Cryptanalysis of Merkle-Hellman Cipher Using Parallel Genetic Algorithm

Nedjmeddine Kantour1 · Sadek Bouroubi1

Published online: 14 February 2019 © Springer Science+Business Media, LLC, part of Springer Nature 2019

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 [4, 5], and [17]). 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 [15], and improved by Coaster and al. [16]. Later on, Schnorr and Shevchenko gave a progressive algorithm for solving the knapsack problem with density close to 1 [24]. On the other hand, metaheuristic based attacks was initiated by Spillman [5], and followed by many other adaptations (see [7, 19–23]). We mention also some exact resolution methods designed for the subset-sum problem, running in pseudo-polynomial time complexity (see [25–27]). 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 [2]. 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 \(\delta\) 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 \(m_i = x_i^1 x_i^2 \ldots x_i^n\), \(x_i^j \in \{0, 1\}\), and let \(C = (c_1, c_2, \ldots, c_k)\) be the correspondent ciphertext, with \(c_i = \sum_{j=1}^{n} b_j x_i^j\), for all \(i \in \{1, \ldots, k\}\). To decrypt \(C\) we first calculate \(d_i = w^{-1} c_i\) for each \(i \in \{1, \ldots, k\}\), secondly, we solve the trivial knapsack \(d_i = \sum_{j=1}^{n} a_j e_i^j\), finally, deducing \(x_i^j\) from \(x_i^j = e_i^{\delta(j)}\).

### 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 \(M^* = m_1^* m_2^* \ldots m_k^*\), \(m_i^* = x_i^1 x_i^2 \ldots x_i^*_n\), \(x_i^* \in \{0, 1\}\), ensuring:

\[
c_i = \sum_{j=1}^{n} b_j x_i^j = 0, \quad \forall i \in \{1, \ldots, k\},
\]

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 \(M^*\) represents its optimal solution; this model is well known as the Multiple Knapsack Problem.

\[
\begin{align*}
\text{Min} (Z) &= \sum_{i=1}^{k} \sum_{j=1}^{n} b_j x_i^j, \\
\sum_{j=1}^{n} b_j x_i^j &\geq c_i, \quad \forall i \in \{1, \ldots, k\}, \\
x_i^j &\in \{0, 1\}, \quad \forall j \in \{1, \ldots, n\}.
\end{align*}
\]

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 \(m_i^*\). 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:

\[
\begin{align*}
\text{Min} (Z) &= |c_i - \sum_{j=1}^{n} b_j x_i^j|, \\
x_i^j &\in \{0, 1\}, \quad \forall j \in \{1, \ldots, n\}.
\end{align*}
\]

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* \([10]\). 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 \([8]\). 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 \([9]\). 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, \(m_i = x_i^1 x_i^2 \ldots x_i^*_n\), \(x_i^j \in \{0, 1\}\), 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 implement the following operations: crossover, mutation, and selection. The crossover operator is defined in GAs as an analogy of the mechanism that allows the reproduction of chromosomes in nature, the aim of this operator is to produce new chromosomes that partially inherit characteristics from its parents, with copying and recombining their genes in a deliberate 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 \(c \in \{2, \ldots, n - 1\}\), here the mating process consists of swapping bits \(c + 1\) to \(n\) of the first parent \(P_1\) with bits \(c + 1\) to \(n\) of the second
parent $P_2$, hence, new chromosomes $C_1$ and $C_2$ are created [5] (Fig. 1).

The second operator is similar to the first one but here we use two cutting points, and new chromosomes are generated by swapping the middle parts of the parents, as it shown in Fig. 2.

The third operator constructs one child chromosome $C$ by alternately choosing random genes from the parents (see Fig. 3), this operator is known as the uniform crossover operator. 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 $f_{\text{max}} := \max_{1 \leq i \leq n} \{ f_i \}$;
2: For each $i \in \{1, \ldots, n\}$, $f_i := f_{\text{max}} - f_i + 1$;
3: For each $i \in \{1, \ldots, n\}$, $p_i := \frac{f_i}{\sum_{j=0}^{n-1} f_j}$;
4: Return $p_1, \ldots, p_n$.

Let $P = \{l_1, l_2, \ldots, l_7\}$ 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 Fig. 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 [3], this operator selects simultaneously multiple chromosomes by choosing a given number of equally spaced values, as it is shown in Fig. 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.

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,

\[
P_1 = [1 \mid 1 \mid 1 \mid 0 \mid 1 \mid 1 \mid 0 \mid 0] \rightarrow C_1 = [1 \mid 1 \mid 1 \mid 0 \mid 1 \mid 1 \mid 1 \mid 1] \]
\[
P_2 = [0 \mid 1 \mid 1 \mid 0 \mid 0 \mid 1 \mid 1 \mid 1] \rightarrow C_2 = [0 \mid 1 \mid 1 \mid 0 \mid 1 \mid 1 \mid 1 \mid 1]
\]

**Fig. 1** One cutting point crossover operator

**Fig. 2** Two cutting point crossover operator

**Fig. 3** Uniform crossover operator

**Fig. 4** Mutation operator
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 \( \phi_1 \) a function that operates on blocks of plaintext, such that \( \phi_1(m) = c_i / \sum_j b_j \). 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 \( \Phi \) 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 then we compare their fitness in different environments (i.e., fitness in each block attack). In practice, we used a set \( J \subset N(m_i) \), 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 \( m = m_1m_2 \) 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 \( \Phi \).

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.

**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 not found) do
        Select parents from and Apply the cross over 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
```
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 then 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 block $i \in \{1, \ldots, k\}$, 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 Fig. 7, we put forward a summation of the attacking scheme presented in the previous sections.

### 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 concretized 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 $n \in \{8 \times i \mid 3 \leq i \leq 14\}$.

Table 1 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 (Fig. 8).

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).

Table 1: The proposed scheme performance

| $n$ | $k$ | 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| 8711| 333  | 333 | 7386 | 932 | 1337|
| 56  | 1   | 149 | 49926| 7685| 11724| 16  | 7829 | 702 | 1077|
|     | 2   | 515 | 35942| 9752| 76   | 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| 139090| 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| 177014| 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|

Fig. 8: Summation of the scheme’s performance
### Table 2  Performance of the PGA attack against the number of blocks

| $k$  | $n = 32$ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|------|----------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|      | 3        | 4  | 5  | 3   | 4  | 5   | 3   | 4  | 5   | 3   | 4  | 5   | 3   | 4  | 5   |
| 3453 | 1321     | 609 | 2480 | 583 | 71  | 507 | 233 | 1211 | 214 | 842 | 1726 | 817 | 947 | 2942 | 429 |
| 348  | 858      | 3821 | 1487 | 233 | 1211 | 214 | 1120 | 1742 | 1708 | 5884 | 4984 | 20244 | 6521 | 7542 | 5318 |
| 356  | 6145     | 8054 | 6795 | 48  | 6145 | 8054 | 6795 | 48  | 6145 | 8054 | 6795 | 48  | 6145 | 8054 | 6795 |
| 364  | 8895     | 21499 | 5807 | 947 | 2942 | 429 | 5486 | 3016 | 2681 | 5486 | 3016 | 2681 | 5486 | 3016 | 2681 |
| 372  | 13098    | 20267 | 27364 | 1120 | 1742 | 1708 | 6521 | 7542 | 5318 | 6521 | 7542 | 5318 | 6521 | 7542 | 5318 |
| 380  | 74945    | 98733 | 233804 | 5884 | 4984 | 20244 | 6521 | 7542 | 5318 | 6521 | 7542 | 5318 | 6521 | 7542 | 5318 |
| 388  | 129658   | 194249 | 173455 | 5884 | 4984 | 20244 | 6521 | 7542 | 5318 | 6521 | 7542 | 5318 | 6521 | 7542 | 5318 |
| 396  | 89590    | 180732 | 307455 | 5884 | 4984 | 20244 | 6521 | 7542 | 5318 | 6521 | 7542 | 5318 | 6521 | 7542 | 5318 |
|      | 177154   | 97566 | 116490 | 11739 | 3567 | 656 | 11739 | 3567 | 656 | 11739 | 3567 | 656 | 11739 | 3567 | 656 |

### Table 3  Resemblance ratio for PGA attack

| Instance | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $I_10$ | Average |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|--------|----------|
| $n = 32$ | 1     | 1     | 1     | 0.44  | 1     | 0.66  | 1     | 1     | 1     | 0.60   | 0.87     |
| 40       | 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     | 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     |

### Fig. 9  Effect of the population size

### Fig. 10  Effect of the mutation probability
### Table 4  Effect of the population size

| Pop. size | n = 32 | 40 | 48 | 56 | 64 | 72 | 80 | 88 | 96 | 102 | 112 |
|-----------|--------|----|----|----|----|----|----|----|----|-----|-----|
|           | 20     | 40 | 80 | 100| 200| 20  | 40 | 80 | 100| 200 | 200 |
| Execution time (ms) | 293   | 1119| 7726| 7230| 17692| 75397| 86263| 150157| 260523| 97101| 1883678 |
| Number of generations | 32    | 768 | 3671| 6775| 5128| 64266| 56250| 80850| 270235| 778023| 1508266 |

### Table 5  Effect of the mutation probability

| pm | 0.01 | 0.1 | 0.2 |
|----|------|-----|-----|
| Execution time (ms) | 2326 | 18669 | 30116 | 27060 | 177154 | 89590 | 163024 | 1401182 |
| Nb. of generations | 198 | 142 | 7091 | 1496 | 11739 | 5486 | 9483 | 28560 |

### Table 6  The impact of $p_s$ variations on the scheme’s preference

| $p_s$ | 0 | 0.3 | 0.5 | 0.8 | 1 |
|-------|---|-----|-----|-----|---|
| Execution time (ms) | 13356 | 36911 | 53259 | 71585 | 144173 | 619685 | 793287 | 3695401 |
| Nb. of generations | 554 | 2396 | 2396 | 2023 | 4829 | 11154 | 5669 | 69124 |

### Fig. 11  Enrollment of one block attack with $0 < p_s < 0.5$
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 key lengths, where the used instances are of length 2 to 4 blocks and the execution time was limited to 1800 seconds.

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 average execution time and their associated number of generations required for a successful decryption (Figs. 9 and 10).

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 among the roulette wheel selection and the elitist selection (Fig. 13).

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 ($\frac{\text{pop.size}}{3}$), where these chromosomes are selected randomly from the current population. The experiment consists of varying the heuristic’s number of iterations, the following table shows the obtained results for different values ($\text{nb.iter} \in \{i \times 100 | 1 \leq i \leq 5\}$) (Fig. 14).

**Comparison with existing attacks** As it is mentioned in the introduction, lattice reduction attack was proposed in [17], using the LLL algorithm, presented in [6] by Lenstra, Lenstra and Lovász. Analogous to our proposed scheme, this method can also be viewed as a form of heuristic search [12]. 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 parameters $n$, $\delta$ and $p$, where, $n$ is the public key size, $\delta$ the density of the knapsack (public key) [14], and $p$ is the proportion of ones in the plaintext $m$;

$$\delta = \frac{n}{\log_2 \max_{1 \leq i \leq n} b_i}, \quad p = \frac{\sum_{i} x_i}{n}.$$

Furthermore, throughout this experiment, we used two lattice bases, the Lagarias-Odlyzko basis [15], and Coster, Joux, LaMacchia, Odlyzko, Schnorr, and Stern (CJLOSS) basis [16]. For this experimentation, we used the SageMath function for LLL algorithm [13, 18]. Tables 8 and 9 present the results of comparison between the LLL-based attack and the proposed scheme, obtained by running respectively:
Table 7 The effect of the heuristic number of iterations

| Nb.iter | Execution time (ms) | Nb. of generations |
|---------|---------------------|--------------------|
|         | 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   |

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.

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%.

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 [5]. 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 [22]. 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 difference 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 Tables 8 and 9, the proposed scheme shows no sensitivity towards the knapsack density $\delta$.
- Regarding the attack [24] 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 [28].
- 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.

Table 8 Success ratios of LLL reduction and the proposed scheme

| $\delta$     | 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 9  Comparison between LLL reduction and the proposed scheme

| Instance | Cryptanalysis method |
|----------|----------------------|
|           | Proposed scheme | LLL reduction |
| **n**    | **δ**     | **p**     |               |               |
| 32       | 0.627     | 0.375     | ✓             | ✓             |
|          | 0.711     | 0.343     | ✓             | x             |
|          | 0.842     | 0.406     | x             | x             |
|          | 0.914     | 0.250     | ✓             | ✓             |
| 40       | 0.634     | 0.634     | ✓             | x             |
|          | 0.769     | 0.525     | ✓             | x             |
|          | 0.869     | 0.425     | ✓             | ✓             |
|          | 0.930     | 0.500     | ✓             | x             |
| 48       | 0.648     | 0.416     | ✓             | ✓             |
|          | 0.716     | 0.625     | ✓             | ✓             |
|          | 0.800     | 0.479     | x             | x             |
|          | 0.906     | 0.542     | ✓             | x             |
| 56       | 0.651     | 0.268     | x             | x             |
|          | 0.778     | 0.500     | x             | ✓             |
|          | 0.875     | 0.464     | ✓             | x             |
|          | 0.933     | 0.267     | ✓             | x             |
| 64       | 0.680     | 0.250     | ✓             | ✓             |
|          | 0.753     | 0.531     | ✓             | x             |
|          | 0.878     | 0.266     | ✓             | ✓             |
|          | 0.941     | 0.438     | x             | x             |
| 72       | 0.637     | 0.486     | ✓             | x             |
|          | 0.758     | 0.208     | x             | x             |
|          | 0.878     | 0.472     | ✓             | x             |
|          | 0.941     | 0.458     | ✓             | x             |
| 80       | 0.620     | 0.187     | ✓             | x             |
|          | 0.721     | 0.412     | x             | x             |
|          | 0.889     | 0.275     | x             | x             |
|          | 0.963     | 0.475     | x             | x             |
| 88       | 0.651     | 0.443     | x             | x             |
|          | 0.745     | 0.50      | ✓             | x             |
|          | 0.862     | 0.420     | x             | x             |
|          | 0.977     | 0.454     | ✓             | x             |
| 96       | 0.676     | 0.218     | ✓             | x             |
|          | 0.701     | 0.395     | x             | ✓             |
|          | 0.857     | 0.427     | ✓             | ✓             |
|          | 0.969     | 0.468     | x             | x             |
| 104      | 0.630     | 0.519     | ✓             | x             |
|          | 0.717     | 0.298     | ✓             | ✓             |
|          | 0.838     | 0.279     | x             | x             |
|          | 0.981     | 0.403     | ✓             | x             |
| 112      | 0.674     | 0.473     | ✓             | x             |
|          | 0.770     | 0.519     | ✓             | ✓             |
|          | 0.854     | 0.375     | x             | x             |
|          | 0.971     | 0.384     | ✓             | x             |

The symbols ✓ and x respectively refer to successful and unsuccessful attacks.
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.

Acknowledgments The authors thank the anonymous referees for their valuable time to review our manuscript. This work was supported by L’IFORCE Laboratory.

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

References

1. Hellman M, Merkle R (1978) Hiding information and signatures in trapdoor knapsacks. IEEE Trans Inf Theory 24(5):525–530
2. Karp R (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum Press, New York, pp 85–103
3. Reeves CR, Rowe JE (2003) Genetic algorithms - principles and perspectives: a guide to GA theory. Kluwer Academic Publishers, Dordrecht
4. Shamir A (1984) A polynomial time algorithm for breaking the basic Merkle Hellman cryptosystem. IEEE Trans Inf Theory IT-30(5):699–704
5. Spillman R (1993) Cryptanalysis of knapsack ciphers using genetic algorithms. Cryptologia 17(4):367–377
6. Lenstra AK, Lenstra HW Jr, Lovász L (1982) Factoring polynomials with rational coefficients. Math Ann, pp 515–534
7. Sinha S, Palit S, Molla M, Khanra A, Kule M (2011) A cryptoanalytic attack on the knapsack cryptosystem using binary firefly algorithm. In: 2nd international conference on computer and communication technology (ICCCT). IEEE, pp 428–432
8. Garg P, Shastri A, Agarwal DC (2007) An enhanced cryptanalytic attack on Knapsack Cipher using genetic algorithm. World academy of science, engineering and technology, international science index 12. Int J Comput Electrical Automation Control Inf Eng 1(12):4071–4074