse of a unimodular matrix is also unimodular.

Theorem 1. *Let $A, B \in \mathbb{Z}^{k \times k}$ be two bases. Then $\Lambda(A) = \Lambda(B)$ iff there exists an integer unimodular matrix U such that $A = BU$.*

A unimodular matrix U represents a unimodular transformation of lattice bases, from A to AU.

Definition 6. The *determinant* of a lattice Λ is given by

$$\det \Lambda := \sqrt{\det A^T A}$$

where A is a basis of Λ.

To clarify the implications of these terms, we will consider an example consisting of the following set of matrices, which we will use to build lattices:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 7 & 2 \\ 12 & 3 & 15 \end{bmatrix}, \quad B = \begin{bmatrix} 14 & 11 & 24 \\ 20 & 11 & -19 \\ 63 & 69 & 330 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 7 & 2 \\ 12 & 3 & 15 \end{bmatrix}$$

In this case, $\Lambda(A) = \Lambda(B)$ because there exists an integer matrix U with $\det U = 1$,

$$U = \begin{bmatrix} 1 & 3 & 27 \\ 2 & 1 & -3 \\ 3 & 2 & 1 \end{bmatrix},$$

and $B = AU$. Additionally, by calculating the determinants of $\Lambda(A)$ and $\Lambda(C)$, we can observe through their differing results (105 and 126, respectively,) that these similar-looking matrices generate different lattices. The idea of multiple generators for a given lattice, as seen with the A and B, raises several questions, including the existence of a "best" lattice basis for a particular application.

4 Lenstra–Lenstra–Lovász (LLL) Lattice Basis Reduction Algorithm

In the beginning of Section 3, we presented the idea that the elements of a knapsack can be used to create a set of vectors that will generate a basis of a lattice. By extension, any ciphertext generated from the knapsack can be represented as a vector that is a combination of the knapsack elements' vectors, making it an element in the lattice as well. Because the knapsack and the ciphertext are made public, a possible attack on many knapsack cryptosystems is lattice basis reduction: Given the described lattice, find the vectors that make up the ciphertext.

In 1982, a paper published by A.K. Lenstra, H.W. Lenstra Jr., and L. Lovász [5] described a computationally-efficient algorithm (LLL) for lattice basis reduction. In the following subsections, I will describe and present that algorithm, and provide an example of it in use on an elementary lattice.

4.1 Description

The LLL algorithm utilizes the information gathered by the Gram–Schmidt process to create a reduced basis for the lattice. Recall that the Gram–Schmidt process is used to create an orthonormal basis for a given set of vectors. While we are looking for an orthogonal basis, we are specifically not looking for an orthonormal basis, so the normalization method is removed from the Gram–Schmidt process. Further, we cannot use the exact results

from this modified Gram–Schmidt process because it is likely that those results will be composed with non-integer coefficients of the initials vectors, causing them to fall outside of the lattice.

Instead, the data from the Gram–Schmidt process is used to improve the orthogonality of the source vectors while keeping them in the lattice. Specifically, the Gram-Schmidt coefficient, $\mu_{kl} = \frac{(b_k, b_l^*)}{(b_l^*, b_l^*)}$, which evaluates the projection of a vector $\mathbf{b_k}$ in the lattice onto the Gram–Schmidt-orthogonalized vector $\mathbf{b_l^*}$, is used as a sensitivity indicator. If μ is too large (namely, if it is greater than $\frac{1}{2}$,) then the orthogonality of $\mathbf{b_k}$ in relation to $\mathbf{b_l}$ can be improved, so the following process similar to the orthogonalization routine found in Gram-Schmidt is performed:

$$b_k := b_k - r b_l. \tag{12}$$

That equation, where r is the integer nearest to μ_{kl}, causes the updated $\mathbf{b_k}$ to improve its orthogonality to $\mathbf{b_l}$, while ensuring that $\mathbf{b_k}$ remains in the lattice.

The LLL algorithm includes the appropriate modifications for its differences from the Gram-Schmidt process, and is written in a computationally-efficient (polynomial time) manner. The complete algorithm follows, in which $B_i = |b_i^*|^2$ and $(,)$ refers to the ordinary dot product.

$b_i^* := b_i$;
$\mu_{ij} := (b_i, b_j^*)/B_j$; $b_i^* := b_i^* - \mu_{ij} b_j^*$; $\Big\}$ for $j = 1, 2, \ldots, i-1$; $B_i := (b_i^*, b_i^*)$; $\Big\}$ for $i = 1, 2, \ldots, n$;
$k := 2$;

(1) perform (*) for $l = k - 1$;
if $B_k < (\frac{3}{4} - \mu_{k,k-1}^2) B_{k-1}$, go to (2);
perform (*) for $l = k-2, k-3, \ldots, 1$;
if $k = n$, terminate;
$k := k + 1$;
go to (1);

(2) $\mu := \mu_{k,k-1}$; $B := B_k + \mu^2 B_{k-1}$; $\mu_{k,k-1} := \mu B_{k-1}/B$;
$B_k := B_{k-1} B_k / B$; $B_{k-1} := B$;

$$\begin{pmatrix} b_{k-1} \\ b_k \end{pmatrix} := \begin{pmatrix} b_k \\ b_{k-1} \end{pmatrix};$$

$$\begin{pmatrix} \mu_{k-1j} \\ \mu_{kj} \end{pmatrix} := \begin{pmatrix} \mu kj \\ \mu_{k-1j} \end{pmatrix} \quad \text{for } j = 1, 2, \ldots, k-2$$

$$\begin{pmatrix} \mu_{ik-1} \\ \mu_{ik} \end{pmatrix} := \begin{pmatrix} 1 & \mu_{kk-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -\mu \end{pmatrix} \begin{pmatrix} \mu_{ik-1} \\ \mu_{ik} \end{pmatrix} \text{ for } i = k+1, k+2, \ldots, n;$$

if $k > 2$, then $k := k - 1$;
go to (1).

(*) if $|\mu_{kl}| > \frac{1}{2}$, then:

$$\begin{cases} r := \text{integer nearest to } \mu_{kl}; \; b_k := b_k - r b_l; \\ \mu_{kj} := \mu_{kj} - r\mu_{lj} \text{ for } j = 1, 2, \ldots, l-1; \\ \mu_{kl} := \mu_{kl} - r \end{cases}$$

Output: Reduced lattice basis $[b_1, \ldots, b_n]$

Figure 1: LLL lattice basis reduction algorithm

4.2 Example

Consider the following lattice constructed from the column vectors in $A = [\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}]$:

$$A = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 0 & 5 \\ 1 & 2 & 6 \end{bmatrix}$$

During the initialization, we set $\mathbf{b_1^*} := \mathbf{b_1}$ and calculate B_1, thus:

$$A = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 0 & 5 \\ 1 & 2 & 6 \end{bmatrix}, \mathbf{b_1^*} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, B_1 = 3$$

We then set $\mathbf{b_2^*} := \mathbf{b_2}$ and subtract from $\mathbf{b_2^*}$ the projection of $\mathbf{b_2^*}$ onto $\mathbf{b_1^*}$:

$$A = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 0 & 5 \\ 1 & 2 & 6 \end{bmatrix}, \mathbf{b_1^*} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, B_1 = 3, \mathbf{b_2^*} = \begin{bmatrix} -\frac{4}{3} \\ -\frac{1}{3} \\ \frac{5}{3} \end{bmatrix}, \begin{matrix} B_2 = \frac{14}{3} \\ \mu_{21} = \frac{1}{3} \end{matrix}$$

Next, $\mathbf{b_3^*} := \mathbf{b_3}$ is introduced, and we subtract its projections onto $\mathbf{b_j^*}$ for $j = 1, 2$:

$$A = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 0 & 5 \\ 1 & 2 & 6 \end{bmatrix}, \mathbf{b_1^*} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, B_1 = 3, \mathbf{b_2^*} = \begin{bmatrix} -\frac{4}{3} \\ -\frac{1}{3} \\ \frac{5}{3} \end{bmatrix}, \begin{matrix} B_2 = \frac{14}{3} \\ \mu_{21} = \frac{1}{3} \end{matrix}, \mathbf{b_3^*} = \begin{bmatrix} -\frac{3}{7} \\ \frac{9}{14} \\ -\frac{3}{14} \end{bmatrix}, \begin{matrix} B_3 = \frac{9}{14} \\ \mu_{31} = \frac{14}{3} \\ \mu_{32} = \frac{13}{14} \end{matrix}$$

Setting $k := 2$ concludes the initialization.

After one iteration of (1), the system goes unchanged and k is increased to 3. In (*), we quickly see that the projection μ_{32} is beyond the sensitivity value of $\frac{1}{2}$, so the "lattice orthogonalization" process is applied to $\mathbf{b_3}$. Because $B_k < (\frac{3}{4} - \mu_{k,k-1}^2)B_{k-1}$ is true, the swap of $\mathbf{b_2}$ and $\mathbf{b_3}$ outlined in (2) is performed and k decremented, with the result:

$$A = \begin{bmatrix} 1 & 4 & -1 \\ 1 & 5 & 0 \\ 1 & 4 & 2 \end{bmatrix}, A^* = \begin{bmatrix} 1 & -\frac{4}{3} & -\frac{3}{7} \\ 1 & -\frac{1}{3} & \frac{9}{14} \\ 1 & \frac{5}{3} & -\frac{3}{14} \end{bmatrix} \begin{matrix} B_1 = 3 & \mu_{21} = \frac{13}{3} \\ B_2 = \frac{2}{3} & \mu_{31} = \frac{1}{3} \\ B_3 = \frac{9}{2} & \mu_{32} = -\frac{1}{2} \end{matrix}, k = 2$$

This brings us back to (1) which leads to a test by (*) of the orthogonality of $\mathbf{b_1}$ and $\mathbf{b_2}$. Because μ_{21} is greater than the sensitivity $\frac{1}{2}$, the orthogonality process is applied. The $B_k < (\frac{3}{4} - \mu_{k,k-1}^2)B_{k-1}$ test passes, so the swap method is called while k is unchanged. The updated results are:

$$A = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix}, A^* = \begin{bmatrix} 1 & -\frac{4}{3} & -\frac{3}{7} \\ 1 & -\frac{1}{3} & \frac{9}{14} \\ 1 & \frac{5}{3} & -\frac{3}{14} \end{bmatrix} \begin{matrix} B_1 = 1 & \mu_{21} = 1 \\ B_2 = 2 & \mu_{31} = 0 \\ B_3 = \frac{9}{2} & \mu_{32} = \frac{1}{2} \end{matrix}, k = 2$$

At the next iteration of (1), because μ_{21} is again greater than $\frac{1}{2}$, the orthogonality process is applied to improve the orthogonality of $\mathbf{b_2}$ to $\mathbf{b_1}$. After this,

the $B_k < (\frac{3}{4} - \mu_{k,k-1}^2)B_{k-1}$ test fails, indicating that the two vectors have sufficiently different lengths. The variable k is incremented, and the results are updated:

$$A = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 2 \end{bmatrix}, A^* = \begin{bmatrix} 1 & -\frac{4}{3} & -\frac{3}{7} \\ 1 & -\frac{1}{3} & \frac{9}{14} \\ 1 & \frac{5}{3} & -\frac{3}{14} \end{bmatrix} \begin{array}{lllllll} B_1 & = & 1 & \mu_{21} & = & 0 & \\ B_2 & = & 2, & \mu_{31} & = & 0, & k = 3 \\ B_3 & = & \frac{9}{2} & \mu_{32} & = & \frac{1}{2} & \end{array}$$

In the next iteration of (1), μ_{32} is within the sensitivity threshold, so the orthogonalization process is not called. Additionally, the $B_k < (\frac{3}{4} - \mu_{k,k-1}^2)B_{k-1}$ test fails, indicating that $\mathbf{b_2}$ and $\mathbf{b_3}$ have sufficiently different lengths. The projection of $\mathbf{b_3}$ onto $\mathbf{b_1}$ (μ_{31}) is tested, and found to be within the bounds of sensitivity for the system. The algorithm terminates, with A as the reduced basis for the lattice generated by the initial collection of vectors. Note that the new basis has smaller dot products, and that is what is accomplished by LLL. This causes an LLL-reduced lattice basis to *tend* to have short vectors.

5 Cryptanalysis of Knapsack Cryptosystems Using Shortest Vectors

The structure of knapsack problems, such as those described in this paper, lends itself to attack through lattice reduction. The challenge is to determine which subset of numbers from a known (public) set was used to form the sum, which is sent as the encrypted message. On a basic level, we can generate an ordered set of column vectors that consist of each element in the set and append a column vector consisting of the encrypted message to form a lattice. Lattice basis reduction, performed on a lattice with the correct modifications for the particulars of a given knapsack cryptosystem, will often return the correct solution for the unencrypted message.

Shortest vector attacks rely on the problem having a low density d, defined for a knapsack composed of the vector $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$ by

$$d(\mathbf{a}) = \frac{n}{\log_2(\max_i a_i)}. \tag{13}$$

In knapsack cryptosystems, $d(\mathbf{a})$ can be used to approximate the *information rate* at which bits are transmitted; specifically

$$d(\mathbf{a}) \cong \frac{\text{number of bits in plaintext message}}{\text{average number of bits in ciphertext message}}. \tag{14}$$

Relating the density measure to the cryptosystems explored in this paper, the Merkle–Hellman knapsack cryptosystem has a relatively low density, while the density of knapsacks generated by the Chor–Rivest system are significantly higher. We will see the implications of that property later in this section.

5.1 Merkle-Hellman Knapsack Cryptanalysis

We generate a super-increasing private knapsack $\mathbf{a}' = \langle a'_1, a'_2, \ldots, a'_k \rangle$, and choose two coprime numbers m and w, where $m > \sum a'_i$. The public knapsack, $\mathbf{a} = \langle w \cdot a'_1 \bmod m, w \cdot a'_2 \bmod m, \ldots, w \cdot a'_n \bmod m \rangle = \langle a_1, a_2, \ldots, a_n \rangle$, and a ciphertext $S = \sum x_i a_i$ are published.

To find $\mathbf{x}$, we generate a lattice from the elements of $\mathbf{a}$, and the message S:

$$A = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 1 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 1 & 0 \\ a_1 & a_2 & \cdots & \cdots & a_n & S \end{bmatrix}$$

In general, if $\bar{\boldsymbol{v}}$ is a short vector in the lattice, then

$$\bar{\boldsymbol{v}} = c_1\bar{\boldsymbol{a_1}} + c_2\bar{\boldsymbol{a_2}} + \ldots + c_n\bar{\boldsymbol{a_n}} + c_{n+1}\bar{\boldsymbol{S}} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ c_1a_1 + \ldots + c_{n+1}S \end{bmatrix}, \qquad (15)$$

and every component c_i should be small.

By the construction of S, we know that the lattice $\Lambda(A)$ contains a vector comprised of some elements of $\mathbf{a}$, and the message S. Therefore, the vector

$$\bar{\boldsymbol{w}} = \sum_{i=1}^{n} x_i\bar{\boldsymbol{a_i}} - \bar{\boldsymbol{S}} \qquad (16)$$

is in the lattice, and $\bar{\boldsymbol{w}}$ is *short*, in particular, $(\bar{\boldsymbol{w}}, \bar{\boldsymbol{w}}) = |I| \leq k$. Using LLL to find short vectors in $\Lambda(A)$ will, in many cases, return the short vector $\bar{\boldsymbol{w}}$.

5.2 Using LLL in the Cryptanalysis of the Chor-Rivest Knapsack Cryptosystem

Due to the relatively higher density of the Chor–Rivest cryptosystem, the shortest vector attack used against the Merkle–Hellman cryptosystem will often fail to find the shortest vector when this cryptosystem is used. Instead, modifications to the lattice basis reduction algorithm have been proposed with the goal of using the public key and message to find other vectors in the lattice, and in turn finding a lattice basis using those vectors.

Several modifications have been proposed, and here we present one of the earliest adjustments to LLL, published by J. C. Lagarias and A. M. Odlyzko [4].

(1) Take the following vectors as a basis $[\mathbf{b_1}, \ldots, \mathbf{b_{n+1}}]$ for an $n+1$-dimensional integer lattice $\Lambda = \Lambda(A)$. Here, each column of A is a vector $\mathbf{b_i}$:

$$A = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 1 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 1 & 0 \\ -a_1 & -a_2 & \cdots & \cdots & -a_n & S \end{bmatrix}.$$

(2) Find a reduced basis $[\mathbf{b_1^*}, \ldots, \mathbf{b_{n+1}^*}]$ of Λ using the LLL algorithm.

(3) Check if any $\mathbf{b_i^*} = \langle b_{i,1}^*, \ldots, b_{i,n+1}^* \rangle$ has all $b_{ij} = 0$ or some fixed λ, for $1 \leq j \leq n$. For any such $\mathbf{b_i^*}$, check whether $x_j = \lambda^{-1} b_{ij}^*$ for $1 \leq j \leq n$ gives a solution to the problem

$$S = \sum_{i=1}^{n} x_i a_i,$$

and if so, halt. Otherwise, continue.

(4) Repeat steps (1)–(3) with S replaced by $S' = \sum_{i=1}^{n} a_i - S$. Then halt.

If the algorithm produces a valid solution, it is said to succeed; otherwise, it fails.

Figure 2: Algorithm SV (Shortest Vector)

It is observed that Algorithm SV is not radically different from the LLL algorithm. The general method is the same in both algorithms, with the variation that if a valid solution is not found after the initial LLL run, the ciphertext vector is replaced with a vector of different length before LLL is re-run. This improves the likelihood that LLL will find a short vector, and "almost always" yields the correct solution for knapsacks with a density $d < 0.645$.

This attack was tested against the Chor–Rivest cryptosystem and its results were published with the cryptosystem. Using carefully chosen parameters p and h, the attack is averted due to the increased density of the

knapsack. For example, using $p = 103$ and $h = 12$, the density is 1.271, and those parameters are of a smaller magnitude than the recommendations of Chor and Rivest.

Further shortest vector attacks have been made against the Chor-Rivest cryptosystem with modifications to account for its high density. An attack by Coster et al. [3], builds upon the Algorithm SV published by Lagarias and Odlyzko. They present two modifications of the algorithm, either of which will solve "almost all" problems of density < 0.9408. Several years later, Schnorr and Hörner [7] presented work which solves, with positive probability, the shortest vector problem for knapsacks with arbitrary densities. In testing, they found it capable of breaking the Chor–Rivest cryptosystem for certain parameters which were previously unbreakable by lattice reduction, but carefully selected parameters again decreased the effectiveness of their attack.

6 Conclusion

This paper presented the well-known "subset sum" problem and explored its use in cryptography. Although the standard problem is NP-complete, knapsack cryptosystems include trapdoors in the generation phase of the knapsack in order to facilitate a non-brute-force, quick decryption for the authorized user, while utilizing tools from number theory to conceal that trapdoor from unauthorized parties.

The Merkle–Hellman knapsack cryptosystem uses a super-increasing set as its trapdoor, and implements modular reduction to hide that property. The Chor–Rivest system utilizes finite field arithmetic and hamming weight to increase the complexity of that system's key and trapdoor generation. These systems, like all other knapsack-based cryptosystems, share a common property: encryption is easy; which is to say that the ciphertexts generated by these systems represent sums of unique subsets of elements in the knapsack.

Because of this property, one method of attack against this kind of cryptosystem is to brute-force the ciphertext by looking at all possible subsets of the knapsack. That method is extremely inefficient, with 2^n possible combinations to examine in the worst case, and it quickly becomes impractical computationally as the size of the knapsack increases. The idea of utilizing the elements of the knapsack to find the ciphertext is instead built upon through lattice representation.

Representing the elements of the knapsack as linearly independent vectors in a lattice allows us to represent the ciphertext as a combination of those vectors. Using this method, if one finds a basis of the lattice, then there will often be a short vector composed of the elements of the knapsack used to create the ciphertext. Although it is difficult to *always* find a basis which includes that short vector, lattice basis reduction algorithms like the LLL algorithm can "almost always" find that short vector of many lattices in a computationally-feasible amount of time.

To counteract attacks such as lattice basis reduction, later knapsack-based cryptosystems utilize different tools in the generation phase to increase the "random" property of its elements. As the cryptosystems become more complex, reduction-based attacks have become more sophisticated, as seen by the Chor–Rivest attacks discussed in Section 5.2. This demonstrates the versatility of lattice basis reduction in the cryptanalysis of knapsack-based cryptosystems, and lends credence to the opinion of many that all knapsack-based cryptosystems are insecure, even if the particulars for a specific system have not yet been found.

7 Acknowledgements

The eighth lecture in *Fundamental Problems in Algorithmic Algebra* [8] was referenced when writing the formal lattice definitions presented in Section 3. I would like to thank Dr. Kevin O'Bryant for his mentorship and guidance during my cryptography studies.

References

[1] R. C. Bose and S. Chowla. Theorems in the additive theory of numbers. *Commentarii Mathematici Helvetici*, 1962.

[2] Benny Chor and Ronald L. Rivest. A knapsack type public key cryptosystem based on arithmetic in finite fields. *IEEE Trans. Inform. Theory*, 1988.

[3] Matthijs J. Coster, Antoine Joux, Brian A. Lamacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 1991.

[4] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 1985.

[5] A.K. Lenstra, H.W. Lenstra Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 1982.

[6] Ralph C. Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions On Information Theory*, 1978.

[7] C. P. Schnorr and H. H. Hörner. Attacking the chor-rivest cryptosystem by improved lattice reduction. In *EUROCRYPT'95: Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*, pages 1–12, Berlin, Heidelberg, 1995. Springer-Verlag.

[8] Chee Yap. *Fundamental Problems in Algorithmic Algebra*. Oxford University Press, 2000.