Laboratoire d'Informatique Fondamentale, de Recherche Opérationnelle, de Combinatoire et d'Économétrie

Availaible on line at http://www.liforce.usthb.dz
bulletin-liforce@usthb.dz
Bulletin du Laboratoire 03 (2017) 57 - 73

Cryptanalysis of Merkle-Hellman cipher using Parallel Genetic Algorithm

Nedjmeddine KANTOUR[1], Sadek BOUROUBI[2].

[1,2] Laboratory L'IFORCE, Faculty of Mathematics,
University of Sciences and Technology Houari Boumediene,
P.B. 32 El-Alia, 16111, Bab Ezzouar, Algiers, Algeria.

nkantour@usthb.dz[1], sbouroubi@usthb.dz[2].

Abstract: In 1976, *Whitfield Diffie* and *Martin Hellman* introduced the public key cryptography or asymmetric cryptography standards. Two years later, an asymmetric cryptosystem was published by *Ralph Merkle* and *Martin Hellman* called $\mathcal{MH}$, based on a variant of knapsack problem known as the subset-sum problem which is proven to be *NP-hard*. Furthermore, over the last four decades, Metaheuristics have achieved remarkable progress in solving *NP-hard* optimization problems. However, the conception of these methods raises several challenges, mainly the adaptation and the parameters setting. In this paper, we propose a Parallel Genetic Algorithm (PGA) adapted to explore effectively the search space of considerable size in order to break the $\mathcal{MH}$ cipher. Experimental study is included, showing the performance of the proposed attacking scheme, and finally a concluding comparison with lattice reduction attacks.

Keywords: Cryptanalysis, Merkle-Hellman Cryptosystem, Knapsack Problem, Genetic Algorithm, Lattice Reduction Algorithms.

1 Introduction

The Merkle-Hellman Cipher has been a subject to several attacks (see [6], [7], and [19]). The existing attacks are essentially heuristic methods, using lattice reduction algorithms or metaheuristics. Mainly, we mention the lattice reduction attack for low density knapsack problem proposed by Lagarias and Odlyzko in [17], and improved by Coaster and al. [18]. Later on, Schnorr and Shevchenko gave a progressive algorithm for solving the knapsack problem with density close to 1 [26]. On the other hand, metaheuristic based attacks was initiated by Spillman [7], and followed by many other adaptations (see [9], [21], [22], [23], [24], [25]). We mention also some exact resolution methods designed for the subset-sum problem, running in pseudo-polynomial time complexity (see [27], [28], [29]). However, this latter usually depends on input values (public key and target sum). Where in the case of $\mathcal{MH}$ cipher, the public key values and target sum can be very large compared to the public key size, which restricts the efficiency of exact resolution methods and solvers softwares against this cipher. The common factor among the existing approaches is that they all proceed one block at a time attack. Thus, since the aim is to solve multiple knapsack problems known to be *NP-hard*, these approaches can be too expensive. In this paper we present a parallel heuristic approach using a Genetic Algorithm adapted to decrypt simultaneously multi-blocks ciphertext, encrypted using $\mathcal{MH}$ cipher.

2 Merkle-Hellman cipher

The $\mathcal{MH}$ Cipher is one of the first asymmetric cryptosystems proposed in 1978 based on the knapsack problem, and this problem is proven to be *NP-hard* [3]. However, there exist some instances where it can be solved in a linear time, as in the case of super-increasing sequence. Therefore, Merkle and Hellman proposed an arithmetical transformation ensuring the transition from a trivial knapsack based on super-increasing sequence to a hard one, identified as a trapdoor sequence. In summary, this cryptosystem uses a trivial knapsack parameter as a private key, and then transforms it to a trapdoor sequence which serves as a public key [1]. These keys are generated thusly:
The private key consists of a super-increasing sequence $\{a_1, a_2, \ldots, a_n\}$, an integer m with $m > \sum_{i=1}^{n} a_i$, an integer $w \in \{1, \ldots, m-1\}$ which is prime with m, and a permutation δ of the set $\{1, \ldots, n\}$; hence, the public key is deduced from the private key by calculating $b_i = a_{\delta(i)} w \ [m]$, for each $i \in \{1, \ldots, n\}$. Let $M = m_1 m_2 \ldots m_k$ be a plaintext written in binary (ASCII code), where $m_i = x_1^i x_2^i \ldots x_n^i,\ x_j^i \in \{0, 1\}$, and let $C = (c_1, c_2, \ldots, c_k)$ be the correspondent ciphertext, with $c_i = \sum_{j=1}^{n} b_j x_j^i,$ for all $i \in \{1, \ldots, k\}$. To decrypt C we first calculate $d_i = w^{-1} c_i \ [m]$, for each $i \in \{1, \ldots, k\}$, secondly, we solve the trivial knapsack $d_i = \sum_{j=1}^{n} a_j e_j^i$, , finaly, deducing x_j^i from $x_j^i = e_{\delta(j)}^i$.

3 Cryptanalytic process

We want to carry out an attack on a ciphertext $C = (c_1, c_2, \ldots, c_k)$, with the aim of finding the correspondent plaintext, this amounts to uncovering the information $M^\star = m_1^\star m_2^\star \ldots m_k^\star,\ m_i^\star = x_1^{\star i} x_2^{\star i} \ldots x_n^{\star i},\ x_j^{\star i} \in \{0, 1\}$, ensuring:

$$c_i - \sum_{j=1}^{n} b_j \, x_j^{\star_i} = 0, \ \forall i \in \{1, \ldots, k\},$$

where $\{b_1, b_2, \ldots, b_n\}$ is the public key.

Therefore, we associate to the decryption process the mathematical model $(\mathcal{P})$ where the plaintext $M^\star$ represents its optimal solution; this model is well known as the Multiple Knapsack Problem.

$$(\mathcal{P}) \begin{cases} Min\,(\mathcal{Z}) = \sum_{i=1}^{k} \sum_{j=1}^{n} b_j x_j^i, \\ \sum_{j=1}^{n} b_j \, x_j^i \geq c_i, \ \forall i \in \{1, \ldots, k\}, \\ x_j^i \in \{0, 1\}, \ \forall j \in \{1, \ldots, n\}. \end{cases}$$

In order to construct a flexible mathematical model, and considering the fact that only the optimal solution breaks the ciphertext (the best or nothing situation). We associate a cryptanalysis model $(\mathcal{P}_i)$ to each ciphered block i, where its optimal solution is $m_i^\star$. Moreover, solving $(\mathcal{P})$ is equivalent to find the optimal solution for each problem $(\mathcal{P}_i)$, $1 \leq i \leq k$. Thus, we can rely on the following model while it maintains the same optimal solution:

$$(\mathcal{P}_i) \begin{cases} Min(\mathcal{Z}) = |c_i - \sum_{j=1}^{n} b_j x_j^i|, \\ x_j^i \in \{0, 1\}, \ \forall j \in \{1, \ldots, n\}. \end{cases}$$

To find the plaintext we are required to solve numerous *NP-hard* problems $(\mathcal{P}_i)_{1 \leq i \leq k}$, thus, we propose an attacking scheme that uses multiple Genetic Algorithms in a parallel way, in which they are enhanced with distinctive search and communication strategies.

4 Genetic Algorithm

Genetic Algorithms (GAs) have been developed by John Holland in 1975, and presented in his book *Adaptation in Natural and Artificial Systems* [12]. Genetic algorithms are optimization algorithms inspired by natural evolution mechanisms and genetic science. The main idea is to combine the principle "survival of the fittest" with information exchange among string structures [10]. This algorithm is initiated with a set of chromosomes (i.e., potential solutions) called initial population, and it uses "natural selection" along with genetics-inspired operators (crossover, mutation and selection) to move from a current population to a new one [11]. Hence, Genetic Algorithm reduces the search procedure from exponential-ordered search space to a collection of incrementally adapted solutions. Regarding the adaptation measurement, a fitness function is defined to evaluate chromosomes according to their performance against a given problem. In the rest of this section we present an adaptation of Genetic Algorithm with slight modifications to solve the knapsack problem $(\mathcal{P}_i)$ defined above. Starting with the representation of the chromosomes m_i (i.e., solutions of the problem $(\mathcal{P}_i)$), we have chosen to use a natural

representation so as a chromosome is a binary string, thus, $m_i = x_1^i x_2^i \dots x_n^i$, $x_j^i \in \{0, 1\}$, where n is the public key length. To initiate the algorithm we need to generate an initial population, for that, we use a greedy heuristic that generates randomly a large set of chromosomes and then retains a given number of best fitted ones, so as to provide a set of chromosomes of "acceptable" quality. When entering the algorithm, we randomly select parents form the current population according to a mating probability p_c (called also crossover probability) , the selected chromosomes are in charge of producing the next population. For that, we need to choose the operators that implements the following operations: crossover, mutation, and selection. The crossover operator is defined in GAs as an analogy of the mecanism that allows the reproduction of chromosomes in nature, the aim of this operator is to produce new chromosomes that partially inherit caracteristics from its parents, with copying and recombinnig their genes in a delaberate way. For the mating process we used three crossover operators proceeding for each pair, first, we use a very classical operator, choosing a cutting point, say $c \in \{2, \dots, n-1\}$, here the mating process consists of swapping bits $c+1$ to r of the first parent P_1 with bits $c+1$ to r of the second parent P_2, hence, new chromosomes C_1 and C_2 are created [7].

$$P_1 = \mid 1 \mid 1 \mid 1 \mid 0 \mid 1 \mid \mathbf{1} \mid \mathbf{0} \mid \mathbf{0} \mid \longrightarrow C_1 = \mid 1 \mid 1 \mid 1 \mid 0 \mid 1 \mid \mathbf{1} \mid \mathbf{1} \mid \mathbf{1} \mid$$

$$\text{swap} \updownarrow$$

$$P_2 = \mid 0 \mid 1 \mid 1 \mid 0 \mid 0 \mid \mathbf{1} \mid \mathbf{1} \mid \mathbf{1} \mid \longrightarrow C_2 = \mid 0 \mid 1 \mid 1 \mid 0 \mid 0 \mid \mathbf{1} \mid \mathbf{0} \mid \mathbf{0} \mid$$

Figure 1: One cutting point crossover operator.

The second operator is similar to the first one but here we use two cutting points, and new chromosomes are generated swapping the middle parts of the parents, as it shown in Figure 2. The third operator constructs one child chromosome C by alternately choosing random

$$P_1 = \mid 1 \mid 1 \mid \mathbf{1} \mid \mathbf{0} \mid \mathbf{1} \mid \mathbf{1} \mid 0 \mid 0 \mid \longrightarrow C_1 = \mid 1 \mid 1 \mid \mathbf{1} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{1} \mid 0 \mid 0 \mid$$

$$\text{swap} \updownarrow$$

$$P_2 = \mid 0 \mid 1 \mid \mathbf{1} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{1} \mid 1 \mid 1 \mid \longrightarrow C_2 = \mid 0 \mid 1 \mid \mathbf{1} \mid \mathbf{0} \mid \mathbf{1} \mid \mathbf{1} \mid 1 \mid 1 \mid$$

Figure 2: Two cutting point crossover operator.

genes from the parents (see Figure 3), this operator is known as the uniform crossover operator. The newly produced chromosomes are added to the current population,

$$\left.\begin{array}{l} P_1 = \mid \underline{\mathbf{1}} \mid \underline{\mathbf{1}} \mid 1 \mid 0 \mid \underline{\mathbf{1}} \mid 1 \mid 0 \mid \underline{\mathbf{0}} \mid \\ P_2 = \mid 0 \mid 1 \mid \underline{\mathbf{1}} \mid \underline{\mathbf{0}} \mid 0 \mid \underline{\mathbf{1}} \mid \underline{\mathbf{1}} \mid 1 \mid \end{array}\right\} \rightarrow \quad C = \mid 1 \mid 1 \mid 1 \mid 0 \mid 1 \mid 1 \mid 1 \mid 0 \mid$$

Figure 3: Uniform crossover operator.

where at this point, we perform mutation according to a probability p_m, here, its value is usually chosen less than 0.1. Furthermore, the used operator consists of replacing a block of genes by its complementary, this latter are generally of length one.

$$P_1 = \mid 1 \mid \boxed{1} \mid 1 \mid 0 \mid 1 \mid 1 \mid 0 \mid 0 \mid \rightarrow P_1' = \mid 1 \mid \boxed{0} \mid 1 \mid 0 \mid 1 \mid 1 \mid 0 \mid 0 \mid$$

Figure 4: Mutation operator.

In order to improve the fitness of chromosomes in the current population (including the offsprings produced in the previous steps), and accelerate the search process before launching the selection operator, we apply a low-cost improving heuristic that essentially withdraws a given number of chromosomes, and then attempts to apply on each drawn chromosome corrections on its genes to extract from it a better fit chromosome, by bringing it to a local optimum. In case of a positive result, the produced chromosomes will be added to the current population, replacing the existing parents. The following step is to select the chromosomes to maintain for the next generation, in which they will serve as parents. Hence, this operator can be critical, since at this point it selects the chromosomes that survive to the next iteration, and potentially spread their genes in forthcoming generations. In contrast, it can also eliminates chromosomes qualified by the chosen operator as unadapted. We used two randomly chosen operators according to a probability p_s. To begin with, we employed the elitist selection, that consists to retain the best fitted chromosomes for the next generation, this operator prevent the loss of "good" genes, in contrast, it can cause a premature convergence. The other used operator is known as the roulette wheel selection, this latter consists of associating for each chromosome i a probability of selection p_i according to its fitness value f_i. Moreover, we allocate for each chromosome i a proportion of length p_i in the segment $[0, 1]$ (an angle of $2\pi p_i$ in case of circular representation), for that we use the Algorithm 1 below:

Algorithm 1

Input: Fitness values $f_1, \ldots, f_n$
Output: Selection probabilities $p_1, \ldots, p_n$

1: Calculate $f_{max} := \max_{1 \leq i \leq n} \{ f_i \}$;
2: For each $i \in \{1, \ldots, n\}$, $f_i := f_{max} - f_i + 1$;
3: For each $i \in \{1, \ldots, n\}$, $p_i := \dfrac{f_i}{\sum_{j=0}^{n} f_j}$;
4: **Return** $p_1, \ldots, p_n$.

Let $P = \{I_1, I_2, \ldots, I_7\}$ a population (a set of chromosomes as defined previously), and respectively their fitness evaluations $f_1 = 100$, $f_2 = 50$, $f_3 = 60$, $f_4 = 160$, $f_5 = 30$, $f_6 = 40$, $f_7 = 10$, by applying the algorithm above we obtained respectively the proportions $p_1 = 0.08$, $p_2 = 0.115$, $p_3 = 0.085$, $p_4 = 0.15$, $p_5 = 0.26$, $p_6 = 0.16$, $p_7 = 0.15$; with which, we constructed the wheel in Figure 5; after that, one "random" position in the wheel is generated at a time to pinpoint a chromosome to be selected. For instance, the value 0.55 (and any value between 0.43 and 0.69) would select the chromosome I_5. Furthermore, there are many variants of this operator, we name the stochastic universal selection (SUS) introduced by James Baker [4], this operator selects simultaneously multiple chromosomes by choosing a given number of equally spaced values, as it is shown in Figure 5, the spinners indicates to select three chromosomes I_3, I_5, I_7. This operator has an advantage over the roulette wheel selection, in case we have a chromosome that dominates the population (in terms of its proportion in the wheel), the latter can be selected excessively.

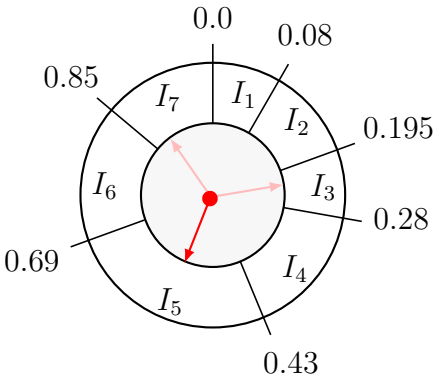

Figure 5: Roulette wheel selection.

As it is mentioned earlier, the selection operator is chosen according to a given probability p_s. In practice, we recommend the values in [0.25, 0.3] as to set an equilibrium between a further exploitation of the current best fitted chromosomes and diversification illustrated as the exploitation of potential "partially fitted" chromosomes, that is, by altering between elitist and roulette wheel selection operators.

5 Cryptanalysis scheme

Commonly, parallel computing is considered as a means to improve the performance of algorithms that have high computational cost (reduces the execution time), as in the process of solving a *NP-hard* problems, in which recourse is often made to metaheuristics. One of the most studied yet effective metaheuristics is Genetic Algorithm. The parallelization of this latter is a technique used to deal with large instances, aiming not only to reduce the computation time but also to improve the quality of solutions. Consequently, it induces higher effectiveness compared to sequential GAs, especially, by the arrival of cooperative multi-search models. In this study, the intuitive approach of parallelization is to carry out a search (attack) on each block separately; however, the employed approach is a distributed attack preformed on each block, and enhanced with a communication strategy among parallel processing elements (PEs) known as the migration in PGA, where the latter is established in order to provide a cooperation among parallel PEs. However, since search elements have different objectives, the use of the migration operator, which we will elaborate later, is justified by an exploitation of the public key's sensitivity. Let Φ a function that operates on blocks of plaintext, such that $\Phi(m_i) = c_i / \sum_j b_j$. Since in practice, the obtained ciphertext c_i can be colossal for large instances; for example, one block of ciphertext of length 64 usually exceeds 10^{20}, so the function Φ is used to reduce the value of ciphertext to an approximated value in $[0, 1]$. In order to study the sensitivity of the public key, we construct a subset of chromosomes from $N(m_i)$, the neighborhood of the optimal solution of the i^{th} block m_i, and than we compare their fitness in different environments (i.e., fitness in each block attack). In practice, we used a set $J \subset E = \{m \,|\, 0 < d(m, m_i) \leq 3\} \subset N(m_i)$, with $d(x\,, y)$, $x, y \in \{0, 1\}^n$, is the Hamming distance between x and y. The set J is obtained by applying, randomly, one to three bit modification on m_i. We observe that, for any block of plaintext m_j $(j \neq i)$, there exists $m \in N(m_i)$, such that:

$$|\Phi(m) - \Phi(m_j)| < |\Phi(m) - \Phi(m_i)|.$$

In other words, a slight modification in a chromosome can change the environment in which it adapts, confirming that we can extract a near optimal solution for a certain block attack from other block's current population. Moreover, it is quite possible even for the most adapted chromosomes to induce adapted solutions for other environments. Figure 6 shows a plaintext $m = m_1 m_2$ and a set of chromosomes in $N(m_1)$ the neighbourhood of m_1 the first block's plaintext, all represented by their approximated fitness values Φ.

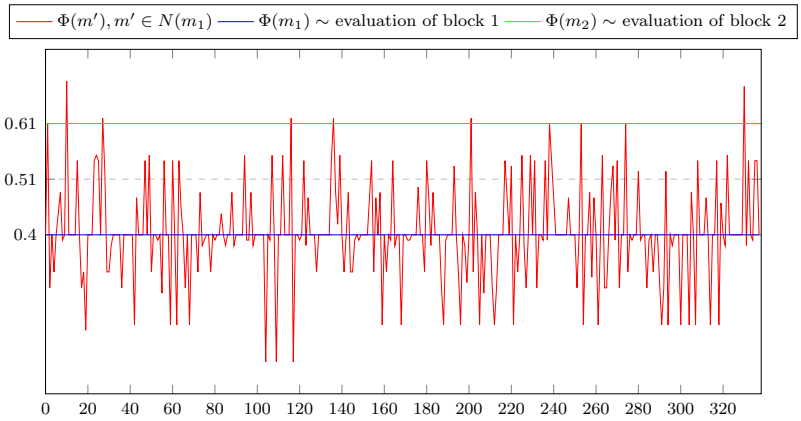

Figure 6: Illustration of the public key sensitivity.

On this basis, we can circumstantiate the importance of establishing a communication link among GAs, illustrated as a migration operator that consists of sending chromosomes that verify the migration condition from a population to another, through connections along all GAs. For each block $j \in \{1, \ldots, k\}$ the attacking process is summarized in the Algorithm 2.

Regarding the migration operator, we have chosen a state dependent operator that proceeds as follows: we evaluate each chromosome in P_j, the current population of the block i, according to their fitness in the rest of the blocks $\{1, \ldots, i-1, i+1, \ldots, k\}$, and than we compare each of them with the best fitted chromosome in the current populations $\{P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_k\}$ (compared with the best solution in the current population, which is not necessarily the best found solution). In other words, let $m \in \{0,1\}^n$ and $f_i(m) = |c_i - \sum_{j=1}^{n} b_j x_j^i|$, the fitness function associated to the i^{th} block; we verify for each bloc $i \in \{1, \ldots, k\}$, if there exists a chromosome $m' \in P_i$ that satisfies:

$$\forall\, m \in P_j,\ f_j(m') < f_j(m),\ j \in \{1, \ldots, i-1, i+1, \ldots, k\},$$

then the chromosome m' is sent to P_j, the current population of the block j, in which it will replace the 'worst' chromosome in the recipient population. However, regarding that the blocks has different objectives, the migration operator is applied before the selection operator, in order to exploit the unfitted chromosomes risking to be removed in the process of selection.

In Figure 7, we put forward a summation of the attacking scheme presented in the previous sections.

Algorithm 2 Parallel Genetic Algorithm (block j)

Begin
Initialize the Number of Generations $nb := 0$;
Generate the initial population P_j^0;
Evaluate chromosomes in P_j^0;
while (plaintext m_j not found) **do**
Select parents form P_j^{nb} and Apply the crossover operator according to p_c;
Apply the mutation operator according to p_m;
Apply an improving heuristic;
Evaluate the chromosomes in P_j^{nb};
if (Migration condition is satisfied) **then**
Send the concerned chromosomes to their new population;
endif
if (Receiving chromosomes) **then**
Remove the 'worst' chromosomes and replace them with the received ones;
endif
Apply a selection operator chosen according to p_s;
Increment nb;
endwhile
End

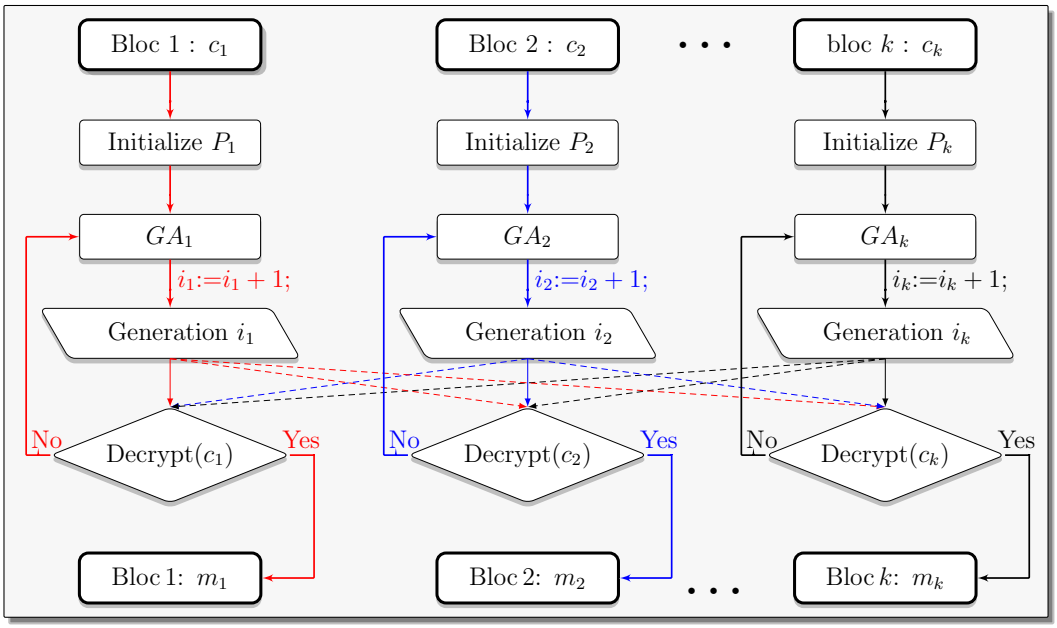

Figure 7: Summation of the cryptanalysis technique.

6 Computational results

In this section, we present the numerical results we have obtained on the cryptanalysis of the Merkel-Hellman cipher using the scheme presented in the previous sections. The implementation was achieved using Java SE platform, in which the parallelization is con-

cretized on a single machine via multithreading. The computational time is measured in terms of CPU time of a PC with the processor of Intel® Core™ i7-5600U, 2.60GHz equipped with 8GBs of RAM. The following experiments consist of studying the impact of the parameters variations on the scheme's performances, while the concerned parameters are: the population size (while the crossover probability p_c is fixed at 0.8), mutation probability p_m, selection probability p_s and the heuristic's number of iterations. We must point that in the experimentation, all the parallel attacks (GAs) are applied with the same parameters values, anywise, the aim is to spot suitable parameters values for a given key size n, and study the effect of these parameters on the scheme. To achieve this, in each experimentation we tested 100 random instances for each key length, where all plaintexts are coded in 8 bits printable ASCII; however, all the obtained results are adduced according to different key lengths $n \in \{8 \times i \,|\, 3 \leq i \leq 14\}$. The following table summarizes the performance of the proposed scheme in statistical results (minimum, maximum, median and average) according to different key lengths, showing the execution time and the number of generations, for successful attacks on one and two blocks of ciphertext.

n	k	Execution time (ms)				Number of generations			
		min	max	median	average	min	max	median	average
24	1	11	4309	47	760	1	1498	14	258
	2	12	3209	63	293	1	1283	9	84
32	1	25	855	281	367	5	341	111	129
	2	31	4511	344	1060	6	1261	81	308
40	1	47	17530	890	1776	10	1958	128	321
	2	141	20481	1384	2922	41	4233	291	507
48	1	125	46645	5039	7853	13	6388	487	1040
	2	2261	39368	6766	8711	333	7386	932	1337
56	1	149	49926	7685	11724	16	7829	702	1077
	2	515	35942	7462	9752	76	7519	1186	1764
64	1	140	82617	8827	13716	11	17385	950	2070
	2	1964	77429	9480	17346	234	11341	739	1738
72	1	1412	107863	37285	78302	80	6341	1213	2033
	2	8412	116903	36016	42523	222	9835	1936	3168
80	1	1237	661800	71919	102075	90	54576	5893	8381
	2	2970	392536	65405	87398	87	25908	4419	6084
88	1	790	278324	101874	139909	49	25558	5418	8385
	2	2265	332563	128968	131283	149	27008	10061	10285
96	1	40435	1874332	220279	424788	3591	80321	10560	19542
	2	33613	1114694	243148	337441	2343	65965	18254	23310
104	1	102788	1770714	739343	780345	1905	58271	17467	20447
	2	161213	2032836	786615	846839	6070	90594	23771	31351
112	1	38357	2595738	1335045	1119082	1294	52353	23584	25471
	2	27822	3695401	1356234	1308807	429	69124	24202	25427

TABLE 1. The proposed scheme performance.

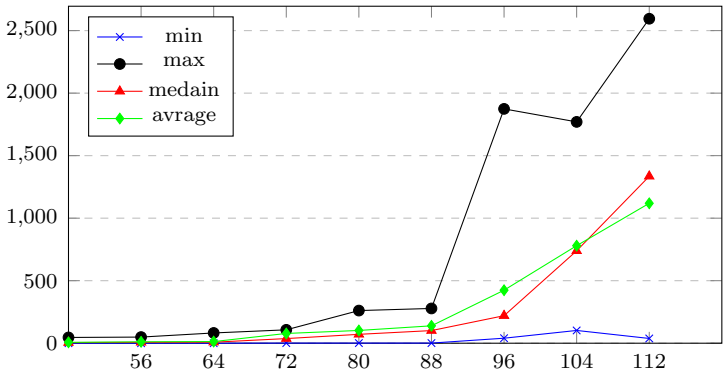

Figure 8: Summation of the scheme's performance.

We tested the proposed scheme against variations of k the number of blocks, by running three instances of size $k \in \{3, 4, 5\}$ of different key size (see Table 2).

	Execution time (ms)			Nb. of generations		
k	3	4	5	3	4	5
$n = 32$	1321	609	2480	583	71	507
40	858	3821	1487	233	1211	214
48	6145	8054	6795	842	1726	817
56	8895	21499	5807	947	2942	429
64	13098	20267	27364	1120	1742	1708
72	74945	98733	233804	5884	4984	20244
80	129658	194249	173455	6521	7542	5318
88	89590	180732	307455	5486	3016	2681
96	177154	97566	116490	11739	3567	656

TABLE 2. Performance of the *PGA* attack against the number of blocks.

The aim of the following experiment is to observe the resemblance ratio by comparing the best found solution to the plaintext for either successful and unsuccessful attacks. Table 3 shows the obtained resemblance ratios for different kay lengths, where the used instances are of length 2 to 4 blocks and the execution time was limited to 1800 seconds.

Instance	*I*1	*I*2	*I*3	*I*4	*I*5	*I*6	*I*7	*I*8	*I*9	*I*10	*Avrage*
$n = 32$	1	1	1	0,44	1	0.66	1	1	1	0.60	0.87
40	1	1	1	1	1	0.43	1	1	0.42	0.50	0.84
48	1	1	1	1	0.40	1	0.46	1	0.52	1	0.88
56	1	1	0.63	0.66	0.59	1	1	0.52	1	1	0.84
64	1	0.52	0.56	1	0.48	1	0.50	1	0.64	1	0.77
72	0.72	1	1	0.67	1	0.95	0.72	0.43	1	0.65	0.81
80	1	0.89	0.63	1	0.81	1	0.79	1	1	0.38	0.85
88	1	0.65	0.56	0.75	1	1	0.58	1	0.53	1	0.81
96	1	0.52	1	1	0.78	0.56	1	1	0.46	1	0.83

TABLE 3. Resemblance ratio for *PGA* Attack.

Tables 4 and 5 resumes the obtained results regarding the effect of the population size and the mutation probability on the performance of the scheme, adduced in the aver-

age execution time and their associated number of generations required for a successful decryption.

	Execution time (ms)					Number of generations				
Pop. size	20	40	80	100	200	20	40	80	100	200
$n = 32$	293	134	540	1228	1420	38	14	43	50	31
40	1119	334	789	768	2278	159	24	47	28	64
48	7726	5511	3671	2122	6854	664	497	122	99	49
56	7230	8478	6775	11282	21717	677	465	433	520	399
64		17692	6489	5128	28328		1212	245	196	519
72		75397	64266	26726	29117		3206	1603	832	647
80		86263	56250	16937	33820		3760	2295	441	343
88			150157	80850	110573			4966	891	1571
96			260523	270235	161947			9483	7991	4655
102				97101	778023				57194	29132
112				1883678	1508266				32847	23561

TABLE 4. Effect of the population size.

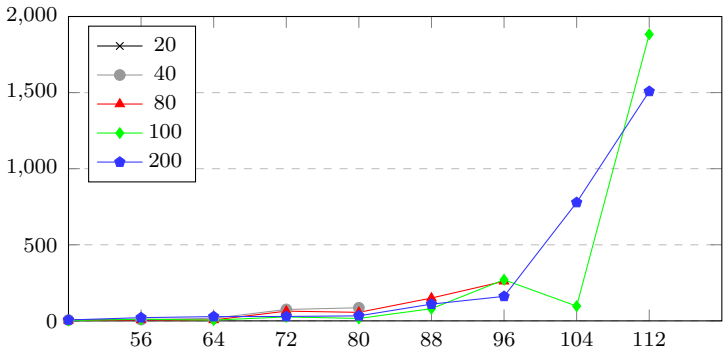

Figure 9: Effect of the population size.

	Execution time (ms)			Nb. of generations		
p_m	0.01	0.1	0.2	0.01	0.1	0.2
$n = 56$	2326	6895	9355	198	340	223
64	18669	7592	19351	142	361	319
72	30116	24249	35708	7091	1153	362
80	27060	48024	77951	1496	2049	928
88	177154	97566	116490	11739	3567	656
96	89590	180732	307455	5486	3016	2681
104	163024	446651	1085037	9483	8129	11118
112	1401182	1269385	1834316	28560	27352	33792

TABLE 5. Effect of the mutation probability.

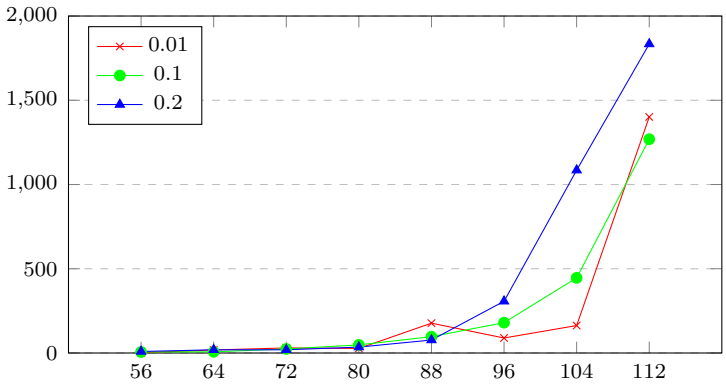

Figure 10: Effect of the mutation probability.

Table 6 shows the results related to a conducted experiment aiming to observe the effect of the selection operator, since this operator has a major impact on the evolution of the algorithm (see Figures 11 and 12), and on both quality of results and execution time. For this purpose, we use p_s the probability that determines the selection operator in each iteration of the algorithm among the roulette wheel selection and the elitist selection.

	Execution time (ms)					Nb. of generations				
p_s	0	0.3	0.5	0.8	1	0	0.3	0.5	0.8	1
$n = 56$	13356	7050	705	2202	1975	554	263	22	63	111
64	36911	28108	41457	21429	31751	2396	1635	423	353	2060
72	53259	46138	28328	24080	55448	2396	1635	423	353	2060
80	71585	62400	124252	81368	107396	2023	2539	2033	1797	2627
88	144173	93195	165704	127010	247476	4829	3651	6637	6051	4316
96	619685	142769	244599	198579	233760	11154	3404	2302	2025	1806
104	793287	269490	418359	280378	896040	5669	3577	4186	2609	9149
112	3695401	908263	1530486	1408183	2202185	69124	1435	24821	28674	21263

TABLE 6. The impact of p_s variations on the scheme's preference.

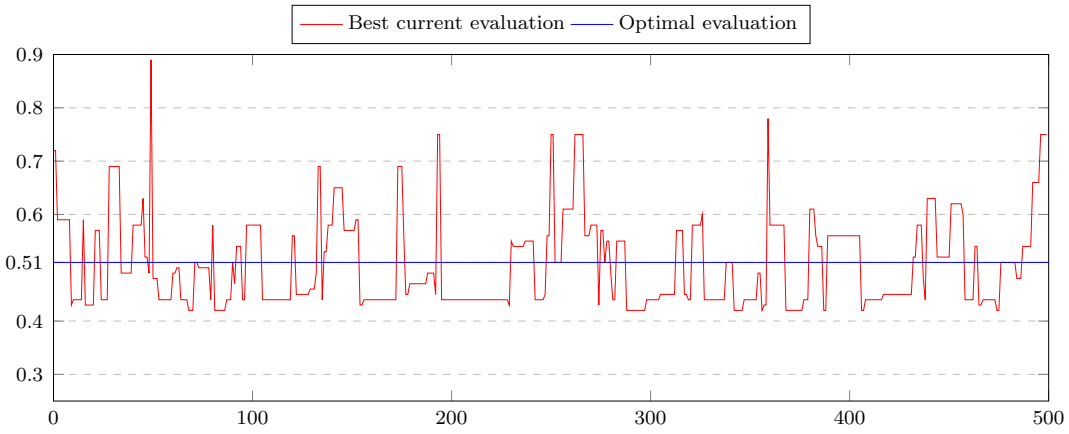

Figure 11: Enrollment of one block attack with $0 < p_s < 0.5$.

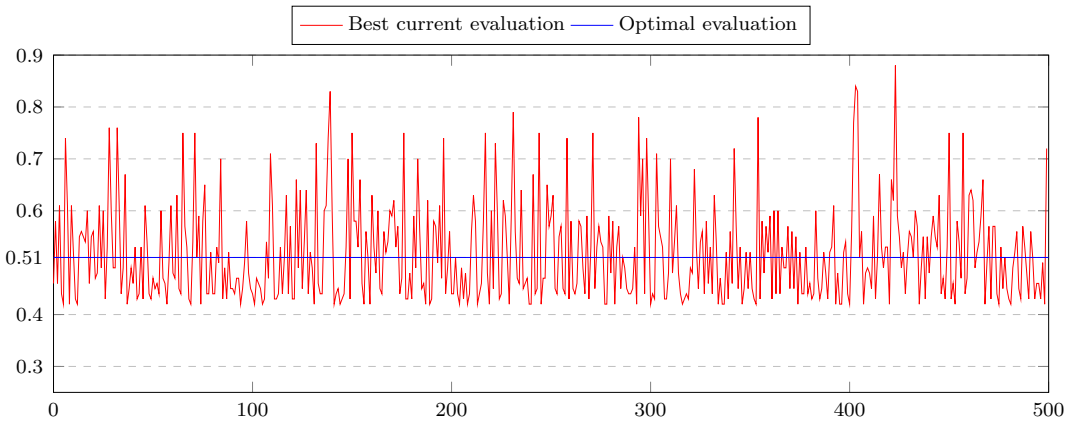

Figure 12: Enrollment of one block attack with $0.5 < p_s < 1$.

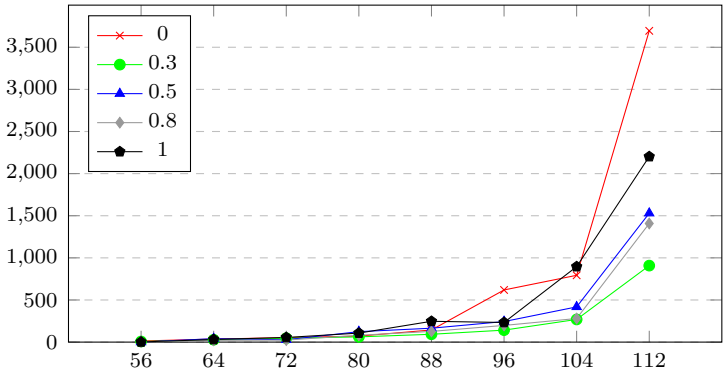

Figure 13: Effect of the selection probability p_s.

As it is mentioned in Section 4, an improving heuristic has been integrated in the GA search process, therefore, we've conducted an experiment aiming to observe the effect of the heuristic's parameter on the scheme's performance. For this, we fixed the number of chromosomes that goes through the heuristic in each GA iteration to one third of the current population size ($\frac{pop.size}{3}$), where these chromosomes are selected randomly from the current population. The experiment consists of variating the heuristic's number of iterations, the following table shows the obtained results for different values ($nb.iter \in \{i \times 100 | 1 \leq i \leq 5\}$).

Comparison with existing attacks. As it is mentioned in the introduction, lattice reduction attack was proposed in [19], using the *LLL* algorithm, presented in [8] by *Lenstra, Lenstra and Lovász*. Analogous to our proposed scheme, this method can also be viewed as a form of heuristic search [14]. We present a comparison with the *LLL* reduction attack, by running instances (public key $\{b_i\}_{1 \leq i \leq n}$, plaintext m and its associated ciphertext c), identified by the paramaters n, δ and p, where, n is the public key size, δ the density of the knapsack (public key) [16], and p is the proportion of ones in the plaintext m;

$$\delta = \frac{n}{\log_2 \max\limits_{1 \leq i \leq n} b_i}, \qquad p = \frac{\sum_i x_i}{n}.$$

Furthermore, throughout this experiment, we used two lattice bases, the *Lagarias-Odlyzko*

	Execution time (ms)					Nb. of generations				
Nb.iter	100	200	300	400	500	100	200	300	400	500
$n = 40$	1016	971	1659	6873	5419	142	107	172	496	274
48	5445	2684	840	566	1840	967	178	75	31	70
56	8626	6306	5171	6876	3396	1393	582	359	312	65
64	13315	7651	1860	3118	6945	1958	901	204	176	273
72	36068	40622	35194	21331	16239	4377	4093	1581	1016	382
80	97361	193045	267799	123560	149724	5832	20946	22451	7637	10992
88	179386	119915	107090	142648	203428	9410	15788	6761	10487	15331
96	387153	328411	684342	239163	168334	37303	24743	40261	11541	18673
104	929999	450031	529709	757108	846461	47277	14494	13075	23874	20434
112	1833462	1938746	797602	896392	1412702	33235	47957	46811	61963	42144

TABLE 7. The effect of the heuristic number of iterations.

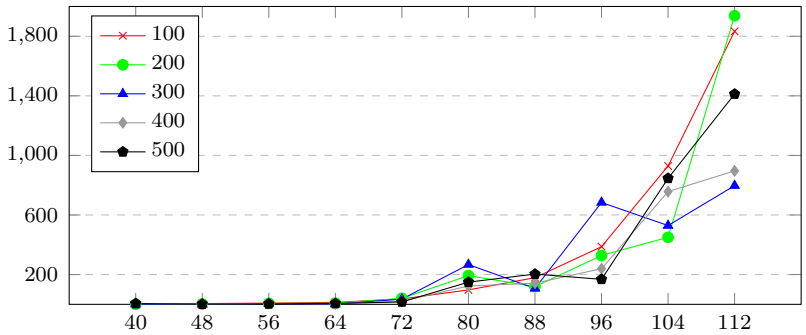

Figure 14: The effect of the heuristic number of iterations.

basis [17], and *Coster, Joux, LaMacchia, Odlyzko, Schnorr, and Stern (CJLOSS)* basis [18]. For this experimentation, we used the *SageMath* function for *LLL* algorithm [20], [15]. Table 8 an 9 present the results of comparison between the *LLL*-based attack and the proposed scheme, obtained by running respectively: 50 instances per density value and 44 instances. The aim of the following experiment is to observe the performance of the proposed scheme against the variation public key density.

δ	*Proposed scheme*	*LLL reduction*
$[0.6, 0.7[$	0.648	0.420
$[0.7, 0.8[$	0.583	0.25
$[0.8, 0.9[$	0.521	0.271
$[0.9, 1.0[$	0.632	0.183

TABLE 8. Success ratios of *LLL reduction* and the proposed scheme.

Throughout the comparison, the *LLL* algorithm attack has shown lower computational cost than the proposed scheme, which is induced by its differences in data processing approaches and its objectives, which have an important influence on the results quality. In contrast, the success ratio of *LLL* algorithm is 29%, while the proposed scheme has a success ratio of 60%, attaining a significant difference of 31%.

Instance			Cryptanalysis method	
n	δ	p	*Proposed scheme*	*LLL reduction*
32	0.627	0.375	✓	✓
	0.711	0.343	✓	✗
	0.842	0.406	✗	✗
	0.914	0.250	✓	✓
40	0.634	0.634	✓	✗
	0.769	0.525	✓	✗
	0.869	0,425	✓	✓
	0.930	0.500	✓	✗
48	0.648	0,416	✓	✓
	0.716	0,625	✓	✓
	0.800	0,479	✗	✗
	0.906	0.542	✓	✗
56	0.651	0,268	✗	✗
	0.778	0,500	✗	✓
	0.875	0,464	✓	✗
	0.933	0,267	✓	✗
64	0.680	0,250	✓	✓
	0.753	0.531	✓	✗
	0.878	0.266	✓	✓
	0.941	0.438	✗	✗
72	0.637	0.486	✓	✗
	0.758	0.208	✗	✗
	0.878	0.472	✓	✗
	0.941	0.458	✓	✗
80	0.620	0.187	✓	✗
	0.721	0.412	✗	✗
	0.889	0.275	✗	✗
	0.963	0.475	✗	✗
88	0.651	0.443	✗	✗
	0.745	0.50	✓	✗
	0.862	0.420	✗	✗
	0.977	0.454	✓	✗
96	0.676	0.218	✓	✗
	0.701	0.395	✗	✓
	0.857	0.427	✓	✓
	0.969	0.468	✗	✗
104	0.630	0.519	✓	✗
	0.717	0.298	✓	✓
	0.838	0.279	✗	✗
	0.981	0.403	✓	✗
112	0.674	0.473	✓	✗
	0.770	0.519	✓	✓
	0.854	0.375	✗	✗
	0.971	0.384	✓	✗

TABLE 9. Comparison between *LLL reduction* and the proposed scheme.

(The symbols ✓ and ✗ respectively refer to successful and unsuccessful attacks)

The design of metaheuristic based attacks goes in general through two important steps. First, the search algorithm needs to be adapted properly to the problem (define a binary relation between elements of the search space, and operator(s) of the metaheuristic, etc.). In the case of GA based attack, this was achieved in [7]. And then investigate further improvements on the basic adaptation (cooperative schemes, hybridization, etc.), in order to achieve high quality results or extend its efficiency to larger instances. Regarding the comparison of the PGA attack with similar attacks, we mention the PSO based attack proposed in [24]. It is shown that the real time taken by MBPSO, in case of knapsack of size 40 is 1.5 hour, while BPSO taking 2 hours. Compared with the suggested PGA attack, we have obtained better results (see Table 1).

Results discussion and comments.

- The results in Table 1 shows a minor diffeIUrence in both average and median decryption time among attacks on one and two blocks, which, illustrates the efficiency of the proposed parallel approach, even for large instances.
- Only the parameter's values that manifest consistency has been included in the previous tables, which explains the blank cells in Table 4.
- The scheme's performance has shown a considerable sensitivity towards the parameters p_m and p_s (see Tables 5 and 6). Regarding the selection operator the recommended value as to set the required equilibrium is $p_s \approx 0.3$, which gives the elitist selection operator the preponderance; while according to the experimentations, the suitable value for $p_m \approx 0.1$.
- The integrated heuristic has an important role in the global optimization process (decryption), nevertheless, as it is shown in Table 7, an overloaded heuristic can delay the process of decryption, in other words, the cost of the local optimization can emerge, in case of an encumbered parameters.
- The fact that the LLL algorithm is known to be effective against low density knapsack problem, according to the results in Table 8 and 9, the proposed scheme shows no sensitivity towards the knapsack density δ.
- Regarding the attack [26] against knapsack with density close to 1, further work needs to be done, to evaluate the efficiency of lattice reduction attacks, since the assessment of knapsack cryptosystems in general against lattice reduction algorithms can not be restrained to the knapsack density [30].
- We must point that the suggested attacking scheme, can be used to enhance any other metaheuristic based attack (PSO, TS, etc.), by replacing Genetic Algorithm. While their performance against practical instances remains to be studied.

7 Conclusion

In this paper we present cryptanalysis scheme based on Genetic Algorithm, adapted to break the Merkle-Hellman cipher; regarding that in this latter a plaintext is ciphered in blocks, the attacking approach is essentially an adapted parallelization of multiple Genetic Algorithms, aiming to decrypt a ciphertext in full. Furthermore, the scheme is enhanced with a deliberate cooperation among the search entities (GAs) via the migration operator, and each GA has a distinct target solution (single block plaintext). We show that the migration of solutions can be useful under a given condition. Finally, we conclude with some experimental results, illustrating the scheme's performance in regards to its parameters.

References

[1] M. Hellman and R. Merkle. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Transactions on Information Theory*, 24(5) : 525-530, September 1978.

[2] M. Hellman W. Diffie. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6) : 644-654, 1976.

[3] R. Karp. Reducibility among combinatorial problems. *Miller R.E. & Thatcher, J.W. (eds.) Complexity of Computer Computations*, Plenum Press, New York, page 85-103, 1972.

[4] Colin R. Reeves, Jonathan E. Rowe. *Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory*. Kluwer Academic Publishers, Dordrech : 2003.

[5] B. Martin. *Codage, cryptologie et applications*. Presses polytechniques et universitaires romandes, Lausanne : 2004.

[6] A. Shamir, *A polynomial time algorithm for breaking the basic Merkle Hellman cryptosystem*, IEEE Transactions on Information Theory, vol. IT-30, no. 5, pp. 699-704, September 1984.

[7] R. Spillman , *Cryptanalysis of knapsack ciphers using genetic algorithms*. Cryptologia, 367-377, 1993.

[8] Lenstra, A.K. and Lenstra, H.W.jun. and Lászlo Lovász, *Factoring polynomials with rational coefficients*, Mathematische Annalen, 515-534, 1982.

[9] S.N. Sinha, S. Palit, M.A. Molla, A. Khanra, M. Kule, *A cryptanalytic attack on Knapsack cipher using Differential Evolution Algorithm*, Recent Advances in Intelligent Computational Systems (RAICS), IEEE, 317-320, 2011.

[10] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley: Reading, MA, 1989.

[11] Mitchell, Melanie. *An Introduction to Genetic Algorithms. Cambridge*, MA: MIT Press, 1996.

[12] J. Holland, *Adaption in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press, 1975.

[13] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*, Springer-Verlag, Berlin, Heidelberg, 2008.

[14] M. Stamp, *Information Security: Principles and Practice*, Wiley-Interscience, 2005.

[15] A. McAndrew, *Introduction to cryptography with open-source software*, CRC Press, 2011.

[16] D. L. Kreher and D. R. Stinson, *Combinatorial Algorithms: generation, enumeration and search*, CRC Press, 1999.

[17] J.C Lagrias and A.m Odlyzko. *Solving low-density subset problems*. J. Assoc. Comp. Mach., 1985.

[18] M.J. Coster, A. Joux, B.A. LaMacchia, A.M. Odlyzko, C.P Schnorr, and J. Stern. *An improved low-density subset sum algorithm*. Computational complexity 2, 1992.

[19] L.M.Adleman. *On beaking generalized knapsack public key cryptosystems*. ACM, Proceedings of 15th STOC, 1983.

[20] W. Stein and al. *Sage Mathematics Software (Version 4.2.1)*. The Sage Development Team, 14th November 2009. `http://www.sagemath.org`.

[21] S. Palit, S. Sinha, M. Molla, A. Khanra, M. Kule, *A cryptanalytic attack on the knapsack cryptosystem using binary firefly algorithm*. 2nd International Conference on Computer and Communication Technology (ICCCT), IEEE, 428-432, 2011.

[22] T. Mandal, M. Kule, *An improved cryptanalysis technique based on Tabu Search for Knapsack cryptosystem*. International Journal of Control Theory and Applications, 16(9): 8295-8302, 2016.

[23] P. Garg, A. Shastri, D.C. Agarwal, *An enhanced cryptanalytic attack on Knapsack Cipher using Genetic Algorithm*. World Academy of Science, Engineering and Technology, International Science Index 12, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 1(12), 4071 - 4074, 2007.

[24] A. Jain , N.S. Chaudhari, *Cryptanalytic Results on Knapsack Cryptosystem Using Binary Particle Swarm Optimization*. de la Puerta J. et al. (eds) International Joint Conference SOCO14-CISIS14-ICEUTE14. Advances in Intelligent Systems and Computing, vol 299. Springer, Cham, 2014.

[25] M. Abdel-Basset, D. El-Shahat, I. El-henawy, A. K. Sangaiah, S. H. Ahmed. A Novel Whale Optimization Algorithm for Cryptanalysis in Merkle-Hellman Cryptosystem. Mobile Networks and Applications, 2018, p. 1-11.

[26] C.P. Schnorr, T.Shevchenko, *Solving subset sum problems of density close to 1 by andomized BKZ-reduction*. IACR Cryptology ePrint Archive 2012: 620 (2012).

[27] N. Howgrave-Graham, A. Joux, *New Generic Algorithms for Hard Knapsacks*, Advances in Cryptology-EUROCRYPT 2010, Springer, 235-256, 2010.

[28] K. Koiliaris and C. Xu. *A faster pseudopolynomial time algorithm for subset sum.* In SODA17, 2017.

[29] K. Bringmann, *A near-linear pseudopolynomial time algorithm for subset sum.* In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 1073-1084, 2017.

[30] S.M. Jen, T.L. Lai, C.Y. Lu, J.F. Yang, *Knapsack cryptosystems and unreliable reliance on density.* 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA), 748-754, IEEE, 2012.