Knapsack Key-Generation System using JavaScript on Web

Shinsuke Hamasho, Yasuyuki Murakami

Abstract

Although some studies have been conducted in knapsack cryptosystems, it is not necessarily easy to generate keys in public-key cryptosystem. There are few open, scalability and easy to use systems. For web usage, programs written in JavaScript have high transparency and scalability, and are very suitable for cryptosystems. This paper provides the necessary systems to generate keys, encrypt and decrypt in knapsack cryptosystem, which can be applied to most knapsack PKC by simply defining some functions including generating secret keys, translating into public keys and others. This paper also provides two examples, Merkle-Hellman and shifted odd knapsack PKC.

Keywords: Knapsack Public-key Cryptosystem, Key-generation System, JavaScript

1. Introduction

Many public-key cryptosystems (PKC) have been proposed[1], [2], [3]. However, it is not necessarily an easy task to construct PKC based communication system. Although some software make it useful, there are some problems for many users. For instance if one wants to check the source code in programs, package software don’t allow the user to see it. Additionally, most software’s have only poor scalability. If one creates a new knapsack cryptosystem, the software must be redesigned.

JavaScript has two great advantages, safety and transparency. When using JavaScript to encipher, all processes are executed in the web browser, then no encrypted information circulates on the Internet. This character guarantees the safety of the cryptosystem. Also, a programming code written in JavaScript can be seen by anyone. All users will be able to use the system accurately due to this transparency.

The first knapsack scheme was proposed by Merkle and Hellman, Merkle-Hellman knapsack PKC (MH knapsack PKC)[2]. It is based on the difficulty of the subset sum problem (SSP). Some algorithms for solving SSP have been proposed[4], [5], [6], [7], when the density is low. However, a polynomial-time algorithm for solving general SSP has not been proposed yet. It is proved that quantum computers solve factorization problem, which is the base of RSA PKC, most used cryptosystem on the Internet[8]. On the other hand SSP is expected not to be solved by even quantum computers. Therefore we expect that researches for this cryptosystem must become active as the study of quantum computer progressed. However, there are few systems able to generate public keys, publish it and decrypt a ciphertext.

In this paper, we implement an useful system to use knapsack PKC which has good operability and high scalability. In prospect of the development of active researches, this system is designed in order to enable users to implement a new kind of knapsack PKC easily.

This research will allow everyone to use and implement knapsack PKC.

2. Preliminary

In this section, we show the algorithms for generating public key, encrypt and decrypt for basic knapsack PKC. We also show two examples, MH knapsack PKC and Shifted-Odd knapsack PKC.

2.1. Knapsack PKC

Knapsack PKC is a cryptosystem based on the difficulty of SSP[2], [3]. SSP is expected to be one of the problems that are difficult to solve even by quantum computers[8]. SSP is to find the solution \((x_1, \ldots, x_n) \in \mathbb{Z}_0, 1^m\) such as
for given positive integer set \(a_1, \ldots, a_n\) and a subset sum \(C\). This problem is known to be \(NP\)-hard. The public key cryptosystem using SSP is usually referred to as the knapsack PKC.

The algorithm of designing public key, encryption and decryption for basic knapsack PKC is shown in Algorithm 1, 2 and 3[9].

Let \(\sigma\) be a randomly chosen element from the permutation group of \(\{1, 2, \ldots, n\}\).

Let the plaintext message be represented by \(m = (m_1, \ldots, m_n) \in \{0, 1\}\). Let the ciphertext be represented by \(C \in \mathbb{Z}\).

### 2.2. MH knapsack PKC

The first knapsack PKC which uses the superincreasing sequence as the trapdoor was proposed in [2]. In this subsection and the next subsection, consider two examples of this cryptosystem. The first is about Merkle-Hellman knapsack PKC.

A super increasing sequence \(s_i\) is defined as follows:

\[
s_i > \sum_{j=1}^{i-1} s_j \quad (1 \leq i \leq n)
\]

It should be noted that it is very easy to solve the subset sum problem using a superincreasing sequence. For instance, let \(x = (1, 2, 5, 10, 22, 43, 87, 178)\) and the subset sum \(C = 145\), you can immediately find 178 cannot be included. 87 and 43 are included and the then the remainder is 15. Finally you will easily find that the remaining elements are 10 and 5. The algorithm for solving a superincreasing SSP are shown in Algorithm 4.

It is so simple that this system cannot be used for cryptosystems. Therefore some changes must be made. The algorithm for transforming MH knapsack PKC from a superincreasing sequence into a random-like sequence is shown in Algorithm 5 and the algorithm for solving MH knapsack PKC is also shown in Algorithm 6.

This cryptosystem has still broken in [10], [11], [12]
2.3. Shifted-odd knapsack PKC

The knapsack PKC using a shifted-odd sequence as the trapdoor was proposed in [3].

A shifted-odd sequence \( s_i \) is defined as \( s_i = m 2^{i-1} - 1 \) (1 ≤ \( i \) ≤ \( n \)) where \( m \) is odd. It should be noted that all \( |s_i| \) are similar values. The SSP of this sequence is also very easy to solve. The algorithm is the following.

The transforming method is the same as in MH knapsack PKC. The algorithms for generating public key and decrypting are shown in Algorithms 8 and 9.

The transforming method is the same as in MH knapsack PKC. The algorithms for generating public key and decrypting are shown in Algorithms 8 and 9.
3. Implementation

JavaScript is an object-orientation language. In this language functions can have functions as arguments. Because knapsack PKC is usually designed by making sequence methods, transforming function and all, this property are very useful.

3.1. Division of methods

From Algorithms 1, 2 and 3, consider the common system of knapsack PKC.

3.1.1. Generation for knapsack public key

As shown in Algorithm 1 basic knapsack public key generator demands following arguments:

- The number of the sending bit length;
- Method to generate a secret sequence;
- Function to transform the secret sequence into a public sequence;
- Function to permute the public sequence;

Therefore, the object to design a public-key can be defined with the following functions.

\[ \text{mkSeq, conv, perm} \]

\( \text{mkSeq} \) is the method to create a secret sequence, which SSP can be solved easily. \( \text{conv} \) is the function to transform the sequence designed by \( \text{mkSeq} \) into a random-appearance sequence. If the sequence is perfectly random, it’s an NP-complete problem. \( \text{perm} \) is the method to permute the sequences. This makes knapsack PKC stronger. These three functions are different in variant knapsack cryptosystems, and most knapsack PKC can be defined by these functions.

3.1.2. Encryption

When considering knapsack PKC, plaintext is traditionally treated as binary number. Encryption is executed as

\[ C = \sum_{i=1}^{n} a_i b \]

Where \( C \) is the ciphertext, \( \langle a_i \rangle \) is a public key and \( \langle b_i \rangle \) is i digit of binary plaintext.

3.1.3. Decryption.

As shown in Algorithm 3, decryption is executed with following method and arguments.

- Secret key;
- Function to transform cipher text;
- Method to solve SSP with specific sequence;
- Function to permute sequence;

Therefore, decryption can be defined with following functions.

\[ SK, \text{convC, solve, perm} \]
SK is the secret sequence. convC is the function to transform the cipher text C into intermediate plain-text M. solve is the method to solve a specific SSP. perm is the method to permute the sequences. When generating public-key, public sequence was permuted. Therefore solution of solve must be corrected with the permutation used in generating public key.

3.2. Definition of Knapsack PKC Object

In this section we define the object to treat knapsack PKC. This object includes all properties and methods to treat knapsack PKC. Once an instance is generated, you can refer all knapsack PKC properties like secret sequences and execute some methods like displaying public sequence, encryption and decryption.

3.2.1. Declaration Instance.

When calling the knapsack PKC object, following methods are executed and properties are defined.

Algorithm 10 Generate the instance of knapsack PKC object

1. Input n ← plaintext length, mkseq, conv, perm
2. Generate secret key
3. Generate public key

3.2.2. Properties.

Show the necessary properties.
- Number of knapsack sequence;
- Secret sequence;
- Public sequence;
- Secret modulus;
- Secret multiplier;

3.2.3. Methods.

Show the necessary methods.
- Encrypt;
- Decrypt;

While encryption method is the same in various knapsack cryptosystems, different systems require the different decryption way.

3.2.4. Usage.

When using Merkle-Hellman or Shifted-odd knapsack PKC, you just declare the instance and execute its methods. For instance you want to use MH knapsack PKC with 16 bit length and random permutation, all you have to do is only following program.

1) mh.pkc = newknapsackPKC(16, geneSI, solveSI, perm);
2) C = mh.pkc.encrypt(message);
3) M = mh.pkc.decrypt(C);

Where geneSI is the function to generate superincreasing sequence, solveSI is the function to solve superincreasing subset sum problem and perm is the permutation function. By this program you execute all knapsack PKC’s processes, generation of public key, encryption and decryption. These object and functions can be get in my web page [14].

Finally show the image file of our web page in Fig1
4. Conclusion

In this paper, we have implemented the system to use knapsack PKC easily. This paper will enable any users to use MH or shifted-odd knapsack PKC. All they have to do is call some functions we have defined in this paper. In addition one can easily implement a newly suggested cipher. This system is completely transparent, any user can check the source code at any time. And as shown in the program, the information is properly encrypted before traveling across the Internet.

We have released the library in the web site. If you find some mistakes or conceive some improvements, please send those ideas to my email address.

What we have to do is improve of this program. For instance we will apply this system for other cryptosystems. Small elements must also be upgraded.

We hope that this system will help any people use cryptosystem safely and easily and researchers investigate knapsack PKC.

Acknowledgement

We are very thankful to Leemon Baird for his public domain JavaScript library BigInt.js.

References


[13] Leemon Baird, "Big Integer Library v. 5.4," http://leemon.com/crypto/BigInt.js,


Knapsack Key-Generation System using JavaScript on Web
Shinsuke Hamasho, Yasuyuki Murakami
Figure 1. Page Image