AMS595 Group Project

Using MATLAB to Implement Encryption, Decryption and Cracking for Knapsack Cryptosystem

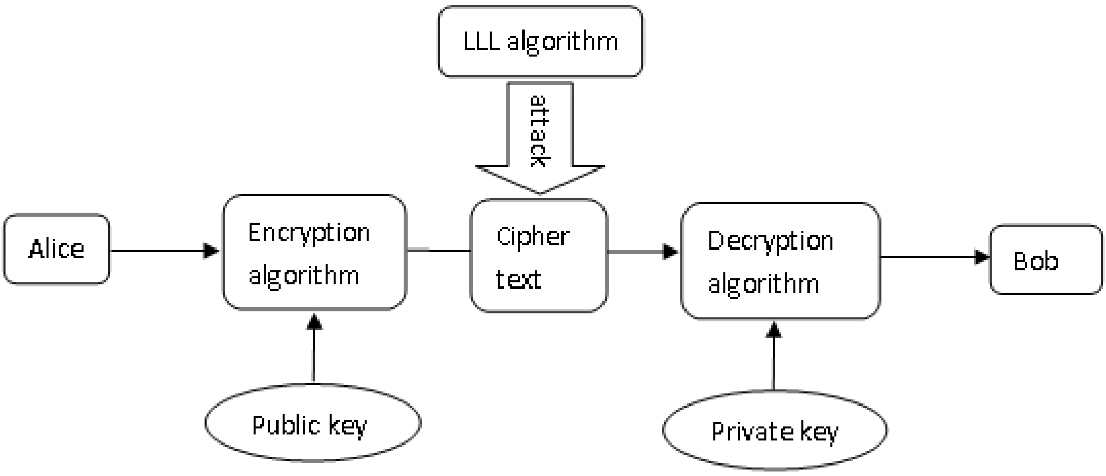
Group members: Syamala Balasubramanian,

Zhaoyan Liu, Chicheng Zhang

Contents

1. Introduction

The Merkle–Hellman knapsack cryptosystem was one of the earliest public key cryptosystems invented by Ralph Merkle and Martin Hellman in 1978. It is an asymmetric-key cryptosystem, meaning that two keys are required for communication: a public key and a private key. Furthermore, the public key can be known widely and used only for encryption, and the private key is known only by the owner and used only for decryption.



As the name implies, knapsack cryptosystem is based on a math problem called knapsack problem. The problem is as follows: given a set of non-zero natural numbers A and a number b, find a subset of A which sums to b. In general, this problem is known to be NP-complete. However, if the set of numbers (called the knapsack) is superincreasing, meaning that each element of the set is greater than the sum of all the numbers in the set lesser than it, the problem is "easy" and solvable in polynomial time. In Merkle-Hellman, the keys are two knapsacks. The public key is a 'hard' knapsack A, and the private key is an 'easy', or superincreasing, knapsack B.

However, without knowing private key, this knapsack cryptosystem could be cracked by Lenstra–Lenstra–Lovász (LLL) lattice basis reduction algorithm invented by Arjen Lenstra, Hendrik Lenstra and László Lovász in 1982.

In this project, our group is going to learn the four algorithms used in key generation, encryption, decryption and cracking, and then implement all the four processes on MATLAB. The final objectives are that by using our programs, the receiver can generate his/her public key and private key, the sender can use public key to encrypt text, the receiver can use private key to decrypt cipher text, and the eavesdropper can also decrypt the cipher text but without private key.

2. Techniques and Algorithms

2.1 Knapsack problem

Definition 2.1.1. given a set of non-zero natural numbers

$$\mathrm{B} = \{b_1,\ b_2, \dots,\ b_n\}$$

and positive integer b. ***Knapsack problem*** is considering whether there is a subset of B so that

$$\sum_{i \in I} b_i = b, \qquad I \subseteq \{1,2,\dots,n\}. \tag{2.1.1}$$

Now, we rewrite knapsack problem as following. We define that

$$x_i = \begin{cases} 1 & if\ i \in I \\ 0 & if\ i \notin I \end{cases} \qquad (i = 1,2,\dots,n).$$

Further, equation (2.1) can be written as

$$\sum_{i=1}^{n} x_i b_i = b. \tag{2.1.2}$$

Definition 2.1.2. Sequence $b_1,\ b_2, \dots,\ b_n$ is called ***superincreasing sequence*** if it satisfies

$$b_i < \sum_{j=1}^{i-1} b_j \qquad (i = 1,2,\dots,n). \tag{2.1.3}$$

That means each element of the set is greater than the sum of all the numbers in the set lesser than it.

For any superincreasing sequence, knapsack problem can be easily solved by recursion from back to front. Since

$$b_n > b_1 + b_2 + \cdots + b_{n-1}, \tag{2.1.4}$$

We can set

$$x_i = \begin{cases} 1 & if\ b > b_1 + b_2 + \cdots + b_{n-1} \\ 0 & if\ b \le b_1 + b_2 + \cdots + b_{n-1} \end{cases}. \tag{2.1.5}$$

More simply,

$$x_n = 1 \quad \Leftrightarrow \quad b > \sum_{j=1}^{n-1} b_j. \tag{2.1.6}$$

Now, we can remove b_n from the sequence and transfer this knapsack problem to a new one, which means that the superincreasing sequence becomes to $b_1, b_2, \dots, b_{n-1}$, and the sum is $\mathrm{b} - x_n b_n$:

$$\sum_{i=1}^{n-1} x_i b_i = b - x_n b_n. \tag{2.1.7}$$

Then, we have

$$x_{n-1} = 1 \quad \Leftrightarrow \quad b - x_n b_n > \sum_{j=1}^{n-2} b_j. \tag{2.1.8}$$

By repeating the process above, finally we can get $(\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{n}) \in \{0,1\}^n$.

2.2 Modular arithmetic

In mathematics, modular arithmetic is a system of arithmetic for integers, where numbers "wrap around" upon reaching a certain value—the modulus (plural moduli).

Given an integer n, all integers can be classified by whether it can be exactly divide by n. For

those that cannot be divided exactly by n, we can classify them depending on the remainder.
Definition 2.2.1. Suppose a is an integer, $n > 1$ is a positive integer. $\boldsymbol{a\ (mod\ n)}$ is the remainder *r* of *a* divided by *n*, which means

$$r = a\ (mod\ n) = a - \lfloor a/n \rfloor n.$$

Definition 2.2.2. For a positive integer *n*, two numbers *a* and *b* are said to be ***congruent modulo n***, if their difference *a* − *b* is an integer multiple of *n* (that is, if there is an integer *k* such that *a* − *b* = *kn*). This congruence relation is typically considered when a and b are integers, and is denoted

$$a \equiv b(mod\ n).$$

The congruence relation satisfies all the conditions of an equivalence relation:

- Reflexivity: $a \equiv a(mod\ n)$
- Symmetry: $a \equiv b(mod\ n)$ if and only if $b \equiv a(mod\ n)$
- Transitivity: If $a \equiv b(mod\ n)$ and $b \equiv c(mod\ n)$, then $a \equiv c(mod\ n)$.

Definition 2.2.3. The modular multiplicative inverse is defined by the following rules:

- Existence: there exists an integer denoted a^{-1} such that $aa^{-1} \equiv 1(mod\ n)$ if and only if *a* is coprime with *n*. This integer a^{-1} is called a ***modular multiplicative inverse*** of *a* modulo *n*.
- If $a \equiv b(mod\ n)$ and a^{-1} exists, then $a^{-1} \equiv b^{-1}(mod\ n)$ (compatibility with multiplicative inverse, and, if *a* = *b*, uniqueness modulo n}})
- If $ax \equiv b(mod\ n)$ and a is coprime to n, the solution to this linear congruence is given by $x \equiv a^{-1}b(mod\ n)$.

2.3 Lattice reduction

Lattice reduction is a powerful technique which can be used to solve many different types of combinatorial problems. Consider, for example, the vectors

$$c_0 = \binom{-1}{1} \quad and \quad c_1 = \binom{1}{2}.$$

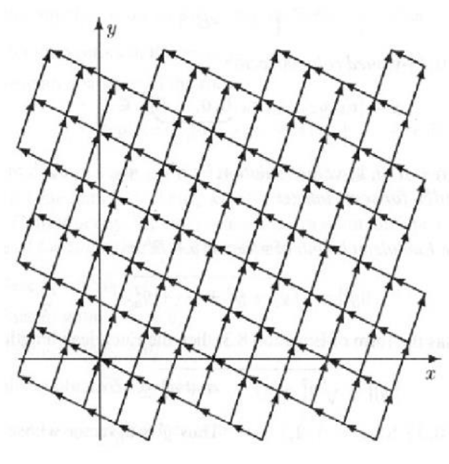


Figure 1: A lattice in the plane.

Since c_0 and c_1 are linearly independent, any point in the plane can be uniquely represented by $\alpha_0 c_0 + \alpha_1 c_1$, where α_0 and α_1 are real numbers. If we restrict the coefficients to integers, that is, we require that α_0 and α_1 are integers, then we obtain a lattice consisting of discrete points in the plane. Figure 1 illustrates the lattice spanned by c_0 and c_1. In general, a ***lattice*** L is the set of all linear combinations of a set of column vectors c_i with integer coefficients.

Given an $m\times n$ matrix A and an $m\times 1$ matrix B, suppose we want to find a solution U to the matrix equation $AU = B$, with the restriction that U consists entirely of 0s and 1s. If U is a solution to $AU = B$, then the block matrix equation

$$MV = \begin{bmatrix} I_{n\times n} & 0_{n\times 1} \\ A_{m\times n} & -B_{m\times 1} \end{bmatrix} \begin{bmatrix} U_{n\times 1} \\ 1_{1\times 1} \end{bmatrix} = \begin{bmatrix} U_{n\times 1} \\ 0_{m\times 1} \end{bmatrix} = W \tag{2.3.1}$$

holds true, since $MV = W$ is equivalent to $U = U$ and $AU - B = 0$. Consequently, finding a solution V to the block matrix equation $MV = W$ is equivalent to finding a solution U to the original matrix equation $AU = B$. Note that the columns of M are linearly independent, since the $n\times n$ identity matrix appears in the upper left and the final column begins with n zeros.

Let $c_0, c_1, \ldots, c_n$ be the $n + 1$ columns of the matrix M in (2.3.1) and let $v_0, v_1, \ldots, v_n$ be the elements of V. Then

$$W = v_0 c_0 + v_1 c_1 + \cdots + v_n c_n. \tag{2.3.2}$$

We have $MV = W$, where

$$W = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \binom{U}{\mathbf{0}} \tag{2.3.3}$$

and we want to determine U. Instead of solving linear equations to obtain V, we will find U by determining W. Note that because of (2.3.2), W is in the lattice L, spanned by the columns of M.

The Euclidean length of a vector $Y = [y_0, y_1, \ldots, y_{n+m-1}]^T$ is

$$||Y|| = \sqrt{y_0^2 + y_1^2 + \cdots + y_{n+m-1}^2}.$$

However, the length of a vector W in (2.3.3) is

$$||W|| = \sqrt{u_0^2 + u_1^2 + \cdots + u_{n-1}^2} \le \sqrt{n},$$

which is much "shorter" than a typical vector in L. Furthermore, W has a very special form, since its first n entries consist of 0s and 1s with its last m entries being all 0.

2.4 LLL algorithm

LLL Algorithm provides an efficient method to find short vectors in a lattice. In Table 1, we give an outline of their algorithm in pseudo-code, where $GS(M)$ refers to the Gram–Schmidt process, which returns an orthonormal basis for the subspace spanned by the columns of M. The Gram–Schmidt process appears in Table 2. Note that a small number of lines of pseudo-code suffices to specify the entire LLL Algorithm.

Table 1: LLL Algorithm

```
// find short vectors in the lattice spanned
// by the columns of M = (b_0, b_1, ..., b_n)
repeat
    (X, Y) = GS(M)
    for j = 1 to n
        for i = j − 1 to 0
            if |y_ij| > 1/2 then
                b_j = b_j − ⌊y_ij + 1/2⌋ b_i
            end if
        next i
    next j
    (X, Y) = GS(M)
    for j = 0 to n − 1
        if ||x_{j+1} + y_{j,j+1} x_j||^2 < (3/4)||x_j||^2 then
            swap(b_j, b_{j+1})
            goto abc
        end if
    next j
    return(M)
abc: continue
forever
```

Table 2: Gram–Schmidt Process

```
// Gram–Schmidt M = (b_0, b_1, ..., b_n)
GS(M)
    x_0 = b_0
    for j = 1 to n
        x_j = b_j
        for i = 0 to j − 1
            y_ij = (x_i · b_j)/||x_i||^2
            x_j = x_j − y_ij x_i
        next i
    next j
    return(X, Y)
end GS
```

2.5 Algorithms of processes in knapsack cryptosystem

There are four main processes in knapsack cryptosystem: key generation, encryption, decryption and cracking (by LLL Algorithm).

2.5.1 Key generation

(i) According to the Knapsack cryptosystem, the first step in encryption is to generate a public key by given private key. The private key is supposed to be a super-increasing sequence $a_1, a_2, \dots, a_n$ satisfying that $a_i > \sum_{j=1}^{i-1} a_j$. Here we set n equals to 7 because the length of ASCII binary code of alphabets are 7 digits.

(ii) Set multiplier m and modular w. Notice that m, w are integers and satisfy that

$$m > 2a_n, \ \gcd(m, w) = 1.$$

(iii) Then the public key can be calculated as

$$b_i \equiv wa_i (\bmod m), \ 0 \le b_i < m.$$

As the public key is not a super-increasing sequence, it is not easy to solve the Knapsack problem by using the public key.

2.5.2 Encryption

(i) After gaining the public key, transfer the text that need to be encrypted into binary digits.
For example, 'A' can be transferred as '1000001' in binary digits. Take x_{ij} as the jth digit of the ith letter in text.

(ii) Then the ith letter can be encrypted as follows,

$$s_i = b_1 x_{i1} + b_2 x_{i2} + \cdots + b_7 x_{i7}.$$

Subsequently, the cypher text will be sent to receiver who already has the private key. During this situation, the cypher text and public key maybe captured by other code breakers.

2.5.3 Decryption

(i) After the receiver gains the cypher text, the main purpose is to transfer the problem into a solvable Knapsack problem. First step is to calculate w^{-1}, the modular inverse of w with respect to the modulus m by extended Euclidean algorithm.

(ii) For every cypher text s_i, we have

$$\begin{aligned} t_i &= w^{-1} s_i (mod\ m) \\ &= w^{-1} \sum_{j=1}^{7} b_i x_{ij}\ (mod\ m) \\ &= w^{-1} \sum_{j=1}^{7} wa_i (mod\ m) x_{ij}\ (mod\ m) \end{aligned}$$

$$= \sum_{j=1}^{7} a_i x_{ij} \, (mod\ m) = \sum_{j=1}^{7} a_i x_{ij}$$

(iii) Knowing the private key a_1, a_2, Λ, a_n, the Knapsack problem $t_i = \sum_{j=1}^{7} a_i x_{ij}$ can be easily solved by means referred above. Thus, every cypher text can be transferred in to binary digits $x_{i1}x_{i2}x_{i3}x_{i4}x_{i5}x_{i6}x_{i7}$. Finally, the binary digits can be transferred back to characters, which is the original text.

2.5.4 Cracking

(i) As for the code breakers who capture the cypher text during transmission. The LLL reduction algorithm can be used to decrypt the cypher text without private key. Firstly, by given cypher text s and public key a, we generate a matrix

$$M_i = \begin{bmatrix} I_{7\times7} & 0_{7\times1} \\ a_{1\times7} & -s_i \end{bmatrix}.$$

It's obvious that the column vectors in M_i are linear independent with each other. Thus, they combine a basis of a lattice.

(ii) Because the basis is relatively long, by using LLL algorithm can reduce the length of basis. The LLL Algorithm outputs a matrix M_i', consisting of short vectors in the lattice spanned by the columns of the matrix M_i.

(iii) Find a column in matrix M_i' that only consists of 1 and 0 or -1 and 0 (LLL algorithm probably generates an inverse reduced basis). The first seven digits are the binary code of the ith alphabet. Finally, convert the binary code into character, we gain the original text.

3. Experimental results

In our experiment, we use the text 'Math', private key $a = [1\ 2\ 4\ 9\ 19\ 39\ 80]$, multiplier $w = 13$, modular $m = 199$ to undergo the total processes.

- The result of public key generation shows as follows:

```
Please input private key:[1 2 4 9 19 39 80]
Enter the value of w: 13
Enter the value of m(m>w & gcd(w,m)==1): 199

*******************************************
The public key is:
    13    26    52    117    48    109    45
```

Therefore, the public key is $b = [13\ 26\ 52\ 117\ 48\ 109\ 45]$.

- The result of encryption with public shows as follows:

```
Please input text you want to encrypt:'Math'
****************************************
The cipher text is:
223 84 139 156
```

Therefore, the cipher text is $[223\ 84\ 139\ 156]$.

- The result of decryption with private key shows as follows:

```
Please input w:13
Please input m:199
Please input private key:[1 2 4 9 19 39 80]
Please input cypher text:[223 84 139 156]
****************************************
The decrypted text is:
 Math >>
```

- The result of cracking by LLL algorithm shows as follows:

```
****************************************
Please input the public key:[13 26 52 117 48 109 45]
Please input the cypher text:[223 84 139 156]
```

```
The reduced matrix is:
   -2    0   -1    1    1    1    0    0
    1   -2    0    0    0    0    0    1
    0    1   -2    1    0   -1    0    0
    0    0    1    0    1   -1   -1   -1
    0    0    0    0    1    1    1    0
    0    0    0   -1    0    1   -1    0
    0    0    0    1    1    0   -1    2
    0    0    0    1    0    1    0   -1

The LLL decryption text is: M
```

```
The reduced matrix is:
   -1   -1   -1    0    0    0    0    1
    0    1   -1    1   -1   -1    0    0
    1   -1    0    0    0   -1    0    0
    0    0    0    0   -1    1   -1    0
    0    0    0    0    0    0    1    2
    0    0    0   -1   -1    0    1   -1
    1   -1   -1    0    0    1    1    0
    0    0    0    1    0    0    1    0

The LLL decryption text is: a
```

```
The reduced matrix is:
    0   -2    0   -1    0    0    1    0
   -1    1   -1   -1    0    0    0    0
    0    0    1   -1   -1   -1   -1    0
    1    0   -1    0   -1    1   -1    0
    1    0   -1   -1    0    0    1    1
    0    0    0    0   -1   -1    1    0
    0    0    0    0    0    1    0    2
    0    0    0    0    0    1    1   -1

The LLL decryption text is: t
```

```
The reduced matrix is:
   -1   -1   -1    0    0    0    1    0
    0    1   -1   -1    1    0    0    1
    1   -1    0   -1    0    0    0    0
    1   -1   -1    2   -1    0    0   -1
    0    0    0    0    0    1    2    0
    0    0    0    0   -1    1   -1    0
    0    0    0    0    1    0    0    2
    0    0    0    0    1    1    0   -1

The LLL decryption text is: h >>
```

The highlight column in each matrix is the binary code of each ciphertext. Combine them, we can get the cracked text, the ‘Math’.

4. Conclusion

4.1 Individual contributions

- Syamala Balasubramanian:
  Implement encryption on MATLAB; write introduction and conclusion parts of report.

- Zhaoyan Liu:
  Implement key generation, decryption and applying LLL algorithm to crack knapsack cryptosystem on MATLAB; do the presentation; write introduction, techniques, and conclusion parts of report.

- Chicheng Zhang:
  Implement key generation, encryption, and applying LLL algorithm to crack knapsack cryptosystem on MATLAB; do the presentation; write algorithms, experimental results, and conclusion part of report.

4.2 Advantages and disadvantages

In our work, the total processes are efficient because we defined as many as functions to shorten the main code. Also, we made a clear structure and detailed comments to make the codes easy to understand and convenient to operate. And we interpreted the binary code as seven digits, which includes not only alphabets but also other symbols, and totally 128 American characters.

As for the weakness of code and algorithm, although in practice, the lattice reduction attack is highly effective against the original Merkle– Hellman knapsack, the LLL Algorithm will not always produce the desired vector and therefore, the attack is not always successful. That means the output reduced matrix sometimes doesn’t have a column satisfies the condition so the cracking process fails when using some specific public key. And the codes break down when this situation happens. We will try to figure this out in the future.

4.3 Future work

In the future work, we will consider using eight or more digits to cover the total processes, which can carry more complex information in the text like Chinese characters. Moreover, we will train the code into a practical program or software for users to encrypt, decrypt and crack the text.

5. Reference

- https://en.wikipedia.org/wiki/Merkle%E2%80%93Hellman_knapsack_cryptosystem
- https://en.wikipedia.org/wiki/Modular_arithmetic
- https://www.mphasis.com/content/dam/mphasis-com/global/en/services/applicationservices/data-engineering/Whitepaper_Cryptography%20Enlightening%20its%20importance%20on%20Network%20Security_10May2017.pdf
- https://brilliant.org/wiki/public-key-cryptography/
- R. C. Merkle, M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. IEEE Transactions on Information Theory 24(1978) 525–530.
- Shamir. A polynomial-time algorithm for breaking the basic MerkleHellman cryptosystem. IEEE Transactions on Information Theory 30(1984) 699–704.

APPENDIX

Table of binary code for alphabet:

Character	A	B	C	D	E	F	G	H	I
Binary code	1000001	1000010	1000011	1000100	1000101	1000110	1000111	1001000	1001001
Character	J	K	L	M	N	O	P	Q	R
Binary code	1001010	1001011	1001100	1001101	1001110	1001111	1010000	1010001	1010010
Character	S	T	U	V	W	X	Y	Z	
Binary code	1010011	1010100	1010101	1010110	1010111	1011000	1011001	1011010	

Character	a	b	c	d	e	f	g	h	i
Binary code	1100001	1100010	1100011	1100100	1100101	1100110	1100111	1101000	1101001
Character	j	k	l	m	n	o	p	q	r
Binary code	1101010	1101011	1101100	1101101	1101110	1101111	1110000	1110001	1110010
Character	s	t	u	v	w	x	y	z	
Binary code	1110011	1110100	1110101	1110110	1110111	1111000	1111001	1111010	