The Use of a Lattice Reduction Algorithm on the Merkle-Hellman cryptosystem's security

Olga Kucharska

VIII Prywatne Akademickie Liceum Ogólnokształcące w Krakowie

Opiekun pracy: mgr Alicja Rozpędzka

Table of Contents

Introduction and background information to the cryptosystem 2

Subset sum problem 3

The Merkle-Hellman knapsack cryptosystem 6

Encryption of a message 7

Decryption of a message 12

Defining lattices 18

Lattice reduction and the LLL algorithm 20

Attacking the cryptosystem 27

Conclusion 31

Bibliography 33

Appendix 36

Binary Representations 36

LLL Algorithm 37

Introduction and background information to the cryptosystem

The study of cryptography has evolved from simple ciphers into more reliable techniques of conveying information. Contemporary cryptography is crucial as it is used to protect confidential information. Public-key cryptography refers to systems which use a public key and a private key to exchange messages (Hellman 1978, 25–26). The public key can be displayed publicly, while the private key remains secret. It works in a way that anyone can encrypt a message using the public key, creating ciphertext, but only the possessor of the private key can decrypt it and obtain the message sent.

An early example of public-key cryptography is the Merkle-Hellman cryptosystem invented in 1978 by Ralph Merkle and Martin Hellman (Liu et al. 2019, 1). It is based on a special case of the subset sum problem. The cryptosystem was broken in 1981 and was since considered insecure. One of the methods which can be used to attack it is lattice reduction, which almost always yields the original message, without acquiring the private key used for decryption.

The paper will examine the use of the Lenstra–Lenstra–Lovász lattice reduction algorithm, known as the LLL algorithm, in attacking the Merkle-Hellman cryptosystem. The usage of the LLL algorithm on the Merkle-Hellman cryptosystem will be analyzed by encrypting and decrypting a chosen message.

Subset sum problem

The Merkle-Hellman cryptosystem bases on the subset sum problem which is defined below (Stamp and Low, 2007):

Given a sequence W of r weights,

$$W = (w_0, w_1, \ldots, w_{r-1})$$

where $r \in \mathbb{Z}^+ \cup \{0\}$ and $w_i \in \mathbb{Z}^+$ and a sum $S, S \in \mathbb{Z}^+$,

find a bit sequence $x = (x_0, x_1, \ldots, x_{r-1})$, where $x_i \in \{0,1\}$, meaning x_i is a binary number, and $i \in \{0, 1, \ldots, r-1\}$., so that

$$S = \sum_{i=0}^{r-1} x_i w_i$$

For the purpose of illustration suppose the weights are $W = (4, 8, 3, 1, 23, 40), r = 6$ and the given sum is $S = 34$. Then, a solution to the subset sum problem is given by a unique bit sequence $x = (011010)$, since

$$0 \cdot 4 + 1 \cdot 8 + 1 \cdot 3 + 0 \cdot 1 + 1 \cdot 23 + 0 \cdot 40 = 34$$

The solution is unique in this case because there does not exist another way to obtain the sum S, when $x_i \in \{0,1\}$. This can be checked by comparing x_i with the sum S, as presented further.

40 cannot be a part of the solution because it exceeds the sum 34, hence the bit corresponding to it is 0. 23 must be used to reach the sum because the sum of all the other numbers is $4 + 8 + 3 + 1 < 34$. The sum that must be reached with the usage of $\{4, 8, 3, 1\}$ is $34 - 23 = 11$, which is only possible by adding $8 + 3 = 11$. Therefore, the

bits corresponding to 8 and 3 are 1s and the bits corresponding to 4 and 1 are 0s. Thus, it is ensured that $x = (011010)$ is the only solution.

Nevertheless, there exist examples of the subset sum problem that have more than one solution, therefore the bit sequence x is not always unique. Suppose W has not changed and the given sum is now $S = 35$. Then the possible solutions are

$x = (011110)$, since

$$0 \cdot 4 + 1 \cdot 8 + 1 \cdot 3 + 1 \cdot 1 + 1 \cdot 23 + 0 \cdot 40 = 35$$

or $x = (110010)$, since

$$1 \cdot 4 + 1 \cdot 8 + 0 \cdot 3 + 0 \cdot 1 + 1 \cdot 23 + 0 \cdot 40 = 35$$

The case is that when different weights w_i can be summed to obtain the same result, there will be more than one solution. The subset sum problem has no solution if it is impossible to obtain the sum S for the given set of weights. Let us take the previous $W = (4, 8, 3, 1, 23, 40)$ and a new sum $S = 10$. In this case, there does not exist a way to obtain the solution. Another example with no solution, using the unchanged W, occurs when the sum $S = 6$.

A type of the subset sum problem is a superincreasing knapsack. A superincreasing knapsack is a sequence of r weights ordered from the least to the greatest, where each next weight is greater than the sum of all the previous weights

$$W_n > \sum_{i=0}^{n-1} W_i$$

for all $n \in \{2,3, \dots, r-1\}$ (Stamp and Low 2007, 267-268). For example,

$$W = (2, 5, 18, 26, 54, 106, 219, 447)$$

is a superincreasing knapsack.

To find the sequence x, that is the solution to the knapsack, it will be checked which values are needed to obtain the sum S. Suppose the given sum is $S = 374$. Since $S < 447$, it can be concluded that $x_7 = 0$. Next,

$$2 + 5 + 18 + 26 + 54 + 106 = 211$$

which is less than 219, so $x_6 = 1$, because it would be impossible to obtain S without x_5, due to the sum off all remaining values being less than 219. Then,

$$2 + 5 + 18 + 54 = 79$$

and

$$79 + 219 < 374$$

so it must be the case that $x_4 = 1$. Otherwise, the sum S could not be reached.

Now let

$$S_1 = S - (219 + 106) = 49$$

considering that

$$26 < S_1 < 54 \text{ and } x_4 = 0 \text{ and } x_3 = 1$$

Continuing this process,

$$2 + 5 + 26 + 106 + 219 < 374$$

so $x_2 = 1$. Currently, having obtained x_i, $i \in \{2,3,\ldots,7\}$ the sum is equal to 369, so the only possible case is that $x_1 = 1$ and $x_0 = 0$ in order to solve this superincreasing knapsack, hence,

$$x = (01110110)$$

To verify the answer:

$$0 \cdot 2 + 1 \cdot 5 + 1 \cdot 18 + 1 \cdot 26 + 0 \cdot 54 + 1 \cdot 106 + 1 \cdot 219 + 0 \cdot 447 = 374 = S$$

therefore, the obtained answer $x = (01110110)$ is true. This algorithm can be used to solve any superincreasing knapsack. Additionally, because of its superincreasing property there always exists at most one solution, because once you reach the target sum, no subsequent weights can be added, because they will exceed the sum.

The Merkle-Hellman knapsack cryptosystem

The idea behind the Merkle-Hellman cryptosystem is creating a public and private key. The private key has a form of a superincreasing sequence, a modulo and a factor. The superincreasing sequence is transformed into the public key. After a message is sent to the receiver, the receiver uses his private key to reverse the encryption process and is able to solve the original superincreasing sequence to obtain the message.

The cryptosystem operates based on a superincreasing knapsack, which is a sequence of numbers used to encode and decode information, however, as presented earlier, obtaining a solution to one is relatively straightforward. To disguise the knapsack, in other words, make the private key unidentifiable for a person from the outside, Merkle and Hellman's idea was to mathematically transform it using a modulo

and factor (Stamp and Low 2007, 268). This way, the newly obtained knapsack T will not be superincreasing, because of the usage of a modulo, and hence, it will be impossible to solve it in that same, simple way. For the purpose of this paper let us take Alexander and Benjamin as the sender and recipient of a message, respectively. As the Merkle-Hellman cryptosystem is a public-key cryptosystem, the disguised knapsack T is Benjamin's public-key, and it is made public. Benjamin then receives ciphertext from Alexander and applies the inverse of the transformation to obtain a superincreasing case and then solves the knapsack, by finding the combination of numbers in the knapsacks that add up to the ciphertext values.

Encryption of a message

In this section an example of message encryption will occur. The first step is to choose a superincreasing knapsack K, which will enable Benjamin to create his public key and private key. Additionally, Benjamin chooses a modulo m, which is the remainder of a division, and factor f, satisfying the conditions that: $gcd(m, f) = 1$, or in other words m and f are coprime, and m is greater than the sum of all the elements of K (Stamp and Low 2007, 268). This ensures that the public key will consist of greater and more varied numbers, making the encoded message more secure. A smaller m would result in a public key consisting of smaller numbers because there would be fewer possible remainders, which would make the message vulnerable.

Suppose the knapsack chosen to create the private and public keys is

$K = (3, 5, 24, 33, 66, 135, 267, 546)$ and $m = 1081$ and $f = 19$. The conditions are met as

$$m > 3 + 5 + 24 + 33 + 66 + 135 + 267 + 546 = 1079$$

m is chosen as 1081, and not 1080 because 1080 has more factors, hence it would be more difficult to find a factor f, with $gcd(m, f) = 1$. The set of factors of 1081 is $\{1, 23, 47, 1081\}$ and f is a prime number, so m and f are coprime. To convert the private key K into the public key T, Benjamin computes T as follows:

$$T = (t_0, t_1, \ldots, t_{r-1}) = \left(k_0 f (mod\ m), k_1 f (mod\ m), \ldots, k_{r-1} f (mod\ m)\right)$$

where k_i is the ith element of K, $i \in \{0,1,\ldots,r-1\}$.

Benjamin's private key is now: K and the modular inverse: $f^{-1}(\mathrm{mod}\ m)$, which will be used to reverse the encryption process. The modular inverse is obtained below using the extended Euclidean algorithm (Extended Euclidean Algorithm Calculator):

$$1081 = 19 \cdot 56 + 17$$

$$19 = 17 \cdot 1 + 2$$

$$17 = 2 \cdot 8 + 1$$

$$1 = 17 \cdot 1 - 2 \cdot 8$$

$$1 = 17 \cdot 1 - (19 - 17 \cdot 1) \cdot 8$$

$$1 = 17 \cdot 1 - 19 \cdot 8 + 17 \cdot 8$$

$$1 = 17 \cdot 9 - 19 \cdot 8$$

$$1 = (1081 - 19 \cdot 56) \cdot 9 - 19 \cdot 8$$

$$1 = 1081 \cdot 9 - 19 \cdot 9 \cdot 56 - 19 \cdot 8$$

$$1 = 1081 \cdot 9 - 19 \cdot (9 \cdot 56 + 8)$$

$$1 = 1081 \cdot 9 - 19 \cdot 512$$

$$-512 (\text{mod } 1081) = 569$$

hence

$$19^{-1} = 569 (mod\ 1081)$$

T is calculated below, by the formula provided earlier:

$$t_0 = k_0 f (\text{mod } m) = (3 \cdot 19) (\text{mod } 1081) = 57 (\text{mod } 1081) = 57$$

$$t_1 = k_1 f (\text{mod } m) = (5 \cdot 19) (\text{mod } 1081) = 95 (\text{mod } 1081) = 95$$

$$t_2 = k_2 f (\text{mod } m) = (24 \cdot 19) (\text{mod } 1081) = 456 (\text{mod } 1081) = 456$$

$$t_3 = k_3 f (\text{mod } m) = (33 \cdot 19) (\text{mod } 1081) = 627 (\text{mod } 1081) = 627$$

$$t_4 = k_4 f (\text{mod } m) = (66 \cdot 19) (\text{mod } 1081) = 1254 (\text{mod } 1081) = 173$$

$$t_5 = k_5 f (\text{mod } m) = (135 \cdot 19) (\text{mod } 1081) = 2565 (\text{mod } 1081) = 403$$

$$t_6 = k_6 f (\mathrm{mod}\ m) = (267 \cdot 19)(\mathrm{mod}\ 1081) = 5073(\mathrm{mod}\ 1081) = 749$$

$$t_7 = k_7 f (\mathrm{mod}\ m) = (546 \cdot 19)(\mathrm{mod}\ 1081) = 10374(\mathrm{mod}\ 1081) = 645$$

Hence, the public key is

$$T = (57, 95, 456, 627, 173, 403, 749, 645)$$

It is worth noticing that t_0, t_1, t_2, t_3 are each equal to $k_i f$. This occurs because in these four cases $k_i f < m$. We can derive that the greater the modulo, the smaller the number of public key elements which need an additional step of computing may be. Therefore, the greater the modulo, the less secure may the cryptosystem be. And, inversely, the smaller the modulo, the more elements need an additional step of computing, and it may be more difficult for an attacker to infer the private key. Due to the fact that an additional operation is needed to obtain the public key, it is more difficult for a potential attacker to understand in what way the private key values were picked.

Benjamin's private key is

$$K = (3, 5, 24, 33, 66, 135, 267, 546)$$

and

$$f^{-1}(\mathrm{mod}\ m) = f^{-1}(\mathrm{mod}\ 1081) = 569$$

Suppose, Alexander wants to encrypt the message $M = dog$. M must be converted to binary but each of the letters takes up 8 bits, so in total there will be 24 bits, while the public key allows only 8-bit structures, since it consists of an eight-element sequence, as $r = 8$. To solve this problem each letter will be encrypted separately as follows: $M_1 = d, M_2 = o, M_3 = g$. $M_1 = 01100100$, $M_2 = 01101111$, $M_3 = 01100111$, according to the binary representations table present in the Appendix (Figure 1). When using the binary notation of letters in ASCII the first bit is always 0 because the letters consist of 7 bits. When Benjamin receives the message, he will have to decrypt each letter separately, which is a limitation that is analyzed further on in this paper.

To compute the ciphertext C Alexander sums each element of T multiplied by the corresponding bits of M, as follows:

$$C_1 = 0 \cdot 57 + 1 \cdot 95 + 1 \cdot 456 + 0 \cdot 627 + 0 \cdot 173 + 1 \cdot 403 + 0 \cdot 749 + 0 \cdot 645 = 954$$

$$C_2 = 0 \cdot 57 + 1 \cdot 95 + 1 \cdot 456 + 0 \cdot 627 + 1 \cdot 173 + 1 \cdot 403 + 1 \cdot 749 + 1 \cdot 645 = 2521$$

$$C_3 = 0 \cdot 57 + 1 \cdot 95 + 1 \cdot 456 + 0 \cdot 627 + 0 \cdot 173 + 1 \cdot 403 + 1 \cdot 749 + 1 \cdot 645 = 2348$$

This process relates to the subset sum problem because specific elements are selected to form a sum, which is the ciphertext. These three parts of ciphertext when put together form the message. The message is secure because someone from the outside, that obtained the ciphertext cannot simply match it to a letter represented in binary.

Decryption of a message

The computed ciphertext C is sent to Benjamin and he reverses the encryption process using his private key, by operating on the received ciphertext and by using the modular inverse.

$$C_1 f^{-1} (\text{mod } m) = (954 \cdot 569) (\text{mod } 1081) = 164$$

$$C_2 f^{-1} (\text{mod } m) = (2521 \cdot 569) (\text{mod } 1081) = 1043$$

$$C_3 f^{-1} (\text{mod } m) = (2348 \cdot 569) (\text{mod } 1081) = 977$$

Due to these computations, Benjamin is able to reverse the encryption, by using the modular inverse f^{-1}.

Benjamin's next step is to solve the superincreasing knapsack K for each of the obtained numbers: 164, 1041, and 977. When solving the superincreasing knapsack *K*, if a number k_i is needed to obtain the sum, the bit corresponding to it is 1, otherwise the bit corresponding to it is 0. Below, the superincreasing knapsack

$$K = (3, 5, 24, 33, 66, 135, 267, 546)$$

is solved for 977 and the message D_3 is uncovered in binary in detail, with reference to solving the subset sum problem in a superincreasing case. The order of decrypting the parts of the message does not matter, because they are not dependent on each

other. To visualize uncovering the message in binary step by step, D_3 is represented in the following way:

$$D_3 = (\overline{y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7})$$

where $y_i \in \{0, 1\}$ and y_i means that y_i is the i^{th} digit of an r-digit number D_3, $i \in \{0,1,\dots, r-1\}$, where each y_i will be exchanged with 0 or 1. First, it is checked whether it is possible to obtain 977 without 546, this is done by adding all the values except 546;

$$3 + 5 + 24 + 33 + 66 + 135 + 267 = 533$$

and $533 < 977$, hence the bit corresponding to 546 is 1.

$$D_3 = (\overline{y_0 y_1 y_2 y_3 y_4 y_5 y_6 1})$$

The same process is repeated with the next number.

$$3 + 5 + 24 + 33 + 66 + 135 + 546 = 812$$

which means that 267 is needed and the bit corresponding to it is also 1.

$$D_3 = (\overline{y_0 y_1 y_2 y_3 y_4 y_5 11})$$

Next, summed up are all the values except 135;

$$3 + 5 + 24 + 33 + 66 + 267 + 546 = 944$$

so 135 is needed, so the corresponding bit is 1.

$$D_3 = (\overline{y_0 y_1 y_2 y_3 y_4 111})$$

Next, all values are summed up except for 66;

$$3 + 5 + 24 + 33 + 135 + 267 + 546 = 1013, \text{ and } 1013 > 977$$

hence the bit corresponding to 66 is 0.

$$D_3 = (\overline{y_0 y_1 y_2 y_3 0111})$$

Next, all the values except 33 and 66, which was concluded to be unnecessary, are summed;

$$3 + 5 + 24 + 135 + 267 + 546 = 980 \text{ and } 980 > 977$$

therefore, the bit corresponding to 33 is 0.

$$D_3 = (\overline{y_0 y_1 y_2 00111})$$

For 24;

$$3 + 5 + 135 + 267 + 546 = 956$$

hence the bit corresponding to 24 is 1.

$$D_3 = (\overline{y_0 y_1 100111})$$

Now, 3 and 5 are left, so it can be concluded that to reach the desired sum, the bit corresponding to 5 is 1 and the bit corresponding to 3 is 0. This can be checked by adding all the values that correspond with 1 bits:

$$5 + 24 + 135 + 267 + 546 = 977$$

and

$$D_3 = (01100111)$$

Therefore, $D_3 = (01100111)$ and its binary notation corresponds to the character *g,* as presented in the Binary Representations Table (see Appendix). Below D_1 and D_2 are solved using the same principles:

To obtain D_1 the knapsack $K = (3, 5, 24, 33, 66, 135, 267, 546)$ is solved for 164. Since,

$$546 > 164 \text{ and } 267 > 164$$

The last two elements of the sequence are 0s, $D_1 = (\overline{y_0 y_1 y_2 y_3 y_4 y_5 00})$. Next,

$3 + 5 + 24 + 33 + 66 = 131$, hence $D_1 = (\overline{y_0 y_1 y_2 y_3 y_4 100})$

Then,

$3 + 5 + 24 + 33 + 135 = 200$ and $D_1 = (\overline{y_0 y_1 y_2 y_3 0100})$

Similarly,

$3 + 5 + 24 + 135 = 167$, therefore $D_1 = (\overline{y_0 y_1 y_2 00100})$

and

$3 + 5 + 135 = 143$, hence $D_1 = (\overline{y_0 y_1 100100})$

and

$3 + 24 + 135 = 162$, therefore $D_1 = (\overline{y_0 1100100})$

Now it is visible, that

$5 + 24 + 135 = 164$, therefore $D_1 = (01100100)$

The obtained 8-bit structure is $D_1 = d$, using the Binary Representations Table (Figure 1) .

Below the knapsack $K = (3, 5, 24, 33, 66, 135, 267, 546)$ is solved for 1043. Let us start with $D_2 = (\overline{y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7})$,

$$3 + 5 + 24 + 33 + 66 + 135 + 267 = 533$$

hence 546 is needed, and

$$D_2 = (\overline{y_0 y_1 y_2 y_3 y_4 y_5 y_6 1})$$

Then using the same principles:

$3 + 5 + 24 + 33 + 66 + 135 + 546 = 812$, hence $D_2 = (\overline{y_0 y_1 y_2 y_3 y_4 y_5 11})$

and

$3 + 5 + 24 + 33 + 66 + 267 + 546 = 944$, therefore $D_2 = (\overline{y_0 y_1 y_2 y_3 y_4 111})$

and

$3 + 5 + 24 + 33 + 135 + 267 + 546 = 1013$, hence $D_2 = (\overline{y_0 y_1 y_2 y_3 1111})$

Next,

$3 + 5 + 24 + 66 + 135 + 267 + 546 = 1046$, hence $D_2 = (\overline{y_0 y_1 y_2 01111})$

and

$$3 + 5 + 66 + 135 + 267 + 546 = 1022, \text{ therefore, } D_2 = (\overline{y_0 y_1 101111})$$

The numbers left are 3 and 5 and it can be observed that

$$1038 + 5 = 1043, \text{ therefore, } D_2 = (01101111)$$

Alexander finds that $D_1 = d$, using the Binary Representations Table.

Taken together, he has uncovered three letters: *d, o, g.* After combining them into one message, $D = dog$ it can be observed that it is the same message that was sent by Alexander, $M = dog$. This outcome confirms that the process of encryption and decryption was successful. Encryption and decryption of a short word is easier than when dealing with multiple bit sequence cases, but it can show how the operations are carried out and can demonstrate the Merkle-Hellman cryptosystem's effectiveness in conveying information.

The upcoming sections of this paper combine knowledge of public-key cryptography and linear algebra to perform an attack on the Merkle-Hellman cryptosystem.

Defining lattices

To understand the lattice reduction attack, a few definitions and principles of linear algebra must be understood. In this paper, vectors are denoted using the international standard notation $\vec{v} = \begin{pmatrix} v_a \\ v_b \end{pmatrix}$. For the purpose of this paper let us define a lattice as the

set of linear combinations of linearly independent vectors $\{v_1, v_2, \ldots, v_n\} \in \mathbb{R}^b$, where b is the number of directions in which the vectors can exist, where $n \geq b$ with coefficients in $\mathbb{Z}$ (Micciancio 2013, 3),

$$L = a_1v_1 + a_2v_2 + \ldots + a_nv_n$$

where $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ (Polách 2022). Linear independence is defined as follows:

A set of vectors $\{v_1, v_2, \ldots, v_n\}$ is linearly independent if the vector equation

$$x_1v_1 + x_2v_2 + \ldots + x_nv_n = 0$$

has only one solution, that is:

$$x_1 = x_2 = \cdots = x_n = 0$$

(Margalit and Rabinoff 2019).

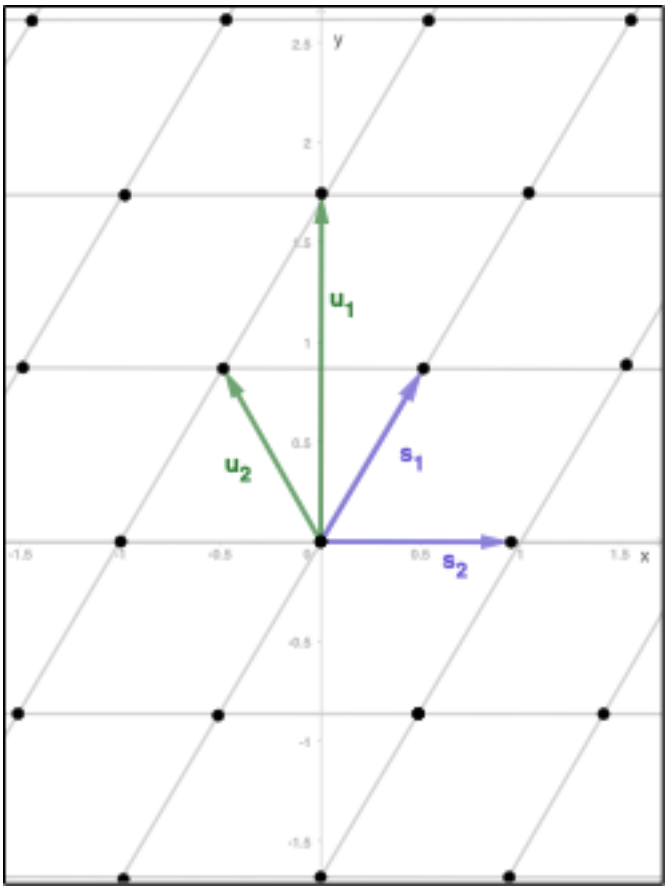

Figure 2: A lattice spanned by different bases (created by the author of this paper)

Figure 2 represents an example lattice spanned, meaning generated, using linear combinations of the basis vectors, by the vectors $\overrightarrow{s_1}$ and $\overrightarrow{s_2}$. For the purpose of further analysis, it is important to understand that the same lattice can be spanned by different bases, here, represented as $\overrightarrow{u_1}$, $\overrightarrow{u_2}$ (Louis 2023). A basis is a set of linearly independent vectors.

Lattice reduction and the LLL algorithm

Lattice reduction is a technique that can be used to find short, nearly orthogonal vectors. The orthogonality defect can be used to check if vectors are nearly orthogonal. It compares the product of the lengths of the basis vectors with the volume

of the parallelepiped they define (Wikipedia contributors, “Lattice Reduction” 2024). This means that the closer the orthogonality defect is to 1, the more orthogonal the basis vectors are. The LLL algorithm is a lattice reduction algorithm invented by Lenstra, Lenstra and Lovász in 1982. One of the subparts of the LLL is the Gram-Schmidt Process, which is responsible for finding orthonormal vectors, meaning ones that are perpendicular and of the same length. The process is performed as follows (Taboga 2024):

The vectors that we start with are $\overrightarrow{s_1}$ and $\overrightarrow{s_2}$. The vectors that are to be obtained are denoted as $\overrightarrow{u_1}$ and $\overrightarrow{u_2}$. The notions used in the Gram-Schmidt Process are: the magnitude and the dot product of two vectors. For vectors to be orthonormal, the conditions below must be met:

For each vector $\overrightarrow{u_1}, \ldots, \overrightarrow{u_k}$:

$$\left|\overrightarrow{u_j}\right| = 1 \text{ for any } j \text{ and } \overrightarrow{u_j} \bullet \overrightarrow{u_k} = 0 \text{ if } j \neq k$$

The first step is normalization, which refers to changing a vector into a unit vector:

$$\overrightarrow{u_1} = \frac{1}{|\overrightarrow{s_1}|} \cdot \overrightarrow{s_1}$$

then a vector projection $\hat{s}_2$ is obtained by mapping the vector $\overrightarrow{s_2}$ onto a line parallel to $\overrightarrow{u_1}$

$$\hat{s_2} = (\overrightarrow{s_2} \bullet \overrightarrow{u_1}) \cdot \overrightarrow{u_1}$$

next compute residual, which is the part of the vector that is not aligned with the direction of the projection:

$$\overrightarrow{\varepsilon_2} = \overrightarrow{s_2} - \widehat{s_2}$$

And finally, normalize:

$$\overrightarrow{u_2} = \frac{1}{|\overrightarrow{\varepsilon_2}|} \cdot \overrightarrow{\varepsilon_2}$$

For the purpose of understanding how this process works an attempt of accomplishing it is presented below.

Let us consider the vectors:

$$\overrightarrow{s_1} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \text{ and } \overrightarrow{s_2} = \begin{pmatrix} 0 \\ -2 \end{pmatrix}$$

presented below in a visual form:

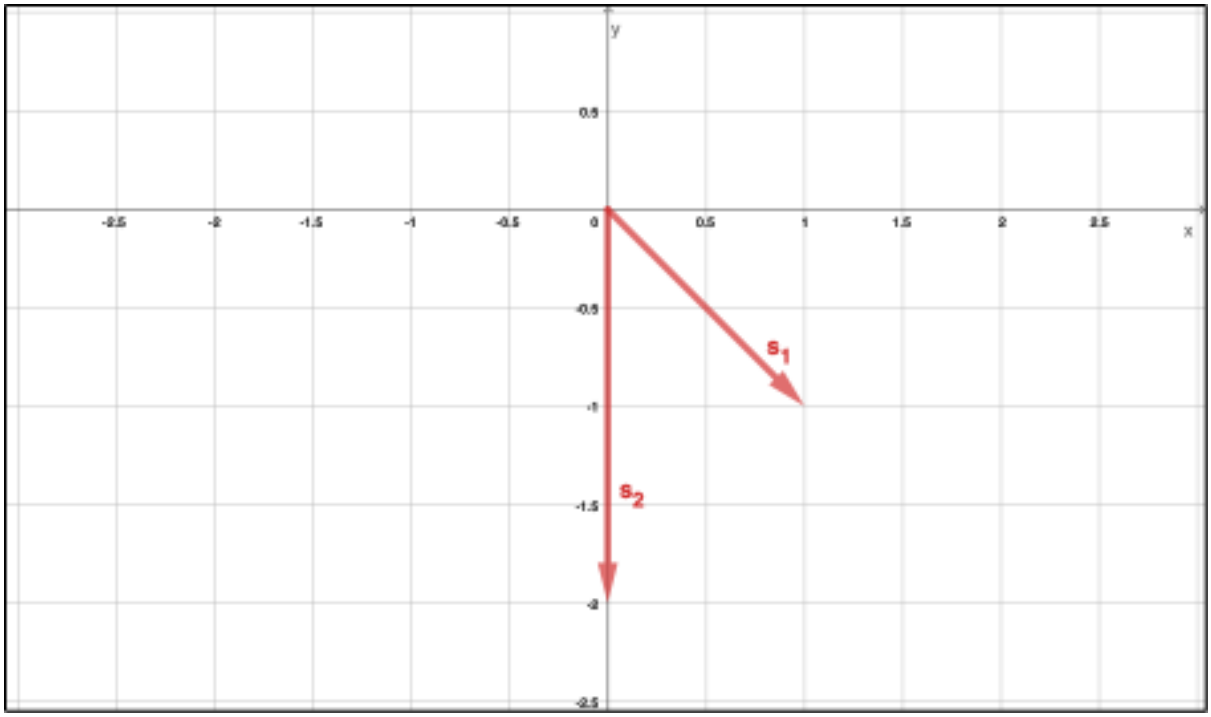

Figure 3: The vectors $\overrightarrow{s_1}$ and $\overrightarrow{s_2}$ presented on a plane using GeoGebra

The Gram-Schmidt Process is presented below with reference to the theoretical framework explained on pages 21 and 22.

$$\overrightarrow{u_1} = \frac{1}{|\overrightarrow{s_1}|} \cdot \overrightarrow{s_1} = \frac{1}{\sqrt{1^2 + (-1)^2}} \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix}$$

and

$$\widehat{s_2} = (\overrightarrow{s_2} \bullet \overrightarrow{u_1}) \cdot \overrightarrow{u_1} = \left(\begin{pmatrix} 0 \\ -2 \end{pmatrix} \bullet \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix} \right) \cdot \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix} =$$

$$= \left[0 + \frac{2}{\sqrt{2}}\right] \cdot \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix} = [\sqrt{2}] \cdot \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix} = \begin{pmatrix} \frac{2}{2} \\ -\frac{2}{2} \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

and

$$\overrightarrow{\varepsilon_2} = \overrightarrow{s_2} - \widehat{s_2} = \begin{pmatrix} 0 \\ -2 \end{pmatrix} - \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$

and

$$\overrightarrow{u_2} = \frac{1}{|\overrightarrow{\varepsilon_2}|} \cdot \overrightarrow{\varepsilon_2} = \frac{1}{\sqrt{(-1)^2 + (-1)^2}} \cdot \begin{pmatrix} -1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} -1 \\ -1 \end{pmatrix} = \begin{pmatrix} -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix}$$

The obtained orthogonal vectors are

$$\overrightarrow{u_1} = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix} \text{ and } \overrightarrow{u_2} = \begin{pmatrix} -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix}$$

Figure 4 presents the vectors $\overrightarrow{s_1}$, $\overrightarrow{s_2}$ and the orthogonalized vectors $\overrightarrow{u_1}$, $\overrightarrow{u_2}$ on the same set of axes.

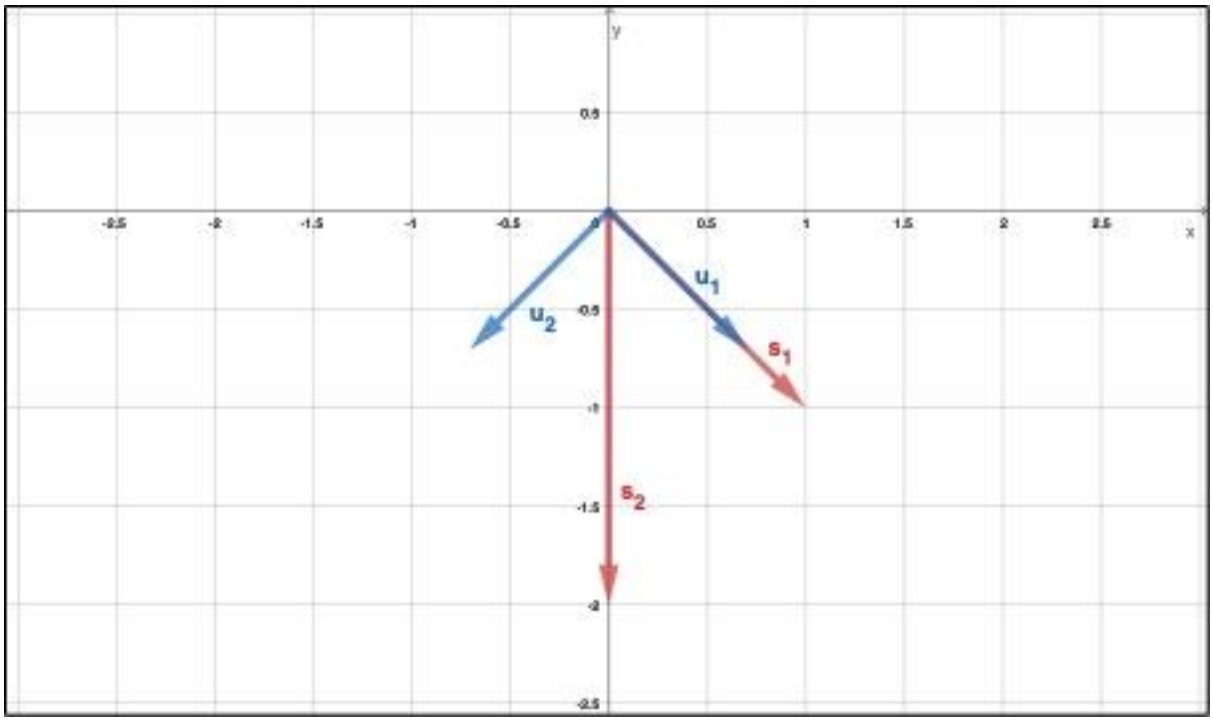

Figure 4: The vectors $\overrightarrow{s_1}, \overrightarrow{s_2}, \overrightarrow{u_1}$, and $\overrightarrow{u_2}$ presented on a plane using GeoGebra

To examine if these vectors really are orthonormal the conditions are checked:

$$|\overrightarrow{u_1}| = \sqrt{\frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2}}{2} + \left(-\frac{\sqrt{2}}{2}\right) \cdot \left(-\frac{\sqrt{2}}{2}\right)} = \sqrt{\frac{1}{2} + \frac{1}{2}} = 1$$

and

$$|\overrightarrow{u_2}| = \sqrt{\left(-\frac{\sqrt{2}}{2}\right) \cdot \left(-\frac{\sqrt{2}}{2}\right) + \left(-\frac{\sqrt{2}}{2}\right) \cdot \left(-\frac{\sqrt{2}}{2}\right)} = \sqrt{\frac{1}{2} + \frac{1}{2}} = 1$$

and

$$\overrightarrow{u_1} \bullet \overrightarrow{u_2} = \frac{\sqrt{2}}{2} \cdot \left(-\frac{\sqrt{2}}{2}\right) + \left(-\frac{\sqrt{2}}{2}\right) \cdot \left(-\frac{\sqrt{2}}{2}\right) = -\frac{1}{2} + \frac{1}{2} = 0$$

Hence both conditions are satisfied, therefore $\overrightarrow{u_1}$ and $\overrightarrow{u_2}$ are orthonormal.

The presented process is the first step in the LLL algorithm. It is explained in the steps in Figure 5.

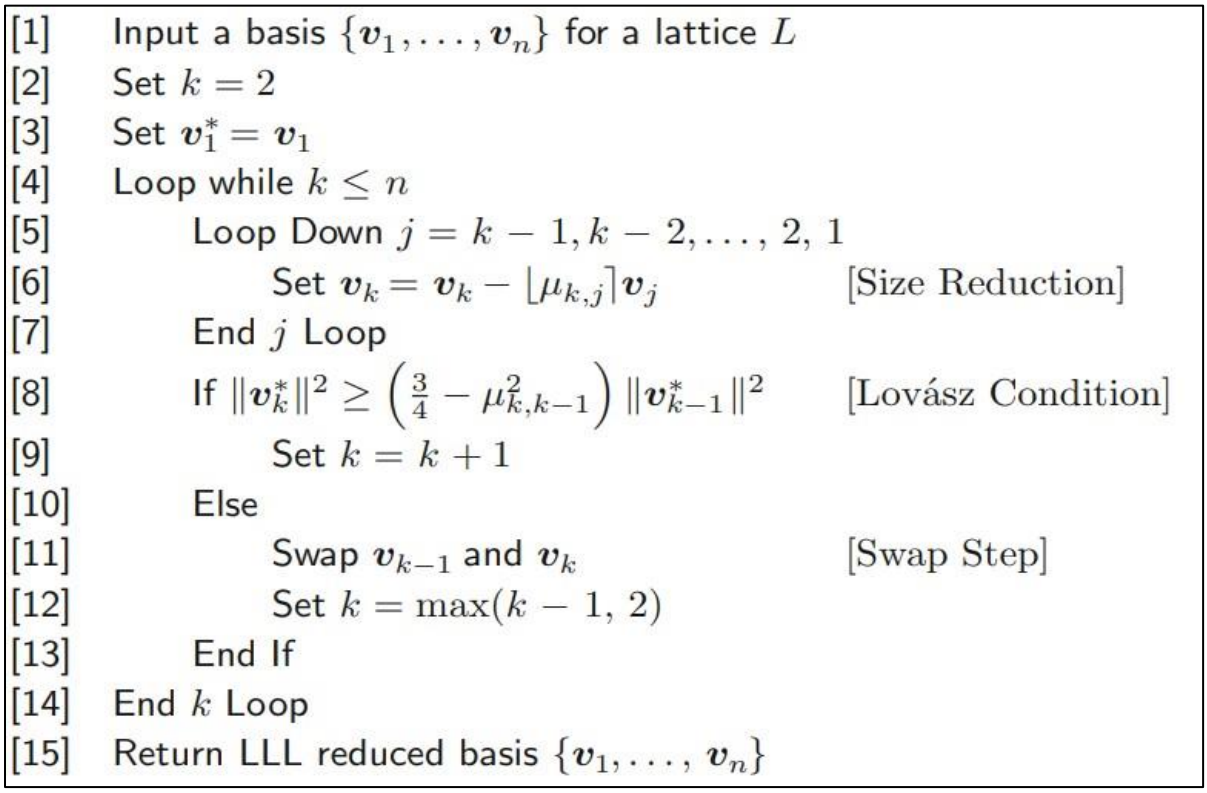

Figure 5: The LLL algorithm (Source No. 7 in Bibliography)

With reference to Figure 5 and the text by Hoffstein, the steps of the LLL algorithm will be explained. At each step, the vectors $\overrightarrow{v_1}^*, \ldots, \overrightarrow{v_k}^*$ form an orthogonal set obtained by applying the Gram-Schmidt process to the current values of $\overrightarrow{v_1}, \ldots, \overrightarrow{v_k}$, where k is an index used to iterate through the basis vectors of the lattice and $\overrightarrow{v_1}^*$ refers to the part of the orthogonalized set. The associated quantity $\mu_{i,j}$ defined as $\frac{\overrightarrow{v_i} \cdot \overrightarrow{v_1}^*}{|\overrightarrow{v_1}^*|^2}$, and is the coefficient used to perform the size reduction step in the algorithm (Hoffstein et al. 2014, 444).

The first step is to input a basis for a lattice, where the vectors are assumed to be linearly independent. Then, the step in line [3] begins the Gram-Schmidt Process discussed throughout pages 21 – 24, starting from the second vector. Each current vector is adjusted by reducing its size as presented in line [6]. This is a part of the [Size Reduction] step. The [Lovász Condition] determines whether vectors are in the correct positions, meaning ordered by their length. This is a crucial step of the algorithm as it ensures that the orthogonalized vectors are short, which leads to efficiently creating a reduced basis. If the condition is satisfied for $k,$ the algorithm checks whether it is satisfied for $k+1$, and so on. If the [Lovász Condition] is not satisfied, it indicates that the vectors are not in the correct order, and the [Swap Step] is performed, meaning that the k^{th} vector is interchanged with the $(k-1)^{th}$ vector. Line [12] is responsible for setting such a value of k, so that the algorithm steps back to check the order of the vectors in case there are further adjustments of the order needed. This process is repeated until all the [Lovász Condition] holds true for all vectors. Once it is satisfied, the loop ends as seen in line [14], and the LLL algorithm outputs a reduced basis for the lattice, which can be seen in line [15].

Attacking the cryptosystem

With reference to the previously encrypted and decrypted messages using the Merkle-Hellman cryptosystem, let us attack the cryptosystem using lattice reduction, through the Lenstra-Lenstra-Lovász algorithm. For the purpose of this paper, we will refer to the attacker as Teresa.

Alexander sends Benjamin a ciphertext block $C_3 = 2348$ encrypted using the public key $T = (57, 95, 456, 627, 173, 403, 749, 645)$. Teresa knows this information allowing her

to solve the matrix equation $TU = C$, where U consists only of 0s and 1s. If U is a solution to $TU = C$, then the block matrix equation

$$JV = \begin{bmatrix} I_{r\times r} & 0_{r\times 1} \\ T_{1\times r} & -C_{1\times 1} \end{bmatrix} \begin{bmatrix} U_{r\times 1} \\ 1_{1\times 1} \end{bmatrix} = \begin{bmatrix} U_{r\times 1} \\ 0_{1\times 1} \end{bmatrix} = W$$

holds, since $JV = W$ is equivalent to $U = U$ and $TU - C = 0$. To ensure that a 0 is produced as the last entry of JV, C occurs with a negative sign in the matrix (Stamp 2005, 203-210). J is the block matrix combining the identity matrix I, which helps to maintain the structure of a lattice and the public key T, along with the ciphertext C, sent by Alexander. V is a matrix that includes U, which is the message sent by Alexander made of 0s and 1s and W is the result of matrix multiplication, one of which contains the original message. Given that the public key is a sequence consisting of 8 elements, Teresa knows that $r = 8$. Hence, she acquires the block matrix equation:

$$JV = \begin{bmatrix} I_{8\times 8} & 0_{8\times 1} \\ T_{1\times 8} & -C_{1\times 1} \end{bmatrix} \begin{bmatrix} U_{8\times 1} \\ 1_{1\times 1} \end{bmatrix} = \begin{bmatrix} U_{8\times 1} \\ 0_{1\times 1} \end{bmatrix} = W$$

and is able to construct the block matrix J.

$$J = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 57 & 95 & 456 & 627 & 173 & 403 & 749 & 645 & -2348 \end{bmatrix}$$

Given that Teresa is able to find a solution V to the block matrix equation $JV = W$ she will consequently find a solution U to the original equation $TU = C$, and hence obtain the bit structure encrypted by Alexander.

To find a solution, Teresa executes the LLL algorithm, with reference to the theoretical framework presented on pages 25 and 26. The LLL algorithm presented in this paper is executed using the mathematical software SageMath. A limitation of this software is that it produces only the final output, and not the steps of the LLL algorithm. In terms of this paper, it means that only the theoretical framework of the LLL algorithm is presented, as well as the input and output of the algorithm. Such a gradual explanation could not fit in the 4000-word limit. By providing J as an input, as seen in Figure 6 in the Appendix, the software outputs the short vectors in the lattice, represented in the matrix J', as follows:

$$J' = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & -2 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -2 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 2 & 0 & -1 \\ -2 & -2 & 0 & 1 & 0 & 0 & 0 & -1 & 1 \\ 1 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 2 & -1 & -1 & 1 & -1 & 0 \end{bmatrix}$$

Now, Teresa has to examine which column has the right form to be a solution to the knapsack problem, that is; the column has to consist of 1's and 0's only, and the digit in the last row must equal 0. This is, because the public key T is a sequence of eight numbers, and the obtained matrix J' has nine rows. The only column that has the right form to be the solution is the third column, so Teresa denotes it:

$$t_1 = (01100111)$$

Teresa obtained t_1, so the final thing left is converting t_1 back from binary. The two other attacks executed can be observed in the Appendix LLL Algorithm, and their outcomes are

$$t_2 = (01101111)$$

and

$$t_3 = (01100100)$$

A problem that Teresa encounters now, is that she does not possess information about the form of the message encrypted by Alexander. She only possesses the binary output and has to guess whether it is encrypted text, or for example a decimal number. Given these outputs Teresa has to try and recover the plaintext. Below, the way to uncover a decimal message is presented.

Teresa decides to check what decimal numbers the binary sequences are equal to, and she obtains:

$$t_1 = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 103$$

and

$$t_2 = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 111$$

and

$$t_3 = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 100$$

Teresa can suspect that the original message was a word. She converts the obtained decimal numbers into letters using the Binary Representations Table (see Appendix) and discovers the following:

$$t_1 = g \text{ and } t_2 = o \text{ and } t_3 = d$$

Therefore, it can be concluded that $T = D$, hence Teresa successfully performed the attack on the Merkle-Hellman cryptosystem, using the LLL algorithm. This demonstrates its effectiveness in attacking the Merkle-Hellman cryptosystem. A limitation of this way of decrypting a message is that it is time-consuming, due to the fact that each part of the message must be obtained individually, which also creates additional room for error. A limitation of the LLL algorithm arises from the fact that it may fail to produce a correct solution, because the lattice might not contain a short vector that corresponds to the solution. Nevertheless, broken in 1981, the Merkle-Hellman cryptosystem was considered insecure, and could not be effectively used.

Conclusion

Throughout this paper the theory behind the Merkle-Hellman system and lattice reduction was presented. The encryption and decryption processes were carried out using the unique properties of superincreasing sequences and provided insight into them, while the lattice reduction was more focused on calculations including vectors and matrices. The most crucial step throughout this paper, is acknowledging the importance of the superincreasing properties of the private key, and the application of

the LLL algorithm, which can be effective, but it does not always yield a solution. This exposes the principles and vulnerabilities of the Merkle-Hellman cryptosystem, shown using an example and creates a better perception of them. By examining the usage of the LLL-algorithm on the Merkle-Hellman cryptosystem, this paper demonstrates vulnerabilities exploited by lattice reduction techniques.

Bibliography

1. *ASCII Table - ASCII Codes, Hex, Decimal, Binary, Html. www.rapidtables.com/code/text/ascii-table.html.*
2. *Cacchiani, Valentina, et al. “Knapsack Problems — an Overview of Recent Advances. Part I: Single Knapsack Problems.” Computers & Operations Research, vol. 143, July 2022, p. 105692. https://doi.org/10.1016/j.cor.2021.105692.*
3. *Calculator Suite - GeoGebra. www.geogebra.org/calculator.*
4. *DG. “Merkle-Hellman Knapsack-based Public Key Cryptosystem - Part 3.” YouTube, 14 Dec. 2020, www.youtube.com/watch?v=8eIGGOw2U3A.*
5. *Galbraith, Steven. Mathematics of Public Key Cryptography. 2nd ed., 2018.*
6. *Hellman, Martin E. “An Overview of Public Key Cryptography: With a public key cryptosystem, the key used to encipher a message can be made public without compromising the secrecy of a different key needed to decipher that message.” IEEE COMMUNICATIONS SOCIETY MAGAZINE, 1978, www-ee.stanford.edu/~hellman/publications/31.pdf. Accessed 10 Oct. 2024.*
7. *Hoffstein, Jeffrey, et al. “An Introduction to Mathematical Cryptography.” Undergraduate texts in mathematics, 2014, https://doi.org/10.1007/978-1-4939-1711-2.*
8. *Lenstra, Arjen K., et al. “Factoring Polynomials With Rational Coefficients.” Mathematische Annalen, vol. 261, no. 4, Dec. 1982, pp. 515–34. https://doi.org/10.1007/bf01457454.*

9. *Liu, Jiayang, et al. “An Improved Attack on the Basic Merkle–Hellman Knapsack Cryptosystem.” IEEE Xplore, Apr. 2019, p. 1. IEEEXplore, ieeexplore.ieee.org/abstract/document/8701428.*
10. *Louis, Frederik. Lattice Reduction Attack on the Merkle-Hellman Cryptosystem. Institut Teknologi Bandung, 2023, informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/Makalah2/Makalah2-Kriptografi-2023%20(25).pdf.*
11. *Margalit, Dan, and Joseph Rabinoff. Linear Independence. 2019, textbooks.math.gatech.edu/ila/linear-independence.html. Accessed 10 Sept. 2024.*
12. *Micciancio, Daniele. Foundations of Lattice Cryptography. 2013, p. 3. www.math.uci.edu/~asilverb/Lattices/Slides/Daniele1uci13-handout.pdf.*
13. *Ortega, Juan. “LATTICE REDUCTION ALGORITHMS.” CSUSB ScholarWorks, scholarworks.lib.csusb.edu/etd/1436.*
14. *Polách, Juraj. Lattice Basis Reduction Using LLL Algorithm with Application to Algorithmic Lattice Problems. Department of Mathematics Uppsala University, Feb. 2022, uu.diva-portal.org/smash/get/diva2:1641300/FULLTEXT01.pdf. Accessed 10 Sept. 2024.*
15. *Shamir, Adi. “A Polynomial-time Algorithm for Breaking the Basic Merkle - Hellman Cryptosystem.” IEEE Transactions on Information Theory, vol. 30, no. 5, Sept. 1984, pp. 699–704. https://doi.org/10.1109/tit.1984.1056964.*
16. *Stamp, Mark. Information Security: Principles and Practices. 2005, www.gbv.de/dms/hebis-darmstadt/toc/133619745.pdf.*
17. *Stamp, Mark, and Richard M. Low. Applied Cryptanalysis. 2007, https://doi.org/10.1002/9780470148778.*

18. *Taboga, Marco. “Gram-Schmidt Process.” StatLect, www.statlect.com/matrix-algebra/Gram-Schmidt-process. Accessed 13 Sept. 2024.*

19. *Towers, Matthew. “Definitions and Matrix Algebra.” UCL, UCL, 2021, www.ucl.ac.uk/~ucahmto/0007_2021/2-1-definitions-and-matrix-algebra.html.*

20. *Extended Euclidean Algorithm Calculator*. www.extendedeuclideanalgorithm.com/calculator.php?mode=2&n=1081&b=19#num.

21. *Use SageMath Online. cocalc.com/features/sage.*

22. *Wikipedia contributors. “Lattice Reduction.” Wikipedia, 22 Jan. 2024, en.wikipedia.org/wiki/Lattice_reduction.*

Appendix

Binary Representations

Dec	Hex	Binary	HTML	Char	Description
97	61	01100001	&#97;	a	
98	62	01100010	&#98;	b	
99	63	01100011	&#99;	c	
100	64	01100100	&#100;	d	
101	65	01100101	&#101;	e	
102	66	01100110	&#102;	f	
103	67	01100111	&#103;	g	
104	68	01101000	&#104;	h	
105	69	01101001	&#105;	i	
106	6A	01101010	&#106;	j	
107	6B	01101011	&#107;	k	
108	6C	01101100	&#108;	l	
109	6D	01101101	&#109;	m	
110	6E	01101110	&#110;	n	
111	6F	01101111	&#111;	o	
112	70	01110000	&#112	p	
113	71	01110001	&#113;	q	
114	72	01110010	&#114;	r	
115	73	01110011	&#115;	s	
116	74	01110100	&#116;	t	
117	75	01110101	&#117;	u	
118	76	01110110	&#118;	v	
119	77	01110111	&#119;	w	
120	78	01111000	&#120;	x	
121	79	01111001	&#121;	y	
122	7A	01111010	&#122;	z	

Figure 1: Binary Representations Table

LLL Algorithm

```
M = Matrix(ZZ, [
    [1, 0, 0, 0, 0, 0, 0, 0, 57],
    [0, 1, 0, 0, 0, 0, 0, 0, 95],
    [0, 0, 1, 0, 0, 0, 0, 0, 456],
    [0, 0, 0, 1, 0, 0, 0, 0, 627],
    [0, 0, 0, 0, 1, 0, 0, 0, 173],
    [0, 0, 0, 0, 0, 1, 0, 0, 403],
    [0, 0, 0, 0, 0, 0, 1, 0, 749],
    [0, 0, 0, 0, 0, 0, 0, 1, 645],
    [0, 0, 0, 0, 0, 0, 0, 0, -2348]
])
M_lll = M.LLL()
M_lll
```

```
[-1  0  0  0 -2  1  0  0  0]
[ 0  0  0  0 -2 -1  1  0  0]
[ 0  1  1  0  0  1  1  1  0]
[ 0  0  1 -1  1  0  0  0  2]
[ 0 -2 -1  0  0  0  0  1 -1]
[-2  1  0 -1  0  0  0  1 -1]
[-1  0  0  2  0  1  1  0  1]
[ 0 -1  2  0 -1  0  0 -1 -1]
[ 1  0 -1 -1  1  0  2 -1  0]
```

Figure 6: LLL Algorithm for 2348

```
M = Matrix(ZZ, [
    [1, 0, 0, 0, 0, 0, 0, 0, 57],
    [0, 1, 0, 0, 0, 0, 0, 0, 95],
    [0, 0, 1, 0, 0, 0, 0, 0, 456],
    [0, 0, 0, 1, 0, 0, 0, 0, 627],
    [0, 0, 0, 0, 1, 0, 0, 0, 173],
    [0, 0, 0, 0, 0, 1, 0, 0, 403],
    [0, 0, 0, 0, 0, 0, 1, 0, 749],
    [0, 0, 0, 0, 0, 0, 0, 1, 645],
    [0, 0, 0, 0, 0, 0, 0, 0, -2521]
])
M_lll = M.LLL()
M_lll
```

```
[-1  0  0  0 -2  1  0  0  0]
[ 0  0  0  0 -2 -1  1  0  0]
[ 0  1  1  0  1  1  1  1  0]
[ 0  0  1 -1  1  0  0  0  2]
[ 0 -2 -1  0  0  0  0  1 -1]
[-2  1  0 -1  0  0  0  1 -1]
[-1  0  0  2  1  1  1  0  1]
[ 0 -1  2  0 -1  0  0 -1 -1]
[ 1  0 -1 -1  1  0  2 -1  0]
```

Figure 7: LLL Algorithm for 2521

```
M = Matrix(ZZ, [
    [1, 0, 0, 0, 0, 0, 0, 0, 57],
    [0, 1, 0, 0, 0, 0, 0, 0, 95],
    [0, 0, 1, 0, 0, 0, 0, 0, 456],
    [0, 0, 0, 1, 0, 0, 0, 0, 627],
    [0, 0, 0, 0, 1, 0, 0, 0, 173],
    [0, 0, 0, 0, 0, 1, 0, 0, 403],
    [0, 0, 0, 0, 0, 0, 1, 0, 749],
    [0, 0, 0, 0, 0, 0, 0, 1, 645],
    [0, 0, 0, 0, 0, 0, 0, 0, -954]
])
M_lll = M.LLL()
M_lll
```

```
[ 0  1  1  0  0  1  0  0  0]
[ 0 -1  0  0  0  1  0  1 -1]
[ 1  1  1  0  0 -1  1  0  0]
[-1  0  0  0 -2  1  0  0  0]
[ 0  0  1 -1  1  0  0  0  2]
[ 0 -1  0  2  0  0  1  0  0]
[-2  0  0  1  0  0  1  1 -1]
[ 0 -1  2  0 -1  0  0 -1 -1]
[ 1  0 -1 -1  1  0  2 -1  0]
```

Figure 8: LLL Algorithm for 954