A Knapsack-type Public Key Cryptosystem based on Gaussian Integers

Klaus Huber, FZ 123a
Deutsche Telekom AG
Forschungs- und Technologiezentrum
P.O.Box 10 00 03
64276 Darmstadt
Germany

Abstract

In this contribution a new knapsack-type public key cryptosystem is proposed.

Index Terms — Public Key Cryptosystem, Knapsack-cryptosystems, Sum of two Squares, Gaussian integers, RLL-codes.

1 Introduction

Although knapsack-type public key cryptosystems are considered with suspicion in the crypto – community, there is no reason to stop consideration of such systems since the underlying subset sum problem belongs to a class of difficult problems. Their main advantages, speed and ease of implementation still make knapsack cryptosystems attractive for implementation. In spite of some dramatic successes in the cryptanalysis of knapsack-type schemes (see [2]), there are examples of knapsacks which are still considered infeasible to break, e.g. the Chor-Rivest knapsack [4]. In this correspondence we propose another public key cryptosystem based on the knapsack problem. The method uses the representation of a large number n as sum of two squares to transform an easy (insecure) knapsack into a 'hopefully' secure knapsack.

2 The Basic Scheme

Let n be a large integer which can be represented as sum of two squares. For example we may select n to be the product of two large primes p and q, i.e. $n = p \cdot q$ with $p \equiv 1 \bmod 4$ and $q \equiv 1 \bmod 4$. Such n can easily be represented as sum of two squares if the factors p and q are known (see e.g. [7] and the appendix). Hence we have

$$n = a^2 + b^2 = \pi \cdot \pi^* \,, \tag{1}$$

where $\pi = a + ib$, and π^* is the conjugate complex number of π. We assume $a > b > 0$. The designer now selects values x_j for an easy knapsack. An easy knapsack is a knapsack which can be 'unpacked' easily. A well-known example of an easy knapsack is the superincreasing knapsack (see [6]), represented by the vector $\mathbf{x} = (x_0, x_1, \ldots, x_{L-1})$. The superincreasing condition $x_k > \sum_{j=0}^{k-1} x_j$ ensures easy decryption. For reasons given later on we do not recommend

using a superincreasing knapsack, but we will use a superincreasing knapsack for illustration in example 1. We further impose

$$\sum_{l=0}^{L-1} x_l < \frac{a-b-1}{2}\,, \tag{2}$$

for reasons of decryption which are explained in the appendix. Let y_j, $j = 0, 1, \ldots, L-1$ be positive random integers which also fulfil

$$\sum_{l=0}^{L-1} y_l < \frac{a-b-1}{2}\,. \tag{3}$$

By the Euclidean algorithm for Gaussian integers we can compute u and v such that

$$1 = u \cdot \pi + v \cdot \pi^*\,. \tag{4}$$

Let c_j be the knapsack weights from the interval $[0, n-1]$ where $c_j \bmod \pi = x_j + iy_j$. Then

$$(x_j + iy_j) \cdot v\pi^* + (x_j - iy_j) \cdot u\pi \equiv c_j \bmod n\,, \quad j = 0, 1, \ldots, L-1. \tag{5}$$

A person who wants to send a message

$$\mathbf{m} = (m_0, m_1, \ldots, m_{L-1}), \quad m_i \in \{0, 1\}$$

to the owner of the secret key π and $\mathbf{x}$, transmits the value c obtained from

$$c = \sum_{j=0}^{L-1} c_j \cdot m_j\,.$$

The legitimate receiver can compute

$$c \bmod \pi = \sum_{l=0}^{L-1} x_l \cdot m_l + i \cdot \sum_{l=0}^{L-1} y_l \cdot m_l\,. \tag{6}$$

Hence, as real part the easy knapsack is obtained, from which the message $\mathbf{m}$ can easily be recovered.

Let us consider a very small example:

Example 1: Let $n = 1373249 = 1168^2 + 95^2$, and $L = 7$. The easy knapsack be given as the superincreasing vector

$$\mathbf{x} = (2, 5, 9, 19, 41, 81, 199)\,,$$

and the $\mathbf{y}$ vector as

$$\mathbf{y} = (46, 89, 111, 13, 29, 129, 77).$$

Using equation (5), we get the knapsack weights

$$(c_0, c_1, \ldots, c_6) = (317452, 1270973, 1183975, 447972, 259879, 114137, 1068941)\,.$$

To encrypt the message $\mathbf{m} = (1, 0, 0, 1, 0, 0, 1)$ we compute

$$c = \sum_{j=0}^{6} c_j \cdot m_j = 317452 + 447972 + 1068941 = 1834365\,.$$

The legitimate receiver computes

$$\mathrm{Re}\{c \bmod \pi\} = \mathrm{Re}\{220 + i \cdot 136\} = 220\,,$$

and solves the easy knapsack

$$220 = 1 \cdot 199 + 1 \cdot 19 + 1 \cdot 2 \quad \Rightarrow \mathbf{m} = (1, 0, 0, 1, 0, 0, 1)\,.$$

3 An Easy Knapsack with high density

A crucial parameter for the security of knapsack systems is the density of the knapsack which is defined as the ratio of the number of weights of the knapsack and the number of bits of the maximal weight. From [3] we know that we should have a density greater than 0.941 to get a secure knapsack. Hence using a superincreasing knapsack for the x_i is not advisable since then we would get a density smaller than 0.5 in the basic scheme of the previous section. In this section we therefore propose another easy knapsack which is able to produce the desired density in the basic scheme. The easy knapsack proposed also has the desired property that the sum of two knapsack sums is not necessarily another knapsack sum.

The principal idea is to transform the message vector $\mathbf{m} = (m_0, m_1, \ldots, m_{L-1})$ of length L into another vector $\mathbf{M} = (M_0, M_1, \ldots, M_{l-1})$ of length $l > L$ using a data translation code, i.e. to increase the number of weights and thus the density. A suitable class of such data translation codes are the (r, s) runlength-limited (RLL) codes where r zeros must follow every one and at most s zeros can follow a one (for RLL codes see e.g. [1], chapter 8). If we use RLL-codes the integer l is close to $\tau(r, s) \cdot L$ where $\tau(r, s)$ is the reciprocal of the rate of the (r, s) RLL code. The greatest possible rate of an (r, s)-RLL code, the capacity $\lambda(r, s)$, can be obtained as base two logarithm of the largest real root of the polynomial $(x^{s+2} - x^{s+1} - x^{s-r+1} + 1)/(x - 1)$. Using the capacity we obtain a lower bound for $\tau(r, s)$, namely $\tau(r, s) \geq 1/\lambda(r, s)$. Preferably we use a prefix (free) data translation code (see e.g. [1]). If we select the l weights of an easy knapsack according to

$$x_i > \sum_{k \geq 1} x_{i-k(r+1)} \quad \text{and } x_{i+1} > x_i, \tag{7}$$

then the runlength-constraints ensure easy decodability of the sum $\sum M_j x_j$ using the intervals I_i defined by

$$I_i = [T_i, S_i] \quad \text{where } T_i = \sum_{k \geq 0} x_{i-k(s+1)}, \quad \text{and } S_i = \sum_{k \geq 0} x_{i-k(r+1)}\,.$$

For the sketch of a computerprogram to unpack such knapsacks see the appendix. The knapsack of eqn. (7) can also be considered as $(r+1)$ interleaved superincreasing knapsacks. The principles of this easy knapsack are best explained with a simple example which uses a popular (2,7) RLL code, the (2,7) Franaszek code of rate 1/2, given in Table 1. The codewords in Table 1 are prefix-free when read from right to left.

Table 1: (2,7) Franaszek code

Information	Codewords
11	0010
10	0001
000	001000
010	000100
011	001001
0010	00010000
0011	00100100

Example 2: Let $L = 7$, $l = 15$, and use the (2,7) Franaszek code as RLL code. The easy knapsack be given by:

$$\mathbf{x} = (x_0, x_1, \ldots, x_{l-1})$$

$$= (7, 11, 13, 17, 19, 23, 25, 33, 37, 51, 65, 77, 105, 133, 153) ,$$

from which we get the S_i's as

$$(7, 11, 13, 24, 30, 36, 49, 63, 73, 100, 128, 150, 205, 261, 303) ,$$

and the T_i's as

$$(7, 11, 13, 17, 19, 23, 25, 33, 44, 62, 78, 94, 124, 156, 178) .$$

To encode the message $\mathbf{m} = (1011110)$ we use Table 1. Remaining symbols at the end of the message are padded according to the padding rules $0 \rightarrow 000$, $1 \rightarrow 11$, $00 \rightarrow 000$, and $01 \rightarrow 010$. We encode $\mathbf{m}$ from left to right: $10 \rightarrow 0001$, twice $11 \rightarrow 0010$, and $0 \rightarrow 000 \rightarrow 001000$ to get the vector $\mathbf{M}' = (0001001000010001000)$.

¿From Table 1 and the padding rules we see that the highest possible non-zero coefficient of $\mathbf{M}'$ is $l - 1$. Hence we can delete the last three zeros of $\mathbf{M}'$ to get the vector $\mathbf{M}$ of length $l = 15$. In our case, we get $\mathbf{M} = (000100100010001)$. In any case we never need a coefficient higher than $x_{l-1} = x_{14}$. The sum $\sum M_j x_j$ gives

$$x_3 + x_6 + x_{10} + x_{14} = 17 + 25 + 65 + 153 = 260 .$$

To speedup the decryption, we may also transmit the length l' of the vector $\mathbf{M}'$ which is $l' = 18$ in the above example. To decode 260 we proceed according to Table 2. The selection criterion about a possible sequence consists of the equation $T_k \leq \xi \leq S_k$, where k is the highest position having an x-entry in the corresponding row and ξ gives the current sum. At the outset only x_{13} or x_{14} are possible since $T_{13} = 156 \leq 260 \leq 261 = S_{13}$ and $T_{14} = 178 \leq 260 \leq 303 = S_{14}$. As $\xi = 127$ violates $T_9 \leq \xi \leq S_9$ we get $x_{14} = 1$ and because of the constraint $r = 2$ we have $x_{13} = x_{12} = 0$. The value 107 lies in the intervals $I_{10} = [78, 128]$ and $I_{11} = [94, 150]$. Thus in the next step only x_{10} or x_{11} can be 1. The selection criterion is only fulfilled with $x_{10} = 1$. As 42 lies in the intervals I_6 and I_7, only x_6 or x_7 may be 1 in the next step. The codewords leading to $x_6 = 1$ or $x_7 = 1$ are listed in step 3. In this way we finally obtain the vector $\mathbf{M}'$ from which $\mathbf{m}$ can be recovered.

The density of this knapsack equals $(15 - 2)/\log_2(153) \approx 1.79$. The number 2 appears in the numerator since in the knapsack the coefficients x_0 and x_1 will never appear because all codewords in Table 1 begin with two zeros.

Table 2: Sample unpacking of easy knapsack

step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	k	T_k	ξ	S_k	possible
1	x	x	x	x	x	x	x	x	x	x	x	0	0	1	0	0	0		11	94	$107 = 260 - x_{14}$	150	yes
	x	x	x	x	x	x	x	x	x	x	0	0	0	1	0	0	0	0	9	62	$127 = 260 - x_{13}$	100	no
2	x	x	x	x	x	x	x	x	0	0	1	0	0	0	1	0	0	0	7	33	$42 = 107 - x_{10}$	63	yes
	x	x	x	x	x	x	x	x	0	0	0	1	0	0	1	0	0	0	7	33	$30 = 107 - x_{11}$	63	no
	x	x	x	x	x	x	0	0	1	0	0	1	0	0	1	0	0	0	5	23	$107 - x_{11} - x_8$	36	no
3	x	x	x	x	0	0	1	0	0	0	1	0	0	0	1	0	0	0	3	17	$17 = 42 - x_6$	24	yes
	x	x	x	x	0	0	0	1	0	0	1	0	0	0	1	0	0	0	3	17	$9 = 42 - x_7$	24	no
	x	x	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1	11	$42 - x_7 - x_4$	11	no
4	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	-1	0	$0 = 17 - x_3$	0	yes

A decoding problem which can arise is that the easy knapsack as decribed above may have several solutions. This ambiguity may be resolved easily in two different ways (or combinations thereof):

- The constraints on the weights x_i and the RLL-code chosen can be sharpened to make the corresponding sums unique.
- A hashvalue $f(\mathbf{m})$ may be concatenated to the message $\mathbf{m}$ before translating it using the RLL-code. Then the receiver can easily identify the correct solution.

In practice it seems best to use a combination of the above ideas, e.g. the designer can select the x_i such that the intersection of suceeding intervals I_k and I_{k+1} is small or empty. This automatically reduces the number of different solutions and speeds up the decoding process. The average decoding complexity can be given as

$$C(\text{RLL-code}, \mathbf{x}) \cdot C_{\text{superincreasing knapsack}} .$$

The second term is the complexity for decoding a superincreasing knapsack, whereas the first term is a function of the RLL-code used and the knapsackvector $\mathbf{x}$.

On the other hand demanding as few constraints as possible on the x_i and the (r,s) RLL code is good for the cryptographical strength of the entire knapsack-system. Hence it is probably a good idea to allow ambiguities for the solution of the knapsack sums and also impose some constraints on the x_i or on the selection of the RLL-code to keep the constant $C(\text{RLL-code}, \mathbf{x})$ small enough (say $\approx 100 \ldots 10000$).

In the appendix a recursive computer program, written in a Pascal-like syntax, is given which finds all solutions for a knapsack using an arbitrary (r,s) RLL-code. The program can easily be modified and speeded up for use with a specific RLL-code.

Using the above easy knapsack together with the basic scheme of the previous section we get

$$S_{l-1} < \frac{a - b - 1}{2}$$

instead of the constraint eqn. (2), and instead of (3) we have the condition that the sum of any $\lceil l/(r+1) \rceil$ weigths y_j is smaller than $(a - b - 1)/2$.

4 The Whole System

In this section we make some small changements to strengthen the system. Namely we do not publish the c_j, but the weigths

$$d_j \equiv c_{P(j)} \cdot W \bmod n , \tag{8}$$

where $P(i)$ is a random permutation, and W a random integer $< n$ with $\gcd(W, n) = 1$. The permutation and the modular multiplication are introduced to prevent the use of different π' which might transform the c_j into another easy knapsack. If we use the easy knapsack of the preceeding section we simply set $P(j) = j$, i.e. we omit the permutation.

To summarize the whole scheme:

Public Key: The weights d_j, $j = 0, 1, \ldots, l - 1$.

Secret Key: W, n, π, $P(.)$, and $\mathbf{x}$.

Encryption: Form $d = \sum d_j \cdot M_j$.

Decryption: Find the permuted weights of the easy knapsack

$$\mathrm{Re}\{(d \cdot W^{-1} \bmod n) \bmod \pi\} .$$

For actual implementations we propose the following parameters:

$L \approx 200$ and $l \approx \tau L$ such that the density is > 0.941

$\log_2 n \approx 500$ bits.

$\log_2(a - b - 1) \approx 250$ bits.

This means that the size of the public key is about $l \cdot \log_2 n$ bits ≈ 25 Kbyte. An encrypted message has about $\log_2(200) + 500 \approx 508$ bits, and the data rate is $L/\log_2(Ln) \approx 0.39$.

Although we do not recommend publishing n, it seems that if n is selected as product of two large primes, it can be published without compromising the security of the system (assuming that the factorization of integers rests difficult). With n published, c can be reduced modulo n before transmission, which slightly increases the data rate.

5 Security and Generalizations

The system is too new to make thorough statements about its security. However it seems that the attacks used to break many knapsack systems as described in [2] are not successful if the easy knapsack is adequately chosen.

The knapsack scheme proposed can be modified and/or generalized in various ways. For example we may use other easy knapsacks, increase the range of the m_j, or use algebraic integers instead of Gaussian integers.

A Gaussian Integers

In this section we give some results needed in the paper. For further details on Gaussian integers consult e.g. [5]. The Gaussian integers are those complex numbers which have integers as real and imaginary parts. Let $[.]$ denote rounding of integers, then we can define rounding of a complex number $z = x + iy$ as

$$[z] = [x + iy] = [x] + i[y].$$

Computing modulo a complex number π can be defined as

$$z \bmod \pi = z - \left[\frac{z \cdot \pi^*}{\pi \cdot \pi^*}\right] \cdot \pi ,$$

where π^* denotes the conjugate complex number of π. The Gaussian primes are the ordinary primes of the form $\equiv 3 \bmod 4$ as well as the complex numbers $a + ib$ where $a^2 + b^2$ are primes of the form $\equiv 1 \bmod 4$. Given a prime of the form $p \equiv 1 \bmod 4$, we can easily compute the values of a and b using the following algorithm (see [7]).

1. Find x such that $x^2 \equiv -1 \bmod p$.
   (If q_{nr} is a quadratic nonresidue of p then $x \equiv q_{nr}^{(p-1)/4} \bmod p$.)
2. Apply the Euclidean algorithm to p and x; the first two remainders less than $\sqrt{p}$ are a and b.

For details see e.g. Wagons paper.

For example to get the representation of $n = 1373249 = 1009 \cdot 1361$ as sum of two squares we compute $1009 = 28^2 + 15^2$, and $1361 = 31^2 + 20^2$ and find e.g. $(28 - i15)(31 + i20) = 1168 + i95$. Hence $n = 1168^2 + 95^2$.

If r and s are integers from the interval $[-(a - b - 1)/2, (a - b - 1)/2]$, then it is not difficult to show that

$$r + is \bmod \pi = r + is ,$$

which explains the constraints (2) and (3). (The number of Gaussian integers which are left unchanged by the modulo operation equals $\pi\pi^*$, to keep things simple, we let the size of r and s be smaller than $(a - b - 1)/2$.)

B Decoding Algorithm for (r,s) constrained Knapsack

```
procedure decode(sum,l1,l2:integer;var dec:menge);
var decolist : menge;
    i        : integer;
begin
 if sum=0 then writeln('Solution=',dec)
   else begin
         decolist:=ininterval(sum,l1,l2);
         for i:=max(l1,0) to l2 do
          if i in decolist then
                       begin
                        dec:=dec + [i];
                        decode(sum-x[i],i-s-1,i-r-1,dec);
                        dec:=dec - [i];
                       end;
       end;
end;
```

In the above program *menge* denotes *set of 0..l-1*. The function ininterval returns a set containing the indices of the intervals where *sum* lies in with the additional condition that the indices are from $l1 \ldots l2$. The lines dec := dec $\pm [i]$ are the Pascal notation for adding/removing the element i to/from the set dec. The program is called with decode($sum, 0, l - 1,$ dec), with dec initially set to the empty set [].

Acknowledgment: I would like to thank A.Odlyzko for sending a copy of [3], which stimulated section 3, and J.Schwenk for carefully reading the manuscript.

References

[1] R.E. Blahut, "Digital Transmission of Information", Addison-Wesley 1990.

[2] E.F. Brickell, M.Odlyzko, "Cryptanalysis: A Survey of Recent Results", *Proceedings of the IEEE*, Vol. 76, No.5, May 1988, pp.578-593.

[3] M.J.Coster, A.Joux, B.A.LaMacchia, A.M.Odlyzko, C-P.Schnorr, J.Stern, "Improved Low-Density Subset Sum Algorithms", *Computational Complexity*, 2, (1992), pp.111-128.

[4] B.Chor, R.L.Rivest, "A Knapsack-Type Public key Cryptosystem based on Arithmetic in Finite Fields", *IEEE Transactions on Information Theory*, Vol.IT-34, No.5, September 1988, pp.901-909.

[5] G.H.Hardy, E.M.Wright, "An introduction to the theory of numbers", fifth edition, Oxford 1979.

[6] R.C.Merkle, M.E.Hellman, "Hiding Information and Signatures in Trapdoor Knapsacks", *IEEE Transactions on Information Theory*, Vol.IT-24, No.5, September 1978, pp.525-530.

[7] S.Wagon, "The Euclidean Algorithm Strikes again", *American Mathematical Monthly*, Vol.97, No.2, 1990, pp.125-129.