The low density attack against the Merkle-Hellman cryptosystem

Jannik Weiß

Faculty of Computer and Information Science
Večna pot 113
1000 Ljubljana

Abstract. Knapsack cryptosystems were among the first public key cryptosystems to be introduced. While they were initially attractive because of fast encryption and decryption, various successful attacks against knapsack cryptosystems made them unpopular and generally considered insecure. The low density attack is a popular and general approach to attacking knapsack-based cryptosystems. It transforms breaking the ciphertext into a lattice shortest vector problem that can be approximated with the Lenstra-Lenstra-Lovasz (LLL) algorithm. With a standard present day personal computer this attack can break the original knapsack cryptosystem (the Merkle-Hellman cryptosystem) fast and with moderate success.

Key words: knapsack cryptosystems, Merkle-Hellman cryptosystem, low density attack, LLL algorithm

1 Introduction

Knapsack public key cryptosystems were among the first public key cryptosystems to be introduced. They are based on the so called knapsack or subset sum problem, which is to find a subset of numbers from a given set that sum up to a given value. While knapsack based cryptosystems are much faster than for example RSA, they are generally not considered secure [8]. The original knapsack cryptosystem, published by Ralph Merkle and Martin Hellman and called the Merkle-Hellman cryptosystem [5], was broken in 1982 by Adi Shamir [9] and also subsequent variants of the cryptosystem have been broken by different approaches. Only a few recent examples claim to be secure, like the proposition by Yasuyuki Murakami and Takeshi Nasako [6], but suffer from skepticism and unpopularity given the history of knapsack cryptosystems.

Nevertheless knapsack cryptosystems are historically interesting, but it is also important to study attacks against this cryptosystem and its variants as even today there are new knapsack cryptosystems being introduced (like [2]).

In this paper we look at the low density attack, one of the most effective and most general approaches to breaking a knapsack cryptosystem and find that this approach can break the originally proposed MH cryptosystem around half of the time in a very short time on today's regular hardware.

The remainder of this paper is organized as follows: In section 2 we explain the Merkle-Hellman cryptosystem, section 3 covers the low density attack on the cryptosystem, in section 4 we test the previously explained attack with an implementation, and we finish with a conclusion in section 5.

2 The Merkle-Hellman Cryptosystem

2.1 The standard and superincreasing knapsack problem

In cryptography, the knapsack or subset sum problem refers to the following problem: Given a sequence of integers $A = \{a_1, ..., a_n\}$ and a value V, can we find $X = \{x_1, ...x_n\}, x_i \in \{0, 1\}$ such that

$$\sum_{i\in\{1,...,n\}} a_i x_i = V$$

In other words, can we find which a_i sum up to V. We are only interested in solvable instances of this problem, since we only construct solvable instances for the cryptosystem.

The knapsack problem is known to be NP-hard. However, there are specific variants of this problem, which are easy and fast to solve. One such variant is the knapsack problem, where A is superincreasing. Superincreasing means that every integer in A is greater than the sum of all previous integers in A:

$$\forall a_i \in A : a_i > \sum_{j\in\{1,...,i-1\}} a_j$$

In this case X can be determined quickly by algorithm 1, which is intuitively similar to converting a decimal number to binary.

Algorithm 1: Solving a superincreasing knapsack problem

```
  Data: $V$, $A$
  Result: $X$
1 for $i \in \{n, ..., 1\}$ do
2     if $V \geq a_i$ then
3         $x_i \leftarrow 1$;
4         $V \leftarrow V - a_i$;
5     else
6         $x_i \leftarrow 0$;
7     end
8 end
```

2.2 *The Merkle-Hellman cryptosystem*

The Merkle-Hellman cryptosystem is constructed in such a way that in order to compute the plaintext the intended receiver only needs to solve an easy superincreasing knapsack problem, but an attacker (without access to the private key) has to solve a seemingly hard general knapsack problem.

A superincreasing set A serves as the private key, together with adequately chosen integers r and q. The restrictions for r and q are explained together with the decryption where it is clear to see why they are needed. r and q are used to construct the set B which serves as the public key and is not superincreasing anymore:

$$B = \{b_i := a_i r \bmod q \mid i \in \{1, ..., n\}\}$$

When encrypting a message, the ciphertext C is computed using the plaintext X and B:

$$C = \sum_{i \in \{1,...,n\}} b_i x_i$$

The ciphertext can be decrypted using the private key by solving the easy knapsack problem with the superincreasing set A and $V = Cr^{-1} \bmod q$. Here we see that we need $q > \sum_{i \in \{1,...,n\}} a_i$, because this is the highest number that can occur, namely with a plaintext of all ones. We also need $\gcd(r, q) = 1$ for r^{-1}. The knapsack problem with C and B and the problem with V and A have in fact the same solution, because

$$\begin{aligned} Cr^{-1} &= \sum_{i \in \{1,...,n\}} b_i r^{-1} x_i \\ &= \sum_{i \in \{1,...,n\}} a_i r r^{-1} x_i \\ &= \sum_{i \in \{1,...,n\}} a_i x_i \pmod q \end{aligned}$$

The modular transformation from A to B aims to disguise the easy knapsack by turning it into a seemingly random general knapsack while leaving a trapdoor to recover the easy knapsack using the private key.

2.2.1 Example: We have a running example with $n = 8$. We choose a superincreasing set

$$A = \{203, 353, 800, 1869, 3842, 8150, 16139, 32720\}$$

and $r = 56909$ and $q = 235298$. For the public key B we calculate

$$\begin{aligned} 203 * 56909 \bmod 235298 &= 22925 \\ 353 * 56909 \bmod 235298 &= 88547 \\ &... \\ 32720 * 56909 \bmod 235298 &= 149406 \end{aligned}$$

to get

$$\begin{aligned} B = \{&22925, 88547, 114686, 8225, \\ &52536, 35992, 86257, 149406\} \end{aligned}$$

We encrypt the message

$$X = \{1, 0, 1, 1, 1, 0, 1, 0\} = 186_{10}$$

and calculate the ciphertext $C = B^\top X = 284629$. In decryption, $V = Cr^{-1} \bmod q = 22853$.

3 The low density attack

There are multiple ways to attack the Merkle-Hellman cryptosystem. The first attack that broke the cryptosystem was discovered by Shamir [9]. It finds that the public knapsack is in fact not completely general and exploits the structure of the transformation to find the private key or at least an equivalent easy knapsack.

The other type of attack is to simply try to solve the knapsack problem of the public key (B) and the ciphertext. This approach does not concern itself with the specific method of transforming the easy knapsack into a harder one and is therefore very general and useful for many different variants of knapsack cryptosystems. It is also referred to as the low density attack [3]. It makes use of lattice theory and the Lenstra-Lenstra-Lovasz (LLL) algorithm [4], which are both used to also create modern cryptosystems after first being influential in breaking knapsack cryptosystems [7].

3.1 *Density*

The density d of a cryptosystem refers to the ratio of plaintext bits to ciphertext bits:

$$d = \frac{\#\text{PlaintextBits}}{\#\text{CiphertextBits}}$$

In the context of the Merkle-Hellman cryptosystem the density is

$$d = \frac{n}{\log_2 \sum_{i=1}^{n} b_i}$$

but sometimes also defined as

$$d = \frac{n}{\log_2 \max b_i}$$

The low density attack involves lattices and gives an upper bound for the cryptosystem density up until which the attack breaks most instances of the knapsack problem.

3.2 Lattices

A lattice is defined by a lattice basis, which is a set of linearly independent vectors. A lattice contains all integer linear combinations of the basis vectors. Formally, a lattice and its basis (here S) are defined as follows:

$$S = \{s_1, ..., s_n\} \subset \mathbb{R}^m \text{ linearly independent}$$

$$L(S) = \{\sum_{i=1}^{n} k_i s_i, k_i \in \mathbb{Z}\}$$

Useful and interesting basic questions regarding lattices are to find short vectors in a lattice or to find nice bases for a lattice which have reasonably orthogonal and short vectors. Given any basis of a lattice where the basis vectors can be arbitrarily large, finding the shortest nontrivial vector in the lattice is an NP-hard problem. However there are ways to find approximate solutions to the shortest vector problem in polynomial time, namely with the LLL algorithm [4], which is explained in the next subsection.

3.3 The lattice attack

Jeffrey Lagarias and Andrew Odlyzko first described how to break a knapsack cryptosystem using lattice theory [3]. The general idea is to transform the knapsack problem into a lattice shortest vector problem such that a short vector in the lattice yields the solution to the knapsack problem. First, however, we take a look at the LLL algorithm (algorithm 3). The algorithm uses the Gram-Schmidt orthogonalization process (GS), which we quickly recall.

3.3.1 Gram-Schmidt orthogonalization: The GS process (algorithm 2) turns any basis of a vector space into an orthogonal basis of the vector space.

Algorithm 2: Gram-Schmidt orthogonalization

```
Data: Basis $S = \{s_1, ..., s_n\}$
Result: Orthogonal basis $S^*$
1 $s_1^* \leftarrow s_1$;
2 for $i \in \{2, ..., n\}$ do
3     $s_i^* \leftarrow s_i - \sum_{j=1}^{i-1} \mu(s_i, s_j^*) s_j^*$
4 end
```

The GS process works by going through the given basis vectors and at each step making the current working vector orthogonal to all previous vectors by subtracting away the current vector's parts that are parallel to each of the previous vectors. The value $\mu(v_1, v_2)$ is the magnitude of the projection of v_1 onto v_2 relative to the magnitude of v_2:

$$\mu(v_1, v_2) = \frac{v_1 \cdot v_2}{v_2 \cdot v_2}$$

$\mu(v_1, v_2)v_2$ is therefore the actual projected vector.

Algorithm 3: Lenstra-Lenstra-Lovasz (LLL) algorithm

```
Data: Lattice basis $S = \{s_1, ..., s_n\}$
Result: Orthogonal lattice basis
1  $S^* \leftarrow GS(S)$;
2  $k = 2$;
3  while $k \leq n$ do
4      for $i \in \{k-1, ..., 1\}$ do
5          if $\mu(s_k, s_i^*) > 0.5$ then
6              $s_k \leftarrow s_k - \text{round}(\mu(s_k, s_i^*)) * s_i$;
7          end
8      end
9      $S^* \leftarrow GS(S)$;
10     if $(\delta - \mu(s_k, s_{k-1}^*))||s_{k-1}^*||^2 \leq ||s_k^*||^2$ then
11         $k \leftarrow k + 1$;
12     else
13         swap$(s_k, s_{k-1})$;
14         $S^* \leftarrow GS(S)$;
15         $k \leftarrow \max(k-1, 2)$;
16     end
17 end
```

3.3.2 The LLL algorithm: The LLL algorithm has similarities to the GS orthoginalization and uses GS as a subroutine, as well as μ. The difference is of course that we are now working with lattices and exactly projected vectors likely do not point to actual points in the lattice. The LLL algorithm therefore finds a new basis of the lattice where the basis vectors are reasonably orthogonal to each other and also reasonably short. The resulting basis vectors are actually roughly ordered by length, so the shortest vector will be one of the first ones of the basis. Why the vectors are only roughly and not exactly ordered by length has to do with the fact that vectors have to be in the lattice. Within the granularity of the lattice two vectors might be close enough in length such that their order does not matter. Like the GS process, the LLL algorithm also works with one vector at a time and then moves on to the next one.

The main part of the LLL algorithm is the size reduction part (lines 4-8). This part looks roughly similar than the GS process. The LLL algorithm keeps a GS orthogonalized bases of the given basis constantly updated, which serves as a sort of "ideal" basis. In the size reduction part the current working vector s_k is reduced by subtracting away parts that are roughly parallel to each of the previous vectors. To determine the magnitude (μ) of these parts, however, the corresponding

GS vector is used. To stay within the lattice only integer multiples of a previous vector can be subtracted from the current working vector, which is why μ is rounded. This rounding also explains the threshold of 0.5 (line 5). In the context of the LLL algorithm $\mu(v_1, v_2)$ means that we can not length reduce a vector v_1 any further using a vector v_2. Critically, this does not mean that we can not length reduce v_2 any further using v_1. This is where the swap part of the LLL algorithm comes into play.

The condition in line 10 is called the Lovasz condition and it checks whether two vectors are roughly orthogonal. This means that the aforementioned implication goes in both directions, that is if neither of two vectors can be reduced any further using the other one. In particular, the Lovasz condition will fail if the vector s_{k-1} is longer than s_k, such that s_k can not be reduced further using s_{k-1}. The parameter δ can be in the range from 0.25 to 1 and implementations usually default to a value very close to 1. Swapping the two vectors, i.e. moving s_k one position to the front, orders them roughly by length, and in a future size reduction step the longer vector will be reduced using the shorter one. The swap also entails that we move the index k back one position and theoretically the current working vector could be swapped all the way to the lowest index. This means that overall the LLL algorithm goes from one vector to the next, with an occasional swapping a vector a few positions towards the front and starting over from that position.

The resulting basis is short, roughly orthogonal and roughly ordered by length. It satisfies the following conditions (size condition and Lovasz condition):

$$\forall 1 \leq i < k \leq n : \mu(s_k, s_i^*)$$

$$\forall 1 < k \leq n : (\delta - \mu(s_k, s_{k-1}^*))||s_{k-1}^*||^2 \leq ||s_k^*||^2$$

3.3.3 Attack using the LLL algorithm: A knapsack problem can be transformed into a shortest vector problem in a lattice by creating a specific lattice basis using the values from the knapsack set B and the ciphertext C. We create $n+1$ basis vectors, one for each b_i and one for C. We want that a short vector is a linear combination of exactly the basis vectors that correspond to the b_i that sum up to C; but this short vector should also expose the b_i of which it is composed. We can achieve this by giving every B-based basis vector a 1 in a unique dimension and we select one dimension in which all B-based vectors have their b_i value. Lastly in this same dimension the C-based basis vector has the value $-C$. Now, in a linear combination of the C-based vector and the correct B-based vectors, all their values in the "common" dimension cancel to 0 and the rest of the resulting vector will have ones in the dimensions of the correct b_i. The so constructed basis S looks like this (basis vectors are rows):

$$S = \begin{bmatrix} 1 & 0 & 0 & & 0 & b_1 \\ 0 & 1 & 0 & \dots & 0 & b_2 \\ 0 & 0 & 1 & & 0 & b_3 \\ & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & & 1 & b_n \\ 0 & 0 & 0 & \dots & 0 & -C \end{bmatrix}$$

After applying the LLL algorithm to S it contains lots of short vectors and if among them is one that contains a 0 in the $(n+1)$-th dimension and else only 0s and 1s, and if this vector yields a correct solution to the knapsack problem, the plaintext is recovered.

Continuing the example from earlier, the particular S looks like this:

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 22925 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 88547 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 114686 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 8225 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 52536 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 35992 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 86257 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 149406 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -284629 \end{bmatrix}$$

After applying the LLL algorithm, we can see that the first vector is the plaintext X plus the additional 0 at the end.

$$S = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 2 & -1 & 3 & 1 & 1 & -1 \\ 1 & 0 & 2 & 0 & -1 & 3 & -2 & 1 & 0 \\ 0 & 1 & 0 & -3 & 3 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -2 & 1 & 1 & -1 & 2 & 4 \\ -1 & -1 & 3 & 0 & -1 & -3 & -1 & 2 & 0 \\ 1 & 3 & -2 & 2 & -1 & 0 & -2 & 1 & 0 \\ -3 & 0 & -1 & 2 & 2 & -1 & -2 & -2 & 1 \\ 1 & 4 & 0 & -2 & 0 & -1 & 3 & -2 & 1 \end{bmatrix}$$

Lagarias and Odlyzko [3] showed that when the density of a knapsack cryptosystem is less than 0.6463... the cryptosystem can be successfully attacked with this approach. Later Coster et al. [1] improved this upper bound on the density to 0.9408... by making only a small change to the process of how the lattice basis S is created. Given

$$N > \sqrt{n}$$

S is constructed in the following way:

$$S = \begin{bmatrix} 1 & 0 & 0 & & 0 & Nb_1 \\ 0 & 1 & 0 & \dots & 0 & Nb_2 \\ 0 & 0 & 1 & & 0 & Nb_3 \\ & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & & 1 & Nb_n \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \dots & \frac{1}{2} & NC \end{bmatrix}$$

4 Experiments

An interesting question to ask is how well the Merkle-Hellman knapsack cryptosystem in it's originally proposed configuration could be broken using the low density attack on a present-day personal computer. Merkle and Hellman suggested in 1978 that n should be above 100 or 200. We implement the MH cryptosystem and both versions of the attack in python with Sage and run 100 attempts for n from 10 to 100. Because implementations of the LLL algorithm are easy to find, implementing the attack is very straight forward. The results are shown in Figure 1.

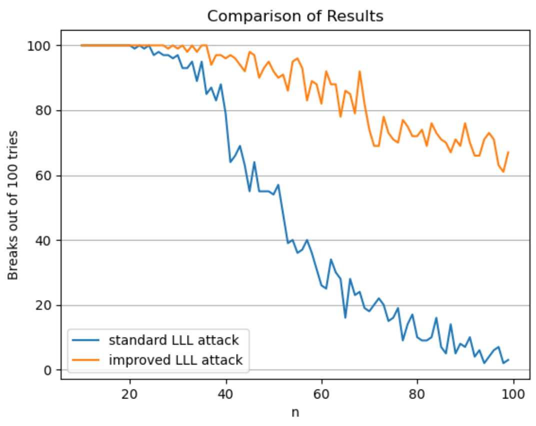

Figure 1. Success of two versions of the low density attack on the Merkle-Hellman cryptosystem

We can see in Figure 1 that for low values of n both versions of the attack break the cryptosystem almost every time. However, the first approach fails almost every time as n reaches 100, while the improved version of the attack still succeeds more than half the time.

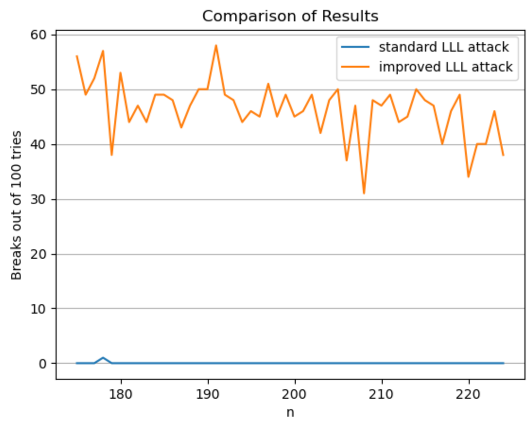

Figure 2. Success of two versions of the low density attack on the Merkle-Hellman cryptosystem

Figure 2 shows the vicinity of $n = 200$. We see that the improved attack still breaks the cryptosystem in around 50% of cases, while the standard attack is not successful anymore. When n is around 200 a single attack takes around 5-6 seconds. The knapsack cryptosystem, implemented according to [5], has a density of approximately 0.5. Notably, 0.5 is less than both upper bounds for the two approaches, however we still see a significant difference in success between the two attacks.

All code is available on GitHub.

5 Conclusion

In conclusion, this paper explains knapsack cryptosystems, focusing on the original Merkle-Hellman cryptosystem. These types of cryptosystems are generally considered not secure, due to various attacks that have been developed over time.

The popular and easily accessible low density attack against knapsack cryptosystems using lattice theory and the LLL algorithm is explained thoroughly. Lattice theory and associated problems and algorithms are popular in the development of modern cryptosystems designed to withstand attacks using quantum computers, but also an important tool for cryptoanalysis of existing cryptosystems.

Experimental results show that the low density attack can be easily implemented and is successful against the original MH cryptosystem, even for higher values of n. However, there is a large difference in the effectiveness of different variants of the attack.

To this day new knapsack-based cryptosystems (like [2] or [6]) are proposed and their security needs to be analyzed. Can you break them? Can you tailor or improve the attack presented in this paper?

References

[1] Coster, Matthijs, Antoine Joux, Brian Lamacchia, Andrew Odlyzko, Claus Schnorr, and Jacques Stern: *Improved low-density subset sum algorithms*. Volume 2, April 1999, ISBN 978-3-540-54620-7.

[2] Díaz, Raúl Durán, Luis Hernández-Álvarez, Luis Hernández Encinas, and Araceli Queiruga-Dios: *Chor-rivest knapsack cryptosystem in a post-quantum world*. In Daimi, Kevin, Hamid R. Arabnia, Leonidas Deligiannidis, Min Shiang Hwang, and Fernando G. Tinetti (editors): *Advances in Security, Networks, and Internet of Things*, pages 67–83, Cham, 2021. Springer International Publishing, ISBN 978-3-030-71017-0.

[3] Lagarias, J. C. and A. M. Odlyzko: *Solving low-density subset sum problems*. J. ACM, 32(1):229–246, jan 1985, ISSN 0004-5411. https://doi.org/10.1145/2455.2461.

[4] Lenstra, Arjen, Hendrik Lenstra, and Lovász László: *Factoring polynomials with rational coefficients*. Mathematische Annalen, 261, December 1982.

[5] Merkle, R. and M. Hellman: *Hiding information and signatures in trapdoor knapsacks*. IEEE Transactions on Information Theory, 24(5):525–530, 1978.

[6] Murakami, Yasuyuki and Takeshi Nasako: *Knapsack public-key cryptosystem using chinese remainder theorem*. IACR Cryptology ePrint Archive, 2007:107, January 2007.

[7] Nguyen, Phong and Jacques Stern: *The two faces of lattices in cryptology*. LNCS, 2146, September 2002.

[8] Odlyzko, Andrew M.: *The rise and fall of knapsack cryptosystems*. 1998. https://api.semanticscholar.org/CorpusID:115995195.

[9] Shamir, A.: *A polynomial-time algorithm for breaking the basic merkle - hellman cryptosystem*. IEEE Transactions on Information Theory, 30(5):699–704, 1984.