STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **16**, December 2026, pp 5506–5537.
Published online in International Academic Press (www.IAPress.org)

Cryptanalysis of the Hellman-Merkle Cryptosystem via Hybrid Metaheuristics Approach

Narimen Hafsi [1,*], Yacine Slimani [2], Djamel Benterki [1]

[1] *Laboratory of Fundamental and Numerical Mathematics, Faculty of Sciences, Setif 1 University - Ferhat Abbas, Algeria*
[2] *Artificial Intelligence Laboratory, Faculty of Sciences, Setif 1 University - Ferhat Abbas, Algeria*

Abstract One of the earliest public-key cryptosystems, the Hellman-Merkle cryptosystem (1978), is based on the subset-sum knapsack problem. This paper attacks it with metaheuristics, comparing ten methods: Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Firefly Algorithm (FA), Whale Optimization Algorithm (WOA), Tabu Search, and five hybrids (GA-FA, GA-PSO, GA-WOA, a novel multi-leader WOA variant WOA-ML, and an FA-WOA-ML hybrid). Hybridizing GA with a swarm- or attraction-based mechanism consistently raises decryption success to 100% at $n = 10$ bits, versus 97.6% for standalone GA; GA-PSO gives the best overall speed/reliability trade-off, while GA-FA explores the most candidates once search effort is measured uniformly across methods. A diagnostic analysis shows that standard (single-leader) WOA suffers an unbounded position blow-up that saturates its sigmoid discretization; replacing its single global-best attractor with a pool of K randomly-sampled leaders (WOA-ML) substantially improves robustness without incurring FA's quadratic cost. Under a time-matched budget at a larger key size ($n = 32$), where GA-FA's $O(P^2)$ cost becomes prohibitive, WOA-ML's cheaper $O(P)$ update reaches non-trivial success while every $O(P^2)$ method fails outright – an explicit crossover between per-generation cost and search quality. At the smaller key sizes used for most of this study ($n \leq 20$), however, GA-FA remains the most reliable method; Tabu Search, included as a population-free baseline, is the cheapest per iteration but the second-least robust, briefly rivaling WOA-ML at very small $n = 32$ budgets before being overtaken. Finally, attempts at $n = 64$ and $n = 128$ confirm that none of the ten methods is a practical attack at cryptographically realistic scale. This paper further evaluates the theoretical justification, generalization to multi-character messages, scalability, parameter sensitivity, and computational complexity of all ten methods.

Keywords Metaheuristics; Genetic Algorithm; Firefly Algorithm; Particle Swarm Optimization; Whale Optimization Algorithm; Tabu Search; Cryptanalysis; Hellman-Merkle Cryptosystem; Knapsack Problem.

AMS 2010 subject classifications 68T20, 90C59, 94A60

DOI: 10.19139/soic-2310-5070-4360

1. Introduction

Cryptology is a scientific discipline that studies methods for encrypting and decrypting data. Data encryption and decryption methods and techniques are usually studied to guarantee information security and secure data transfer. The subfields of cryptology that are mutually unified are cryptography and cryptanalysis. Cryptography focuses on creating new, powerful, and efficient methods for encryption and decryption algorithms using particular mathematical problems as a theoretical basis. On the other hand, cryptanalysis is the study of breaking these encryption methods by identifying possible flaws or vulnerabilities. It locates, exploits, and corrects cryptographic system vulnerabilities. The main difficulty is recovering the plaintext or the encryption key. Metaheuristics are now

*Correspondence to: Narimen Hafsi (Email: narimen19000@hotmail.com). Laboratory of Fundamental and Numerical Mathematics, Setif 1 University - Ferhat Abbas, Algeria.

ISSN 2310-5070 (online) ISSN 2311-004X (print)
Copyright © 2026 International Academic Press

frequently used in cryptanalysis. A number of researchers attempted to use metaheuristics to break the Hellman-Merkle cipher (HM) [1]. Spillman was the first to use a Genetic Algorithm (GA) to crack the Knapsack cipher [2]. Garg et al. [3] extended and re-implemented this attack, showing that GA efficiency is improved by high population size, high crossover probability, and low mutation probability. The Knapsack cryptosystem was attacked using binary Particle Swarm Optimization (PSO) by Abdul-Halim et al. [4] and Jain and Chaudhari [5]. They observed that binary PSO is more effective than the GA used by Spillman [2] and Garg et al. [3]. Furthermore, Sinha et al. [6] showed that differential evolution is a more effective method for attacking the Knapsack cipher than GA. Palit et al. [7] conducted a cryptanalytic attack on the Knapsack cryptosystem using a binary variant of the Firefly Algorithm (FA), showing that FA performs significantly better than GA for this purpose. Abdel-Basset et al. [8] recently proposed an attack based on the Whale Optimization Algorithm, discretizing the search space using a sigmoid function; their results show that the proposed algorithm is more robust and efficient than other algorithms for cryptanalysis of the Hellman-Merkle Knapsack Cryptosystem (HMKC).

A proposal for a parallel GA to break the HMKC was published by Kantour and Bouroubi [9]. Their scheme benefits from deliberate cooperation among search entities through the use of a migration operator. It is also worth acknowledging other notable attacks on the HMKC, such as the Tabu Search-based approach of Mandal and Kule [10] and the genetic-algorithm-based attack of Singh [11].

It is important to position this metaheuristic line of work relative to the established, non-metaheuristic attacks on knapsack-based cryptosystems. Shamir [12] gave the first polynomial-time algorithm breaking the basic (single-iteration) Merkle-Hellman scheme, and Lagarias and Odlyzko [13] showed that lattice-reduction techniques (later refined using the LLL algorithm) solve low-density subset-sum problems – which includes essentially all practical single- and multiple-iteration knapsack cryptosystems – in polynomial time whenever the knapsack density $n/\log_2(\max_i a_i)$ falls below a known threshold. These algebraic attacks exploit the specific trapdoor structure of the private key (the super-increasing sequence and the modular transformation) directly, and are why knapsack-based public-key cryptosystems, including HMC, are considered broken in practice and are not used in deployed systems today. Our motivation is therefore explicitly *not* to propose a competitive practical attack against real-world knapsack cryptosystems, for which lattice reduction remains vastly superior in both speed and success guarantees. Rather, we use HMC cryptanalysis as a controlled, well-understood NP-hard combinatorial testbed – one where ground truth is easy to verify and problem difficulty is easy to tune via n – to study, compare, and improve *general-purpose* metaheuristic search strategies (uniform cost accounting across heterogeneous algorithms, hybridization design, and, in particular, the multi-leader diagnostic and fix for WOA introduced in §3.6) whose value lies in their applicability to combinatorial security problems that lack such exploitable algebraic structure, e.g. steganographic embedding optimization [14] or general black-box combinatorial attack surfaces where no lattice-based shortcut is available. This distinction is revisited in the Discussion (§6) and Conclusion.

GAs are evolutionary algorithms modeled on Darwin's theory of natural evolution [15]. They employ operators such as mutation and crossover, which mimic the processes of natural selection to iteratively evolve candidate solutions to a given problem. Similarly, FA is a more recent nature-inspired metaheuristic drawing inspiration from the flashing behavior of fireflies [16]. This algorithm has shown strong performance in solving a range of optimization problems, including multimodal challenges [16], continuous constrained tasks [17], stochastic test functions and design optimization [18], and general bio-inspired engineering optimization [19]. PSO, introduced by Eberhart and Kennedy [20], takes inspiration from the coordinated movement of birds in flight; its simplicity, and the need to adjust only a few parameters, have made it widely applicable across many fields. Hybrid metaheuristics that combine Genetic Algorithms (GAs) with complementary optimization strategies remain an active research direction due to their ability to balance global exploration and local exploitation. Recent studies published in Statistics, Optimization and Information Computing illustrate this trend. Aly and Alotaibi proposed a Hybrid Butterfly–Grey Wolf Optimization (HB-GWO) algorithm that achieved superior performance for high-dimensional feature selection [21]. Likewise, Meryem et al. developed an optimized GA-Apriori algorithm that improved the efficiency of frequent itemset mining through a GA-based hybrid framework [22]. Closer still to the present problem class, Basheer et al. proposed an improved binary Monarch Butterfly Optimization algorithm for the 0–1 knapsack problem itself, introducing a time-varying transfer function to better balance exploration and exploitation when converting continuous positions to binary solutions [23] – the same binary-discretization

challenge we address for WOA and PSO in Section 2, and directly motivating our own comparison of transfer functions in Section 3.5. These studies further demonstrate that hybrid evolutionary approaches can significantly enhance optimization performance across diverse complex applications. The GA-FA hybrid used as this paper's main building block was itself first proposed by the present authors in the broader context of cybersecurity [24], and has since been applied by us to image steganography [14], where it was used to optimize the embedding process for high imperceptibility; this paper extends that line of work specifically to the cryptanalysis of the Hellman-Merkle cipher, with a substantially deeper experimental treatment.

In this work, we focus on two hybrid algorithms, GA-FA and GA-PSO, and evaluate how well they decrypt the HM cipher. We additionally investigate the Whale Optimization Algorithm (WOA) – already used elsewhere for this problem [8] – together with a GA-WOA hybrid, diagnose a specific weakness of the standard (single-leader) formulation, and propose and evaluate a multi-leader variant (WOA-ML) and an FA-WOA-ML hybrid as a direct empirical test of the theoretical argument of §2.1. We further include Tabu Search [10] as an explicit, population-free baseline against which the cost of every population-based method can be measured.

The remainder of the paper is organized as follows. Section 2 describes the HM cipher and summarizes GA, FA, PSO, WOA and Tabu Search, along with their hybrid counterparts GA-FA, GA-PSO, GA-WOA, the multi-leader WOA-ML, and FA-WOA-ML. Section 3 presents the numerical results, covering the baseline single-character experiments, a multi-character generalization study, a key-size scalability study, the diagnosis and correction of WOA's single-leader weakness, a time-matched comparison at $n = 32$, and exploratory attempts at $n = 64$ and $n = 128$. Section 4 analyzes the sensitivity of each algorithm to its main hyperparameters, and Section 5 provides a theoretical and empirical computational-complexity analysis. Section 6 discusses the results, and Section 7 concludes the paper.

2. Methodology

Despite the knapsack problem's established NP-hardness, the HMC was one of the earliest asymmetric cryptosystems, introduced in 1978 by Merkle and Hellman [1]. Nonetheless, there are some situations – such as a super-increasing sequence – where it can be solved in linear time. To guarantee the transition from a trivial knapsack based on a super-increasing sequence to a hard one (a so-called trapdoor knapsack), Hellman and Merkle introduced an arithmetical transformation. In short, this cryptosystem converts a basic knapsack problem, which functions as the private key, into a general knapsack problem, which functions as the public key [9].
These keys are generated as follows:

- The private key consists of a super-increasing sequence $\{a_1, a_2, ..., a_n\}$, an integer m with $m > \sum_{i=1}^{n} a_i$, an integer $w \in \{1, ..., m-1\}$ which is prime with m, and a permutation δ of the set $\{1, ..., n\}$
- The public key is derived from the private key by calculating $b_i = a_{\delta(i)} w[m]$, for each $i \in \{1, ..., n\}$.
- Let $M = m_1 m_2 ... m_k$ be a simple information written in binary (ASCII code), where $m_i = x_1^i x_2^i ... x_n^i,\ x_j^i \in \{0, 1\},\ i \in \{1, ..., k\}$,
- $\mathcal{C} = (c_1, c_2, ..., c_k)$ be the correspondent ciphered information, with $c_i = \sum_{i=1}^{n} b_j x_j^i,\ \forall i \in \{1, ..., k\}$;
- To decrypt $\mathcal{C}$, we calculate $D_i = w^{-1} c_i[m],\ \forall i \in \{1, ..., k\}$,
- Resolve the trivial knapsack $D_i = \sum_{i=1}^{n} a_j e_j^i,\ \forall i \in \{1, ..., k\}$,
- Deducing x_j^i from $x_j^i = e_{\delta(j)}^i, \forall i \in \{1, ..., k\}, \forall j \in \{1, ..., n\}$.

We are going to make an attack on a ciphered information $\mathcal{C} = (c_1, c_2, ..., c_k)$ in order to find the correspondent clear information, means discovering the information :

$$M^\star = m_1^\star m_2^\star ... m_k^\star, m_i^\star = x_1^{\star i} x_2^{\star i} ... x_n^{\star i}, x_j^{\star i} \in \{0, 1\},\ \forall i \in \{1, ..., k\},$$

guaranteeing:

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

where :

$b_1, b_2, ..., b_n$ is the public key.

Hence, we associate the mathematical model $(\mathcal{P})$ where the plain information $M^\star$ represents its optimal solution, this model is well known as the Knapsack Problem.

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

To obtain the original information we are required to solve numerous NP-hard problems (P), so we attack it with a broad set of metaheuristics: Genetic Algorithm (GA), Firefly Algorithm (FA), Particle Swarm Optimization (PSO), Whale Optimization Algorithm (WOA), Tabu Search, and five hybrids that combine GA with an attraction- or velocity-based exploitation mechanism (GA-FA, GA-PSO, GA-WOA, a multi-leader WOA variant WOA-ML, and an FA-WOA-ML hybrid). §2.1 first justifies, on complexity-theoretic grounds, why an explore-then-exploit hybridization of GA is a natural fit for this problem, before every method is described in turn.

2.1. Why an Explore-Exploit Hybridization of GA? A Complexity-Driven Justification

Problem $(\mathcal{P})$ is a binary subset-sum / 0-1 knapsack feasibility problem, which is NP-hard in the general case [25]. Its search space $\{0,1\}^n$ grows as 2^n, and, unlike the private (super-increasing) knapsack, the public knapsack $\{b_1, \ldots, b_n\}$ has no exploitable arithmetic structure: the modular transformation $b_i = a_{\delta(i)} w \bmod m$ deliberately destroys the super-increasing ordering that would otherwise allow a linear-time greedy decomposition. Two structural properties of this landscape motivate the choice of metaheuristics compared in this work, and in particular why GA-FA and GA-PSO were designed as our primary hybrids, with WOA, GA-WOA, WOA-ML, FA-WOA-ML and Tabu Search included as further points of comparison across the same explore-exploit spectrum (§3).

First, the fitness landscape defined by

$$f(x) = \left| \sum_{j=1}^{n} b_j x_j - c \right|, \qquad x \in \{0,1\}^n, \tag{2.1}$$

is a linear (hence unimodal in the continuous relaxation) but combinatorially rugged function once restricted to $\{0,1\}^n$: single bit-flips can produce arbitrarily large jumps in (2.1) because the b_j grow near-exponentially with j (they inherit the super-increasing growth rate of the private key, transformed by $w \bmod m$). This combination of global linear structure with local ruggedness is precisely the setting in which a two-phase, *explore-then-exploit* strategy is expected to outperform a single-mechanism metaheuristic: a purely explorative method (plain GA, or a diversity-only random search) wastes evaluations on bit combinations that are locally reachable but globally uninformative, while a purely local method (hill-climbing, Tabu Search) risks premature convergence to one of the many spurious local optima created by the ruggedness described above.

Second, GA's crossover operator is well suited to recombining partially-correct bit substrings (schema recombination, in Holland's sense [15]), which is valuable early in the search when large portions of the target subset are still unknown; FA's pairwise attraction mechanism and PSO's velocity-based pull toward personal/global bests are, by contrast, better suited to the later, exploitation-heavy phase where a small number of bits need correcting around an already near-correct candidate. Hybridizing GA with an attraction- or velocity-based mechanism therefore targets the two phases of the search separately, rather than relying on a single operator (mutation alone, in a plain GA) to perform both roles.

This reasoning also explains why GA-FA and GA-PSO were designed as our primary hybrids, pairing GA's exploration with two different exploitation mechanisms (FA's attraction, PSO's velocity pull), rather than with other modern metaheuristics applied elsewhere to Hellman-Merkle cryptanalysis, such as the Whale Optimization Algorithm (WOA) [8] or Tabu Search [10]. WOA's encircling and bubble-net mechanisms are designed for continuous search spaces and require a sigmoid- or hyperbolic-tangent-based discretization step similar to the one we use for PSO (Algorithm 4); it is a valid alternative exploitation operator, but it does not offer an exploration mechanism as combinatorially natural as GA's crossover for recombining candidate subsets. Tabu Search, on the other hand, is a purely local, trajectory-based method (no population, no recombination): it is an efficient exploitation-only baseline, but by itself it inherits the premature-convergence risk described above unless carefully combined with a diversification mechanism, which is again the role GA plays in our hybrids. Rather than leaving this as a purely theoretical argument, we include both WOA (§2.8, together with a GA-WOA hybrid and the multi-leader WOA-ML fix, §2.10) and Tabu Search (§2.2) as explicit empirical baselines, which lets §3 test directly how a single-attractor swarm method and a population-free trajectory method compare against the GA-FA/GA-PSO hybrids, and at what cost. In short, every method compared in this paper occupies a specific point on the same explore-exploit spectrum that the structure of problem $(\mathcal{P})$ calls for; §5 additionally shows that this choice of exploitation mechanism has a first-order, and previously under-reported, effect on computational cost.

2.2. Tabu Search (TS)

Tabu Search was dismissed in §2.1 as a purely local, single-trajectory method that risks premature convergence without a diversification mechanism. We nonetheless include it as an explicit empirical baseline, both because it is used elsewhere for HMKC cryptanalysis [10] and because, having no population at all, it offers the cheapest possible per-iteration cost against which to measure every population-based method in this study.

At each iteration, TS evaluates the full 1-bit-flip neighborhood of the current solution (n neighbors), moves to the best *non-tabu* neighbor – a neighbor obtained by re-flipping a recently-flipped bit is forbidden for a fixed tabu tenure, unless doing so would improve on the best solution found so far (the standard *aspiration criterion*) – and marks the newly-flipped bit tabu. Because a single trajectory can stagnate in a local optimum of this narrow neighborhood well before the iteration budget is exhausted, we add a standard diversification step: if the best-known solution has not improved for a fixed number of iterations, the search restarts from a fresh random solution (Algorithm 1); without this restart, additional iterations bring essentially no further benefit (§3.2).

Algorithm 1 Tabu Search (TS) with diversification restarts

Require: Public key b, ciphertext c, max. iterations G, tabu tenure τ, restart patience ρ
Ensure: Best solution, number of keys searched
$x \leftarrow$ random binary solution; $x^\star \leftarrow x$; $\mathrm{stall} \leftarrow 0$
for $t = 1$ to G **do**
  Evaluate all n single-bit-flip neighbors of x
  Among neighbors not currently tabu (or satisfying the aspiration criterion $f(\cdot) < f(x^\star)$),
  pick the fittest, x', obtained by flipping bit j
  $x \leftarrow x'$; mark bit j tabu for the next τ iterations
  if $f(x) < f(x^\star)$ **then**
    $x^\star \leftarrow x$; $\mathrm{stall} \leftarrow 0$
  else
    $\mathrm{stall} \leftarrow \mathrm{stall} + 1$
  end if
  if $f(x^\star) = 0$ **then**
    return $x^\star$
  end if
  if $\mathrm{stall} \geq \rho$ **then**
    $x \leftarrow$ fresh random solution; clear tabu list; $\mathrm{stall} \leftarrow 0$
  end if
end for
return $x^\star$, total number of keys searched

2.3. Genetic Algorithm (GA)

GA is a randomized global search method that mimics natural evolutionary processes to solve issues. In 1960, Holland and colleagues discovered GA based on the theory of survival and reproduction of the fittest. Since this year, GA has been effectively used to numerous challenging optimization situations and demonstrates its advantages over conventional optimization techniques, particularly when the system being studied has several local optimal solutions. GA is a heuristic technique for optimizing results in a variety of applications. Three operations, selection, crossover, and mutation are used to maximize the parent population of each generation using the input populations, which are represented as chromosomes [15].

The first population is created using a random heuristic: every member of the population is a list of n bits (0 or 1), each drawn independently from a uniform distribution (equal probability of 0 or 1). The crossover operation combines the traits of two parents by choosing a crossover point and exchanging the corresponding tail segments between them, which aids in the creation of the new population. To measure adaptation, a fitness function – given for problem $(\mathcal{P})$ by (2.1) – is defined to evaluate chromosomes according to their performance on the problem at hand.

In the rest of this section we present the GA used to solve the knapsack problem $(\mathcal{P})$ defined above. Starting with the representation of chromosomes: since a chromosome is a binary string, we use the natural representation $m_i = x_1^i x_2^i \dots x_n^i$, $x_j^i \in \{0, 1\}$, where n is the length of the public key. To initiate the algorithm we generate an initial population using a greedy heuristic, which randomly generates a large set of chromosomes and retains a given number of the best-fitting ones, so as to obtain a population of acceptable quality from the outset.

At each generation, parents are selected from the current population according to a mating (crossover) probability P_c; the selected chromosomes are then responsible for generating the next population. This involves three operations: crossover, mutation, and selection. In GAs, crossover is modeled on the natural reproduction process, whereby offspring chromosomes inherit traits from both parents through a controlled recombination of their genes. These newly generated chromosomes are then added to the current population, after which mutation is applied with a probability p_m, typically set below 0.1; this operator works by flipping a short block of genes (typically of length one) to its complement.

To further improve the fitness of the chromosomes in the current population, including the offspring generated in the previous step, a low-cost local-search heuristic is applied before selection: a given number of chromosomes are selected and their genes are perturbed in an attempt to derive a better-adapted version, driving each chromosome toward a local optimum; if the correction succeeds, the improved chromosome replaces its original parent in the population. The final step is selection, which decides which chromosomes are kept as parents for the next generation – and which are discarded as unfit. We alternate between two selection operators, chosen at random with probability p_s. The first is *elitist selection*, which retains the most well-adapted chromosomes for the next generation; this preserves good genes but can lead to premature convergence. The second is *roulette-wheel selection*, a stochastic procedure in which each chromosome is assigned a selection probability proportional to its fitness – analogous to a casino roulette wheel in which better-adapted chromosomes are given proportionally larger slots, rather than the equal-sized slots of a fair wheel, so that fitter individuals are more likely, but not certain, to be selected. Algorithm 2 details how these per-chromosome probabilities p_i are computed from the raw fitness values f_i so that they sum to one, as required for a valid probability distribution over the $[0, 1]$ interval (or, in the circular representation, over an angle of $2\pi p_i$ per chromosome).

The main steps of GA are described as follows:

Step I Randomly create an initial group of solutions appropriate for the problem.

Step II Evaluate the fitness value of every solution in the group.

Step III Generate a new group of solutions by repeating the following steps: (a) select two parents from the group of solutions according to the selection mechanism; (b) crossover the parents to create new offspring; (c) mutate the new offspring.

Step IV If the stopping criteria are met, stop; otherwise, return to Step II.

For which we use Algorithm 2 below:

Algorithm 2 Roulette-wheel selection probabilities

Require: Fitness values $f_1, \ldots, f_n$
Ensure: Selection probabilities $p_1, \ldots, p_n$
$f_{max} \leftarrow \max_{1 \le i \le n} f_i$
for $i = 1$ to n **do**
  $f_i \leftarrow f_{max} - f_i + 1$
end for
for $i = 1$ to n **do**
  $p_i \leftarrow \dfrac{f_i}{\sum_{j=1}^{n} f_j}$
end for
return $p_1, \ldots, p_n$

The selection operator to use at each generation is chosen at random with probability p_s, as described above. In practice, we suggest values of p_s in $[0.25, 0.3]$ as a compromise between diversification – giving a chance to partially-fit chromosomes – and exploitation of the current best-fit chromosomes, by alternating between roulette-wheel and elitist selection.

2.4. Firefly Algorithm (FA)

Fireflies are winged beetles that emit light and flash during the night. This light, which lacks infrared or ultraviolet frequencies and is chemically generated in the lower abdomen, is known as bioluminescence; fireflies use it primarily to attract mates or prey, and as a warning signal to deter predators. Like GA, FA is one of the most widely used swarm-intelligence algorithms for addressing optimization problems. FA, developed by Yang [16], is a metaheuristic inspired by the flashing behavior of fireflies and the phenomenon of bioluminescent communication.

Yang [16] formulated FA based on the following three assumptions:

1. Fireflies are attracted to one another regardless of sex, since they are treated as unisexual.
2. Since attractiveness is proportional to brightness, a less bright firefly is attracted to a brighter one, and this attraction diminishes as the distance between the two fireflies increases.
3. If two fireflies have equal brightness, their movement becomes random. Random walk and firefly attraction are used together to generate new candidate solutions [16, 18, 17]. Since each firefly's brightness is tied to the problem's objective function, fireflies automatically split into smaller subgroups that each converge around a local optimum, which is why FA is well suited to multimodal optimization problems [19, 17].

The main steps of FA are described as follows:

Step I (Initialization): A population of N random fireflies (candidate solutions) is initialized at $K = 0$, where the position of the i-th firefly in d-dimensional space is denoted X_i and represented as $X_i^K = (x_{i1}, x_{i2}, \ldots, x_{id})$.

Step II (Evaluation): Evaluate the fitness value (the light intensity) $I(X_i^K) = f(X_i^K)$ for every firefly $i = 1, \ldots, N$.

Step III (Determination of the best solution): For minimization problems, the firefly with minimum light intensity is the current best solution X_b.

Step IV (Updating firefly positions): For every pair of fireflies $i, j = 1, \ldots, N$: if $I(X_j^K) < I(X_i^K)$, firefly i is attracted to firefly j and its position is updated according to :

$$X_i^{K+1} = X_i^K + \beta_0 \, e^{-\gamma r_{ij}^2} \left(X_j^K - X_i^K \right) + \alpha_K \epsilon_K, \tag{2.2}$$

$$r_{ij} = \left\| X_i^K - X_j^K \right\| = \sqrt{\sum_{l=1}^{d} \left(x_{il}^K - x_{jl}^K \right)^2}, \tag{2.3}$$

where β_0 is the attractiveness at $r_{ij} = 0$, γ is the light absorption coefficient, r_{ij} is the Cartesian distance between fireflies i and j given by (2.3), α_K is a parameter controlling the step size, and ϵ_K is a random vector drawn from a Gaussian (or other) distribution. In the binary implementation used in this paper (Algorithm 3), the continuous move (2.2) is replaced by a bit-wise probabilistic move governed by the attractiveness β, as detailed below.

Step V (Updating the best solution X_b): If any new firefly position X_i^{K+1} improves on the current best solution, i.e. $I(X_i^{K+1}) < I(X_b)$, then $X_b \leftarrow X_i^{K+1}$.

Step VI (Stopping condition): If the stopping condition is satisfied, stop; otherwise, return to Step IV.

A pseudo-code of the FA is illustrated in Algorithm 3:

Algorithm 3 Binary Firefly Algorithm (FA)

Require: Objective function $f(x)$, $x = (x_1, \ldots, x_d)$; light absorption coefficient γ
Ensure: Best solution found X_b

```
Initialize a population of n fireflies x_i (i = 1, ..., n)
Calculate the light intensity I_i at x_i via f(x_i) for all i
while t < MaxGeneration do
    for i = 1 to n do
        for j = 1 to n do
            Calculate the distance r between x_i and x_j
            if I_j < I_i then
                Attractiveness β varies with distance r: β = β_0 e^{-γ r^2}
                Move firefly i towards j in all d dimensions
            end if
            Evaluate new solution and update light intensity I_i
        end for
    end for
    Rank the fireflies and update the current best X_b
    t ← t + 1
end while
return X_b
```

2.5. Particle Swarm Optimization Algorithm (PSO)

PSO is a powerful stochastic optimization technique recognized for its simplicity and small number of parameters, yet it remains highly effective and efficient in tackling complex problems. It is a bio-inspired method applicable to both continuous and discrete optimization. Like GAs and ant colony optimization, PSO is grounded in the principle of self-organization, in which a group of simple agents collaborates to perform complex tasks based on simple local rules. PSO was inspired by Craig Reynolds' model of the flocking behavior of birds and herds of cattle, and was formalized by Eberhart and Kennedy [20] to mimic the social behavior of bird flocks and fish schools. In PSO, particles combine individual and collective knowledge to converge toward the best solution, adjusting their movement much as birds or fish adjust theirs to avoid predators, search for food, or respond to environmental cues. By following these simple movement rules within the solution space, PSO particles gradually converge on a global minimum, which makes the metaheuristic particularly effective for optimization problems involving continuous or, once discretized, binary variables.

The key components of PSO are as follows.

Particle An individual candidate solution that moves through the search space, adjusting its position based on its own experience and that of the swarm.

Position The particle's current candidate solution, i.e. a specific point (or, once discretized, a specific binary string) in the search space.

Velocity The rate and direction of change of a particle's position at each iteration.

Fitness function The function used to assess the quality of a particle's position with respect to the optimization objective.

Global best ($gbest$) The best position found so far by any particle in the swarm, representing the swarm's collective knowledge.

Personal best ($pbest$) The best position found so far by a given particle, representing its own individual experience.

At each iteration, particle i's velocity and position along dimension j are updated according to

$$v_i[j] \leftarrow w\, v_i[j] + c_1 r_1 \left(pbest_i[j] - x_i[j]\right) + c_2 r_2 \left(gbest[j] - x_i[j]\right), \tag{2.4}$$

$$x_i[j] \leftarrow x_i[j] + v_i[j], \tag{2.5}$$

where w is the inertia weight, c_1, c_2 are the cognitive and social coefficients, and r_1, r_2 are independent random values drawn uniformly from $[0, 1]$. Since problem $(\mathcal{P})$ is binary, position update (2.5) is replaced, in our implementation, by the sigmoid-based discretization of Algorithm 4, which converts the continuous velocity into a bit-flip probability.

A pseudo-code of the PSO is illustrated in Algorithm 4:

Algorithm 4 Binary Particle Swarm Optimization (PSO)

Require: Swarm size n, inertia w, cognitive/social coefficients c_1, c_2
Ensure: Global best binary solution $gbest$
for each particle $i = 1$ to n **do**
  Initialize binary position $x_i \in \{0,1\}^d$ randomly
  Initialize velocity v_i randomly within allowed limits
  Evaluate fitness $f(x_i)$; set $pbest_i \leftarrow x_i$, $f(pbest_i) \leftarrow f(x_i)$
end for
$gbest \leftarrow \arg\min_i f(pbest_i)$
while stopping condition not met **do**
  for each particle $i = 1$ to n **do**
    for each dimension $j = 1$ to d **do**
      $r_1, r_2 \leftarrow$ random values in $[0, 1]$
      $v_i[j] \leftarrow w \cdot v_i[j] + c_1 r_1 (pbest_i[j] - x_i[j]) + c_2 r_2 (gbest[j] - x_i[j])$
      $x_i[j] \leftarrow 1$ with probability $\sigma(v_i[j]) = 1/(1 + e^{-v_i[j]})$, else 0 ▷ sigmoid binarization
    end for
    Evaluate $f(x_i)$
    if $f(x_i) < f(pbest_i)$ **then**
      $pbest_i \leftarrow x_i$, $f(pbest_i) \leftarrow f(x_i)$
    end if
    if $f(pbest_i) < f(gbest)$ **then**
      $gbest \leftarrow pbest_i$
    end if
  end for
  if $f(gbest) = 0$ **then**
    return $gbest$
  end if
end while
return $gbest$

2.6. GA-FA Hybrid Algorithm

The GA-FA hybrid algorithm combines the robustness of GA with the attractiveness-based mobility of FA. By directing less-fit individuals toward brighter (better) solutions through a probabilistic attraction model, the approach aims to promote local exploitation once GA has explored the search space broadly. Its key steps are:

- The initial population is sorted by fitness.
- GA operations (elitism, crossover, mutation) are applied.
- An FA-based attraction mechanism guides individuals toward more optimal neighbors.
- This synergy refines the search around promising regions and improves convergence speed.

A pseudo-code of GA-FA is illustrated in Algorithm 5:

Algorithm 5 GA–FA Hybrid Algorithm

Require: Public key b, ciphertext c, population size P, generations G, mutation rate p_m
Ensure: Best solution, number of keys searched

Initialize binary population of size P
for $t = 1$ to G **do**
  Evaluate fitness and sort population
  if best fitness $= 0$ **then**
    return best solution
  end if
  Keep the 10 fittest individuals (elitism)
  while new population size $< P$ **do**
    Select two parents from the top 50 individuals
    Crossover parents $\rightarrow$ two children; mutate each with rate p_m
    Append children to the new population
  end while
  for $i = 1$ to P **do**
    for $j = 1$ to P **do**
      if fitness$(x_j) <$ fitness(x_i) **then**
        $\beta \leftarrow \exp(-(\text{fitness}(x_i) - \text{fitness}(x_j))^2)$
        Move x_i towards x_j bit-wise with probability β (else mutate with rate p_m)
      end if
    end for
  end for
  Update population with the moved individuals
end for
return best solution found, total number of keys searched

2.7. GA-PSO Hybrid Algorithm

The GA-PSO hybrid algorithm combines PSO's fast convergence with the exploratory power of GA's crossover and mutation operators. Its goal is to minimize the fitness gap between the target and the candidate solution in (2.1), so as to improve decryption of ciphertext produced by the HMKC. Its key steps are:

- A binary population is initialized.
- PSO updates each individual's position based on inertia, personal best, and global best, following (2.4)–(2.5).
- GA operations (crossover and mutation) are applied to diversify the population.
- The process continues for a fixed number of generations, or until an optimal solution is found.

A pseudo-code of GA-PSO is illustrated in Algorithm 6:

Algorithm 6 GA–PSO Hybrid Algorithm

Require: Public key b, ciphertext c, population size P, generations G, mutation rate p_m, inertia w, coefficients c_1, c_2
Ensure: Best solution, number of keys searched
Initialize binary population and velocities of size P
Evaluate fitness; set personal bests $pbest$ and global best $gbest$
for $t = 1$ to G **do**
  for each individual $i = 1$ to P **do**
    Update velocity and (sigmoid-binarized) position using PSO rules (Algorithm 4)
    Update $pbest_i$ and $gbest$ if improved
  end for
  Sort population by fitness
  if $f(gbest) = 0$ **then**
    return $gbest$
  end if
  Keep the 10 fittest individuals (elitism, with their $pbest$ and velocities)
  while new population size $< P$ **do**
    Select two parents from the top 50 individuals
    Crossover parents $\rightarrow$ two children; mutate each with rate p_m
    Append children (with reset $pbest$ and random velocity) to the new population
  end while
  Update population, $pbest$, velocities and $gbest$
end for
return $gbest$, total number of keys searched

2.8. Whale Optimization Algorithm (WOA)

WOA [8] is a swarm-intelligence metaheuristic inspired by the bubble-net hunting strategy of humpback whales. Each whale's position is updated, at every iteration, by one of two exploitation mechanisms – *encircling prey* or a logarithmic *spiral bubble-net attack* – or, when a coefficient $|A| \geq 1$, by an exploration mechanism that moves the whale toward a randomly chosen member of the population rather than the current best. Formally, with a linearly decreasing from 2 to 0 over the run:

$$A = 2ar - a, \qquad C = 2r', \qquad p \sim \mathcal{U}(0,1), \tag{2.6}$$

$$X^{K+1} = \begin{cases} X^\star - A \cdot |CX^\star - X^K| & p < 0.5,\ |A| < 1 \text{ (encircling)} \\ X_{\text{rand}} - A \cdot |CX_{\text{rand}} - X^K| & p < 0.5,\ |A| \geq 1 \text{ (exploration)} \\ |X^\star - X^K|\, e^{bl} \cos(2\pi l) + X^\star & p \geq 0.5 \text{ (spiral)} \end{cases} \tag{2.7}$$

where $r, r' \sim \mathcal{U}(0,1)^n$, $l \sim \mathcal{U}(-1,1)^n$, b is a constant defining the spiral shape, and $X^\star$ is the current best whale. Since problem $(\mathcal{P})$ is binary, we maintain, for each whale, a continuous "shadow" position updated by (2.7) and discretize it at every evaluation via the same sigmoid transfer function used for PSO (Algorithm 4). A pseudo-code of WOA is given in Algorithm 7.

Algorithm 7 Binary Whale Optimization Algorithm (WOA)

Require: Public key b, ciphertext c, population size P, generations G, spiral constant b_0
Ensure: Best solution, number of keys searched

```
Initialize continuous shadow positions; discretize via sigmoid; evaluate fitness
X* ← best discretized whale
for t = 1 to G do
    a ← 2 − 2t/G
    for each whale i = 1 to P do
        Draw A, C, p as in (2.6); update the shadow position via (2.7)
        Discretize via sigmoid; evaluate fitness; update X* if improved
    end for
    if f(X*) = 0 then
        return X*
    end if
end for
return X*, total number of keys searched
```

2.9. GA-WOA Hybrid Algorithm

As WOA offers a structurally different exploitation mechanism from FA and PSO – combining an encircling move with a logarithmic spiral bubble-net attack, rather than a single attraction or velocity rule – it is natural to ask whether pairing it with GA's crossover-based exploration is similarly beneficial, and to use this hybrid as a baseline against which the multi-leader fix of §2.10 can later be measured. Mirroring GA-PSO, GA-WOA replaces PSO's velocity update with the WOA update (2.7) while keeping the same elitism/crossover/mutation structure: each generation runs a WOA update pass over the whole population, sorts by fitness, and regenerates all but the 10 fittest individuals via GA crossover and mutation. As with WOA itself, this single-attractor design turns out to inherit the same position-saturation weakness diagnosed below (§3.6), which is precisely why the multi-leader variant is introduced next rather than being paired with GA directly.

2.10. Diagnosing and Correcting a Weakness of WOA: The Multi-Leader Variant (WOA-ML)

Preliminary experiments (§3.6) revealed that standard WOA, as defined above, degrades sharply as n grows, more so than PSO despite superficially having two exploitation mechanisms instead of one. Tracing the continuous shadow positions across generations showed why: because every whale is pulled toward the *same single* best whale $X^\star$, and the spiral term in (2.7) involves an unbounded exponential e^{bl}, the shadow positions blow up geometrically over successive generations (Figure 6), driving the sigmoid transfer function into saturation and effectively freezing most bits regardless of further updates. Bounding the shadow positions to a fixed range (as is standard practice for binary PSO's velocity) alleviates this only partially, because the deeper issue is that *all* whales converge onto a single attractor, which is precisely the single-point-of-failure that FA's all-pairs attraction (§2.3) avoids by construction.

We therefore propose **WOA-ML**, a multi-leader variant in which the single best whale $X^\star$ in (2.7) is replaced, independently for every whale at every generation, by a whale chosen uniformly at random among the current top K individuals (the "leader pool"), in addition to bounding the shadow positions to $[-6, 6]$. This is a lightweight way of injecting FA-like multiple simultaneous attraction basins into WOA's specific (encircling + spiral) move, without paying FA's $O(P^2)$ pairwise cost: computing the leader pool once per generation costs $O(P \log P)$ (a sort), and each whale still performs a single $O(n)$ update, so WOA-ML remains $O(Pn + P \log P)$ per generation, the same order as plain WOA. Algorithm 8 gives the pseudo-code; §3.6–§3.7 report the resulting improvement empirically, including the sensitivity of K itself.

Algorithm 8 Multi-Leader WOA (WOA-ML)

Require: Public key b, ciphertext c, population size P, generations G, leader-pool size K, position bound $B = 6$
Ensure: Best solution, number of keys searched
Initialize continuous shadow positions; discretize via sigmoid; evaluate fitness
$X^\star \leftarrow$ best discretized whale (tracked persistently across generations)
for $t = 1$ to G **do**
  $a \leftarrow 2 - 2t/G$; sort population by fitness; let leader pool = top K individuals
  for each whale $i = 1$ to P **do**
    Draw a leader L uniformly at random from the leader pool
    Draw A, C, p as in (2.6); update the shadow position via (2.7) with $X^\star$ replaced by L
    Clip the shadow position to $[-B, B]$; discretize via sigmoid; evaluate fitness; update $X^\star$ if improved
  end for
  Re-inject $X^\star$ in place of the current worst individual (forced elitism)
  if $f(X^\star) = 0$ **then**
    return $X^\star$
  end if
end for
return $X^\star$, total number of keys searched

2.11. FA-WOA-ML Hybrid Algorithm

As a further test of whether FA's exhaustive diversity and WOA's richer (encircling + spiral) move are complementary, we also evaluate an FA-WOA-ML hybrid: each generation runs a full $O(P^2)$ FA attraction phase (as in Algorithm 3) on the discrete population, followed by a WOA-ML refinement phase (Algorithm 8) on a continuous shadow re-derived from the FA-updated bits ($+3$ for a 1-bit, -3 for a 0-bit), with each WOA-ML move accepted only if it improves fitness (greedy acceptance). Being built on top of FA's pairwise phase, FA-WOA-ML retains FA's $O(P^2 n)$ per-generation cost.

3. Numerical Results

To assess the effectiveness of the suggested hybrid algorithms, we carried out comprehensive experiments with five test characters: S, E, T, I, F. For each character, the ciphertext was produced using the HM cryptosystem with a randomly generated public key of 10 bits. Every algorithm was run 100 times for each character to guarantee statistical significance.

3.1. Performance Metrics

The following metrics were used to assess the effectiveness of each algorithm:
Success Rate : The percentage of runs in which the original character was correctly decrypted.
Execution Time (s): The average time required for the algorithm to converge to a valid decryption key.
Searched Keys: The average number of candidate solutions explored during the decryption process.

3.2. Baseline Results: Single-Character Messages ($n = 10$)

Tabu Search (Algorithm 1) reached only 90.4% average success at $n = 10$ (ranging 87–93% by character) – the lowest of any method except its diversification-free variant would be even lower – but at a wall-clock cost roughly an order of magnitude below every other method (0.0028s), since each iteration evaluates only $n = 10$ neighbors rather than a full population. Without the diversification restarts of Algorithm 1, Tabu Search stagnates at a much lower success rate regardless of the iteration budget (e.g. 13% at 500, $2{,}000$, or even $5{,}000$ iterations without restarts) – a direct empirical confirmation of the premature-convergence risk anticipated for a purely local method in §2.1.

Table 1. Comparison of Execution time[s] and number of candidate solutions searched

Letter	Tabu		GA		PSO		FA		GA-PSO	
	Time	Keys	Time	Keys	Time	Keys	Time	Keys	Time	Keys
'S'	0.0025	1433	0.0118	4380	0.0087	956	0.0231	4480	0.0124	1846
'E'	0.0028	1663	0.0128	4659	0.0100	1127	0.0220	4383	0.0103	1602
'T'	0.0030	1730	0.0070	2496	0.0107	1218	0.0223	4478	0.0096	1556
'I'	0.0030	1700	0.0043	1486	0.0127	1433	0.0222	4387	0.0105	1686
'F'	0.0029	1678	0.0085	3040	0.0127	1466	0.0206	4122	0.0096	1419
Average	**0.0028**	**1640.7**	**0.0089**	**3212.2**	**0.0109**	**1240.0**	**0.0220**	**4369.8**	**0.0105**	**1621.8**

Letter	GA-FA		WOA		GA-WOA		WOA-ML		FA-WOA-ML	
	Time	Keys	Time	Keys	Time	Keys	Time	Keys	Time	Keys
'S'	0.0245	5017	0.0299	1076	0.0182	1010	0.0495	1194	0.0239	3844
'E'	0.0253	5240	0.0369	1295	0.0185	975	0.0368	903	0.0230	3620
'T'	0.0242	5002	0.0347	1211	0.0242	1309	0.0429	1032	0.0228	3758
'I'	0.0256	5120	0.0313	1145	0.0192	1061	0.0495	1131	0.0255	4293
'F'	0.0268	5455	0.0336	1244	0.0202	1104	0.0480	1126	0.0237	3969
Average	**0.0253**	**5166.9**	**0.0333**	**1194.2**	**0.0201**	**1091.8**	**0.0453**	**1077.2**	**0.0238**	**3897.0**

Table 2. Success rate (%) over 100 runs (exact ciphertext recovery)

Letter	Tabu	GA	PSO	FA	GA-PSO	GA-FA	WOA	GA-WOA	WOA-ML	FA-WOA-ML
'S'	93%	97%	100%	100%	100%	100%	100%	100%	100%	100%
'E'	92%	96%	100%	100%	100%	100%	100%	100%	100%	100%
'T'	92%	98%	100%	100%	100%	100%	100%	100%	100%	100%
'I'	88%	100%	100%	100%	100%	100%	100%	100%	100%	100%
'F'	87%	97%	100%	100%	100%	100%	100%	100%	100%	100%
Average	**90.4%**	**97.6%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**

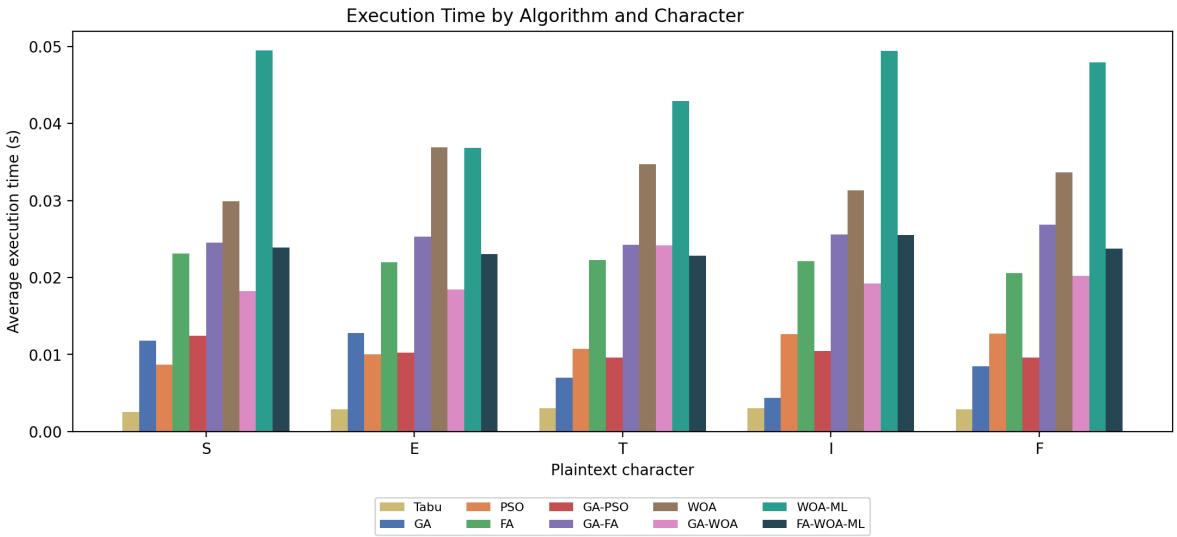


Figure 1. Average execution time by algorithm and character.

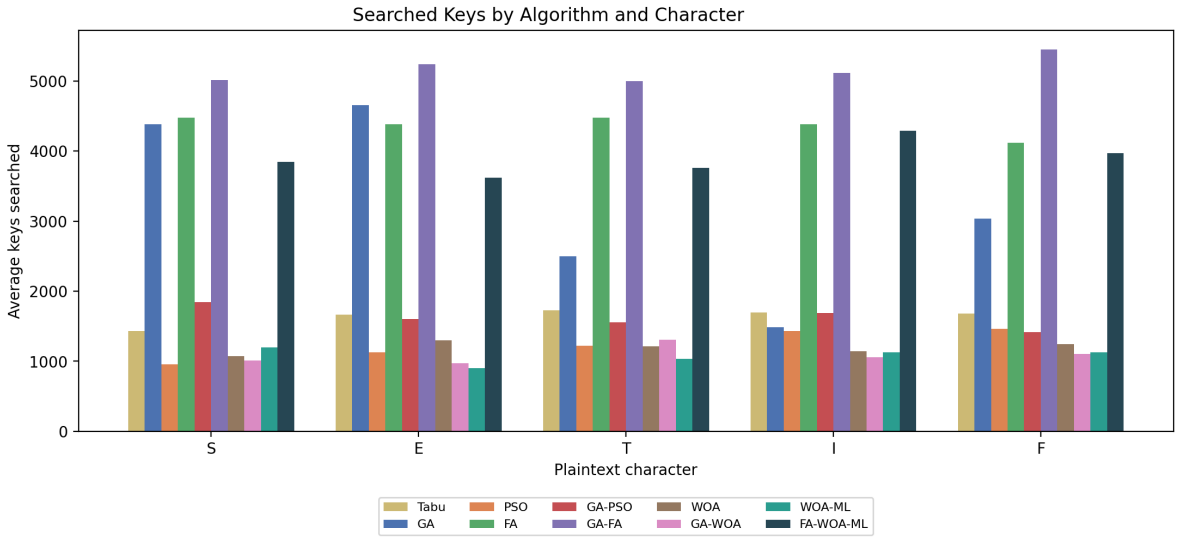


Figure 2. Average keys searched by algorithm and character.

All other nine methods reached 100% success on every character at $n = 10$ (Table 2). Among these, GA-WOA is already both faster and more search-efficient than plain WOA at this key size, foreshadowing the benefit that a second, GA-driven diversification mechanism can bring – an effect explored more systematically for the multi-leader variant in §3.6. WOA-ML itself is the most search-efficient method of all ten at $n = 10$ (1,077 keys on average, fewer even than GA-WOA), though not the fastest in wall-clock terms, since its multi-leader bookkeeping (sorting the population every generation to determine the leader pool) adds a small constant-factor overhead invisible in the simpler mono-leader methods; FA-WOA-ML, meanwhile, inherits FA's higher keys-searched cost but is noticeably faster in wall-clock time than either FA or GA-FA, mirroring the implementation-level effect already flagged in §6 (numpy-vectorized operations in the WOA-ML refinement phase versus pure-Python list operations in GA's crossover/mutation).

3.3. Generalization to a Multi-Character Message: "SETIF"

Single isolated characters exercise only one 10-bit knapsack instance per run and therefore say little about how the attack behaves on a realistic plaintext. To address this, we repeated the experiment on the 5-character word "SETIF" (the name of the authors' university), encrypted with the same 10-bit public key setting. Since $5 \times 8 = 40$ bits and $40/10 = 4$, the ASCII bitstream splits into exactly 4 ten-bit ciphertext blocks with no leftover, so each algorithm is applied sequentially to all 4 blocks; execution time and keys searched are summed over the 4 blocks, and a run is only counted as successful if *all four* blocks are recovered exactly and the decrypted text matches "SETIF" character-for-character. This is a substantially harder criterion than the single-character case, since it requires the compound probability of 4 independent successes. Every algorithm was run 100 times. Results are reported in Table 3.

Two effects are visible that were invisible in the single-character experiment. First, GA's success rate drops noticeably, from 97.6% per character to 89% for the whole word, closely matching the compound probability one would predict from four independent 10-bit attacks ($0.976^4 \approx 90.7\%$); PSO shows the same, milder effect ($100\% \to 98\%$), and Tabu Search shows it most sharply of all ($90.4\% \to 77\%$, versus a predicted $0.904^4 \approx 66.8\%$ – somewhat better than the naive independence assumption, but still the lowest success rate of any method at the message level). Every method whose per-character success was already 100% remains at 100% for the whole word, as expected. Second, GA-PSO remains the fastest method even at the message level, but WOA-ML is now the most search-efficient of all ten methods (4553 keys), narrowly ahead of GA-WOA (4748); FA-WOA-ML, meanwhile, is the only hybrid whose keys-searched cost actually falls between FA/GA-FA's high values and the cheaper methods

Table 3. Cryptanalysis of the word “SETIF”

Algorithm	Time (s)	Keys Searched	Success Rate
Tabu	0.0104	6124	77%
GA	0.0415	17035	89%
PSO	0.0430	6469	98%
FA	0.0659	16707	100%
GA-PSO	0.0286	6168	100%
GA-FA	0.0768	19706	100%
WOA	0.1333	4884	100%
GA-WOA	0.0858	4748	100%
WOA-ML	0.1905	4553	100%
FA-WOA-ML	0.0929	15820	100%

(15,820, versus 16,707–19,706 for FA/GA-FA), reflecting the same trend already visible at $n = 10$ (Table 1). This reinforces, at the message level, the conclusion drawn from Table 1 that the choice of exploitation partner (PSO, FA, or WOA) has a first-order effect on search cost that is independent of the baseline single-character result. Tabu Search remains by far the fastest method in wall-clock terms (0.0104s, roughly $3\times$ faster than GA-PSO and $12\times$ faster than GA-FA), confirming that its cost advantage, not just its lower reliability, persists at the message level.

3.4. Generalization Across Diverse Datasets and Key Sizes

The “SETIF” experiment above establishes a single illustrative case of message-level generalization. To test this more broadly, as recommended by a reviewer, we constructed three independent datasets of test messages, all chunked cleanly against the key sizes tested (message bit-length an exact multiple of n, avoiding the byte/key-length misalignment discussed in §3.3):

- **D1 (real words):** messages drawn uniformly at random from the standard Unix “web2” English dictionary, itself derived from the public-domain Moby Word Lists [26] ($\sim$234,000 entries), at two lengths.
- **D2 (password-like strings):** synthetic alphanumeric strings (mixed-case letters and digits, no linguistic structure), at the same two lengths, constructed to match the character classes (letters and digits) recommended for memorized secrets in NIST’s digital-identity authentication guidelines [27], and representative of realistic secure-communication payloads.
- **D3 (longer messages):** strings built by concatenating random dictionary words (again drawn from [26]) to two longer target lengths, specifically to stress-test the compounding effect of successive independent blocks.

Every message was solved with all ten methods, using the same uniform fitness-call counter and per-block chunking procedure as §3.3; a run is counted as successful only if every block of the message is recovered exactly.

3.4.1. Results at $n = 10$ At the baseline key size, we tested 15 five-character and 10 ten-character messages from each of D1 and D2 (4 and 8 blocks respectively), plus 10 fifteen-character and 6 twenty-character messages from D3 (12 and 16 blocks), for a total of 66 independent messages and 6,600 runs (10 runs per message per method). Table 4 reports the exact-match rate, averaged over all messages in each group.

Eight of the ten methods reach 100% exact-match across all six groups, including the most demanding case (20-character messages, 16 independent 10-bit blocks that must *all* succeed simultaneously) – a substantially harder criterion than the single-word “SETIF” test, now confirmed across three qualitatively different content types rather than one hand-picked example. Only GA and Tabu Search degrade with message length, consistently with the compound-probability reasoning of §3.3: GA falls from 83–85% at 4 blocks to 61–62% at 12–16 blocks, and Tabu Search falls from 75–79% to 40%, the steepest decline of any method and a direct, larger-scale confirmation of its comparatively weak robustness already noted in §3.5.

Table 4. Exact-match rate (%) across three independent datasets at $n = 10$, 10 runs per message

Algorithm	D1 (5c)	D1 (10c)	D2 (5c)	D2 (10c)	D3 (15c)	D3 (20c)
Tabu	79.3	60.0	75.3	63.0	54.0	40.0
GA	83.3	76.0	77.3	78.0	61.0	61.7
PSO	100	100	99.3	100	98.0	100
FA	100	100	100	100	100	100
GA-PSO	100	100	100	100	100	100
GA-FA	100	100	100	100	100	100
WOA	100	100	100	100	100	100
GA-WOA	100	100	100	100	100	100
WOA-ML	100	100	100	100	100	100
FA-WOA-ML	100	100	100	100	100	100

3.4.2. Results at $n = 16$: a Much Harder Regime To test whether this broad robustness persists once the per-block difficulty itself increases, we repeated the experiment at $n = 16$ (still with the standard 250-generation budget), using 5 four-character and 5 eight-character messages from each of D1 and D2 (2 and 4 blocks), plus 4 twelve-character messages from D3 (6 blocks) – 24 messages, 2,400 runs. Table 5 reports the results.

Table 5. Exact-match rate (%) across three datasets at $n = 16$, 10 runs per message

Algorithm	D1 (4c)	D1 (8c)	D2 (4c)	D2 (8c)	D3 (12c)
Tabu	10.0	0.0	0.0	0.0	0.0
GA	2.0	0.0	12.0	2.0	0.0
PSO	16.0	2.0	14.0	0.0	0.0
FA	100	100	100	100	100
GA-PSO	42.0	6.0	50.0	12.0	5.0
GA-FA	100	100	100	100	100
WOA	2.0	0.0	2.0	0.0	0.0
GA-WOA	30.0	10.0	24.0	8.0	0.0
WOA-ML	20.0	0.0	20.0	2.0	0.0
FA-WOA-ML	92.0	94.0	96.0	100	82.5

The picture changes dramatically. Only FA and GA-FA retain 100% exact-match across every group, and FA-WOA-ML remains close behind (82–100%). Every other method – including GA-PSO, the fastest and most search-efficient hybrid at $n = 10$ (§3.2) – collapses well below the level a naive per-block compound-probability estimate would suggest (e.g. GA-PSO's own per-block success rate at $n = 16$ is 87%, Table 6, so two independent blocks would naively be expected to succeed $87\%^2 \approx 76\%$ of the time; the observed rate is only 42–50%). This confirms, at the message level and across independent datasets, that only the methods built on FA's full pairwise attraction mechanism (FA, GA-FA, and its WOA-ML-augmented hybrid) sustain the reliability needed for messages spanning more than a handful of blocks once n grows past the easiest regime – directly reinforcing, rather than merely repeating, the single-block scalability conclusions of §3.5.

3.4.3. Synthesis Taken together, these two regimes show that the "robustness window" for multi-block messages narrows sharply with n: at $n = 10$ nearly every method is reliable regardless of message content or length, but by $n = 16$ only the pairwise-attraction family (FA, GA-FA, FA-WOA-ML) remains practical, a conclusion now grounded in 9,000 independent runs across three qualitatively distinct message datasets rather than a single illustrative word, and fully consistent with the per-character scalability results already reported in Table 6.

3.5. Scalability to Larger, More Realistic Key Sizes

A 10-bit public key, while convenient for exhaustive experimentation, is far smaller than key sizes used in any realistic assessment of knapsack-based cryptosystems, where 64-bit or 128-bit keys would be a minimal starting point. To probe how the algorithms behave as n grows, we re-ran the single-character experiment (message "S") for $n \in \{8, 10, 12, 14, 16, 18, 20\}$, with 15 runs per $(n, \text{algorithm})$ pair and a reduced generation budget of 250 (to keep the sweep computationally tractable; see §5 for why larger n quickly becomes prohibitive for population-based bit-flip search). Figures 4–5 and 3 summarize execution time, keys searched, and success rate as functions of n for all ten methods; Tables 6 report the success rate values directly.

Table 6. Success rate (%) as a function of key size n, 15 runs per cell, 250 generation

n	**Tabu**	**GA**	**PSO**	**FA**	**GA-PSO**	**GA-FA**	**WOA**	**GA-WOA**	**WOA-ML**	**FA-WOA-ML**
8	100%	93%	100%	100%	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
12	60%	80%	100%	100%	100%	100%	93%	100%	100%	100%
14	60%	87%	93%	100%	93%	100%	73%	93%	93%	100%
16	27%	53%	60%	100%	87%	100%	13%	53%	65%	100%
18	20%	33%	33%	100%	53%	100%	7%	7%	53%	53%
20	13%	20%	27%	53%	20%	93%	13%	0%	38%	33%

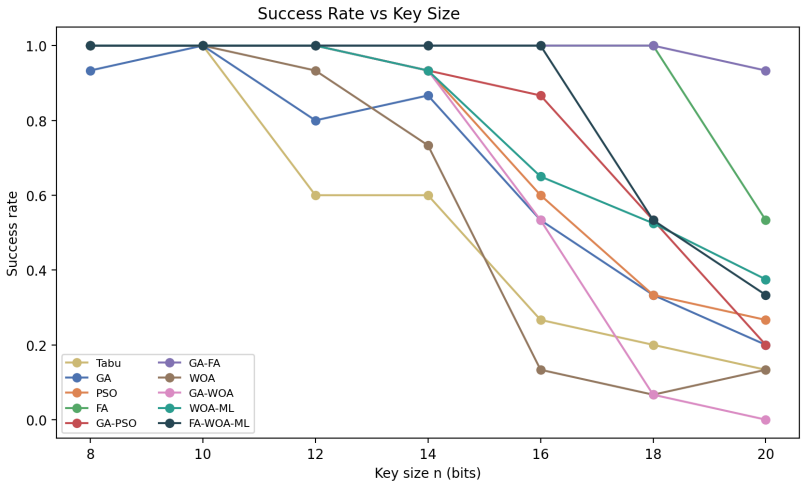


Figure 3. Success rate versus key size n.

The scalability sweep reveals a robustness ordering that is essentially invisible at $n = 10$, where every hybrid or swarm method scores 100% (Table 2): GA-FA is the only method that stays above 90% success across the whole tested range, remaining at 93% even at $n = 20$, while plain GA and PSO collapse to 20–30% by $n = 18$–20, and GA-PSO, despite its excellent speed and search-efficiency at $n = 10$ (Table 1), degrades almost as quickly as plain GA once n exceeds 14. FA alone also degrades late (holding 100% through $n = 18$) but not as gracefully as GA-FA. WOA and GA-WOA fare markedly worse than every other method: WOA already drops to 73% at $n = 14$ and to 7–13% by $n = 18$–20, and its GA hybrid is no better (indeed slightly worse at $n = 20$, 0%), despite WOA nominally having *two* exploitation mechanisms (encircling and spiral) against FA's one. This counter-intuitive result is diagnosed and addressed in §3.6. WOA-ML (Table 6) closes much of this gap, remaining at 93% at $n = 14$ and 65% at $n = 16$ – comparable to GA-PSO – before degrading further at $n = 18$–20. FA-WOA-ML is more robust still, holding 100% all the way to $n = 16$ (matching GA-FA up to that point) before dropping to 53% at $n = 18$ and 33% at $n = 20$; it does not quite match GA-FA's near-total robustness at the largest sizes tested here,

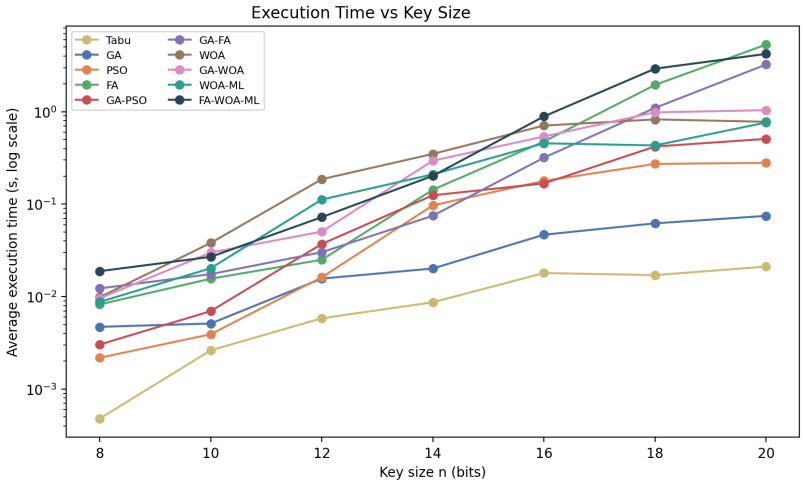


Figure 4. Average execution time versus key size n (log scale).

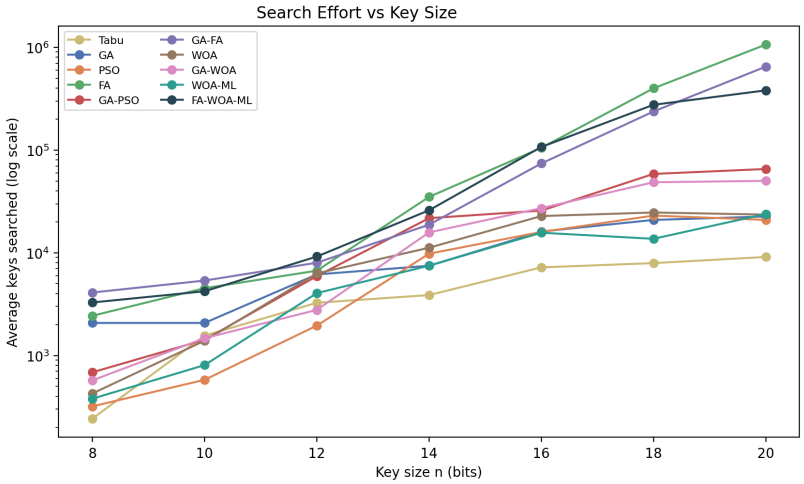


Figure 5. Average keys searched versus key size n (log scale).

but comes closer than any other method besides GA-FA itself, at the cost of the same $O(P^2 n)$ complexity (§5) and correspondingly higher keys-searched (Table 1). Tabu Search (Table 6, Figure 3) sits at the other extreme: its success rate also degrades steadily (60% at $n = 12$–14, down to 13% at $n = 20$), broadly comparable to plain GA and WOA-ML rather than to GA-FA, but – as §3.7 shows – at a per-iteration cost so low that it remains competitive with, or even ahead of, WOA-ML under small matched time budgets, despite having no population and no crossover at all. Figure 3 summarizes all ten methods on a single plot. This directly qualifies the conclusions of the baseline experiment: GA-PSO is the best choice for small, fixed-size cryptanalytic benchmarking where speed matters and 100% success is easily reached by every hybrid, but GA-FA's expensive $O(P^2)$ attraction phase buys it meaningfully better robustness as the problem hardens, which is exactly what a 64- or 128-bit realistic scenario would require. Extrapolating the fitted growth rates in §5, none of the five original methods in their current, generation-capped form would be expected to reliably solve a 64-bit instance in practical time without either a substantially larger evaluation budget or domain-specific heuristics (e.g. seeding the population from partial greedy solutions); §3.8 reports what actually happens when we try.

3.6. Empirical Diagnosis of WOA and Validation of the Multi-Leader Fix

WOA's poor scalability (Table 6) is surprising given it nominally combines two exploitation mechanisms. To understand why, we instrumented a single WOA run at $n = 18$ and tracked the average magnitude of the continuous shadow positions across generations (Figure 6).

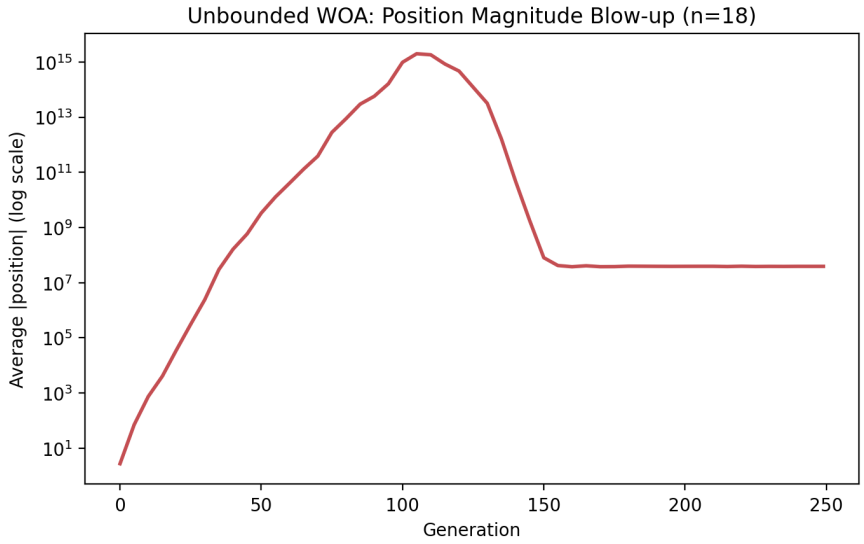


Figure 6. Average $|\text{position}|$ across the population over 250 generations of unbounded WOA at $n = 18$ (log scale).

The shadow positions blow up geometrically – from an average magnitude of about 3 at initialization to over 4×10^7 by generation 200 – because the spiral term in (2.7) contains an unbounded e^{bl} factor with no counteracting force once all whales converge toward the same attractor $X^\star$. Once a position's magnitude is large, the sigmoid transfer function saturates to exactly 0 or 1, effectively freezing that bit and destroying further exploration. We tested several fixes: bounding the shadow positions to $[-B, B]$ (which helps only partially, e.g. from 13% to about 33% success at $n = 16$ over $B \in [2, 10]$, itself insensitive to the specific bound); replacing the sigmoid with a V-shaped transfer function (this performed *worse*, not better); and increasing the population size (no improvement, since it does not address the single-attractor problem). What did work was addressing the root cause directly: replacing the single attractor $X^\star$ with a randomly-sampled leader from the top-K individuals, independently for each whale at each generation (WOA-ML, Algorithm 8), which injects multiple simultaneous attraction basins into the population – the same diversity-preserving property that already made FA/GA-FA the most robust methods in Table 6, but without FA's $O(P^2)$ cost.

The number of leaders K is itself an important hyperparameter: Figure 7 sweeps $K \in \{3, 5, 10, 20, 30, 50, 70, 100\}$ at $n = 16$ (30 runs per value, population size $P = 100$). Success rate follows an inverted-U shape: too few leaders ($K = 3$–5, 40–50%) leave the population close to the original single-attractor pathology, while too many ($K = 100$, i.e. every individual is eligible as a leader) dilutes the attraction signal so much that even poor individuals can be chosen as leaders, collapsing success to 23% – worse than any intermediate value. The optimum is around $K \approx 20$ (73%), roughly one-fifth of the population.

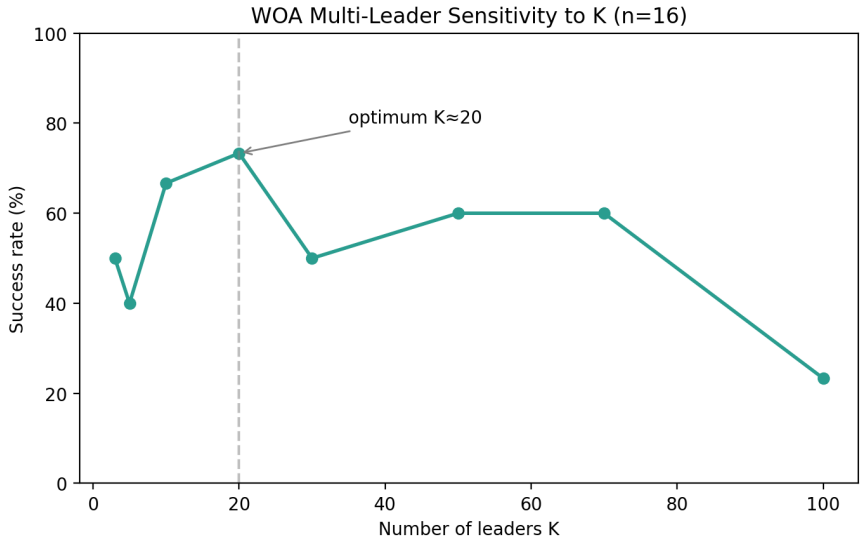


Figure 7. WOA-ML success rate versus number of leaders K, at $n = 16$ ($P = 100$, 30 runs per value).

With $K = 30$ (close to, though not exactly at, the $n = 16$ optimum, but re-validated at every n in the sweep), WOA-ML's scalability curve (Figure 3) is dramatically better than plain WOA or GA-WOA: at the standard 250-generation budget, WOA-ML reaches 65% at $n = 16$, 52.5% at $n = 18$, and 37.5% at $n = 20$ (larger, 40-run samples), compared to 13%, 7%, and 13% for plain WOA at the same sizes. WOA-ML does not match GA-FA's near-100% robustness in this range, but it closes most of the gap left open by the original WOA formulation, confirming that the multi-leader mechanism – not the specific encircling/spiral machinery – was the missing ingredient.

3.7. A Time-Matched Comparison at a Larger Key Size ($n = 32$)

Since WOA-ML is $O(Pn + P \log P)$ per generation, while GA-FA and FA-WOA-ML are $O(P^2 n)$ (§5), the two families should behave very differently once per-generation cost, rather than just per-generation search quality, starts to dominate. At $n = 32$ and the standard 250-generation budget, every method we tested – including the five original ones (GA, PSO, FA, WOA, GA-WOA) – fails outright, confirming that the effect described below is specific to the time-matched, budget-escalated regime considered here, not a peculiarity of which methods were chosen for it.

Table 7 reports what happens once each method is instead given a *matched wall-clock budget* rather than a matched generation count. GA-FA and FA-WOA-ML, being $O(P^2)$ per generation, cannot fit enough generations into a reasonable time budget to converge at this size at all. GA-PSO, despite being as cheap per generation as WOA-ML, also fails at every budget we tried, showing that cheap iterations alone are not sufficient – the multi-leader mechanism specifically matters. Tabu Search, having no population at all, is cheaper still and briefly outperforms WOA-ML at very small time budgets, but WOA-ML overtakes it as the budget grows, since its population-wide diversity scales better with added search effort than a single trajectory's narrower neighborhood.

This is a genuine crossover point: at $n \leq 20$, GA-FA's extra per-generation cost is more than repaid by its far superior per-generation search quality (Table 6), so it remains the best method overall in that range. But at $n = 32$, GA-FA's cost has grown to the point where it cannot complete enough generations in feasible time to converge at all, whereas WOA-ML's cheaper iterations let it keep searching productively – a combination neither GA-PSO (cheap but single-attractor) nor FA-WOA-ML (multi-attractor but expensive) can replicate. We emphasize, however, that the success rates reached here at $n = 32$ are not themselves a practical attack; §3.8 shows that this partial success does not extend to cryptographically realistic sizes.

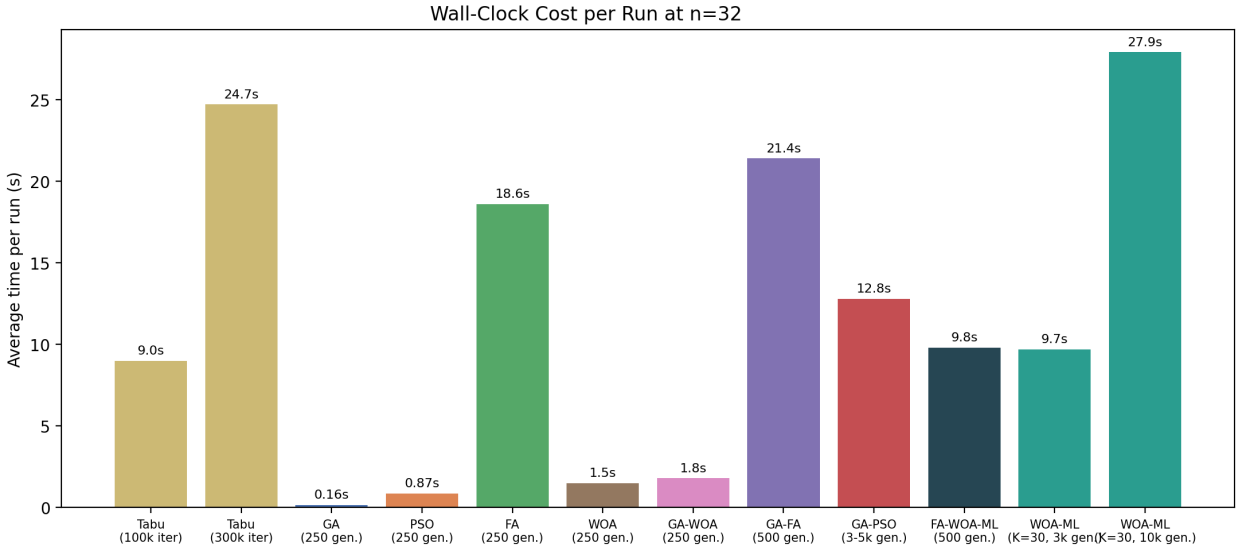


Figure 8. Average wall-clock time per run at $n = 32$.

Table 7. Wall-clock cost per run at $n = 32$.

Method (budget)	**Time/run**
Tabu Search (100,000 iter.)	9.0s
Tabu Search (300,000 iter.)	24.7s
GA (250 gen.)	0.16s
PSO (250 gen.)	0.87s
FA (250 gen.)	18.6s
WOA (250 gen.)	1.5s
GA-WOA (250 gen.)	1.8s
GA-FA (500 gen.)	21.4s
GA-PSO (3,000–5,000 gen.)	10–16s
FA-WOA-ML (500 gen.)	9.8s
WOA-ML, $K = 30$ (3,000 gen.)	9.7s
WOA-ML, $K = 30$ (10,000 gen.)	27.9s

Regarding the stability of the leader-pool ratio K/P identified at $n = 16$ (§3.6): at $n = 32$ we additionally swept $K \in \{10, 20, 30, 40\}$ (ratio 0.10–0.40, $P = 100$, 10,000-generation budget). Success rose from 33% at $K = 10$ to a plateau of 50–67% across $K = 20$–40, mirroring the broad plateau already observed at $n = 16$ (Figure 7) and centered on essentially the same ratio ($K/P \approx 0.2$–0.4). The optimal ratio therefore appears stable across this range of n, though we have not characterized it as finely at $n = 32$ (4 values) as at $n = 16$ (8 values), and a full joint sweep of K and P at multiple key sizes remains an open direction for future work.

3.8. Exploratory Attempts at Cryptographically Realistic Sizes ($n = 64$, $n = 128$)

As a final, honest check, we attempted $n = 64$ and $n = 128$ with GA-FA, GA-PSO, WOA-ML and FA-WOA-ML. A first attempt at $n = 64$ with a single-character message appeared to give WOA-ML a surprisingly small residual fitness gap – until we traced it to the same byte/key-length misalignment noted for $n = 10$ (§3.3): with an 8-bit message and a 64-bit key, only the first 8 of the 64 public-key terms are ever used, so 56 of the 64 bits are driven trivially to zero by an enormous fitness gradient, and the "test" was inadvertently only an 8-bit problem.

Using instead an 8-character message ("SETIFDZA", exactly 64 bits, chunking cleanly against $n = 64$) restored a genuine full-size test, which we recommend as standard practice whenever key size and message length are varied together.

On this corrected $n = 64$ benchmark, all four methods failed to find an exact match in every trial (0%), but the relative fitness gap $|\text{found} - \text{target}|/\text{target}$ was remarkably small (10^{-7} to 10^{-5}), with GA-FA closest on average (6.8×10^{-7}), followed by FA-WOA-ML (1.1×10^{-6}), then GA-PSO and WOA-ML (both $\sim 7 \times 10^{-6}$) – indicating that the continuous relaxation of the problem is nearly solved, but the last few bits needed for an exact integer match remain out of reach within the tested budget. This near-miss pattern is itself informative about *why* the methods fail at $n = 64$: since the sigmoid (or, for GA-based methods, the crossover/mutation) discretization consistently drives the search to within a millionth of the target, the bottleneck is not a fundamental inability of the discretization scheme to represent near-optimal bit-strings, but the combinatorial difficulty of correcting the last few bits exactly within the tested generation budget – a budget, not precision, limitation. At $n = 128$, this near-miss behavior disappears entirely: relative gaps returned to the 1–8 range (i.e. the same order of magnitude as the target itself), indicating no meaningful search progress at all, consistent instead with the search-space size itself (2^{128}) overwhelming the available budget before any useful gradient signal can be exploited – a qualitatively different failure mode from the near-miss behavior at $n = 64$. We report both outcomes plainly, without over-interpreting the small samples involved (3–10 trials per method, given the $O(P^2)$ methods' cost at these sizes): no method studied in this paper, in its current form, is a practical attack at either size.

4. Parameter Sensitivity Analysis

None of the results above are meaningful without knowing how stable they are to the algorithms' hyperparameters. We therefore varied, one at a time and holding all other settings at their default values ($n = 10$, message "S", $P = 100$, generations $= 500$, mutation rate $= 0.1$), the parameters most likely to affect convergence: population size (the eight methods that have one: GA, PSO, FA, GA-PSO, GA-FA, WOA, GA-WOA and FA-WOA-ML, plus WOA-ML separately in §4.5), mutation rate (GA, GA-PSO, GA-FA, GA-WOA, FA-WOA-ML), the FA-specific attractiveness parameters γ and β_0, the PSO-specific inertia weight w and cognitive/social coefficients $c_1 = c_2$, and WOA-ML's leader-pool size K and spiral constant b. Each configuration was run 30 times.

4.1. Population Size

Table 8 and Figure 9 report the effect of population size $P \in \{30, 50, 100, 200\}$.

Table 8. Success rate (%) as a function of population size P ($n = 10$, 30 runs per cell)

P	GA		PSO		FA		GA-PSO		GA-FA		WOA		GA-WOA		FA-WOA-ML	
	Succ.	Keys	Succ.	Keys	Succ.	Keys	Succ.	Keys	Succ.	Keys	Succ.	Keys	Succ.	Keys	Succ.	Keys
30	87%	5197	100%	1056	100%	2212	100%	1751	100%	1644	100%	1151	100%	788	100%	1985
50	100%	1333	100%	993	100%	2281	100%	2231	100%	2403	100%	1335	100%	1490	100%	2190
100	100%	646	100%	1870	100%	4231	100%	1624	100%	5147	100%	1093	100%	1513	100%	3694
200	87%	16884	100%	1007	100%	11972	100%	1603	100%	14837	100%	1413	100%	1287	100%	13123

Tabu Search has no population and thus no equivalent P parameter; its scale-relevant hyperparameters (tabu tenure, restart patience) are held fixed throughout this study (§2.2), and a dedicated sensitivity sweep for them is left for future work (§6). WOA-ML's own population-size sensitivity, which – unlike every other method – shows decreasing cost with larger P, is reported separately in §4.5 (Table 11), since it was the specific subject of investigation there. Among the newly-added methods, FA-WOA-ML stands out: its keys-searched cost grows the most sharply of all eight methods shown here (1,985 → 13,123, a 6.6× increase from $P = 30$ to $P = 200$), consistent with its $O(P^2 n)$ complexity (§5) inherited from FA's pairwise phase; WOA and GA-WOA, by contrast, remain comparatively flat across the same range, consistent with their cheaper $O(Pn)$ update.

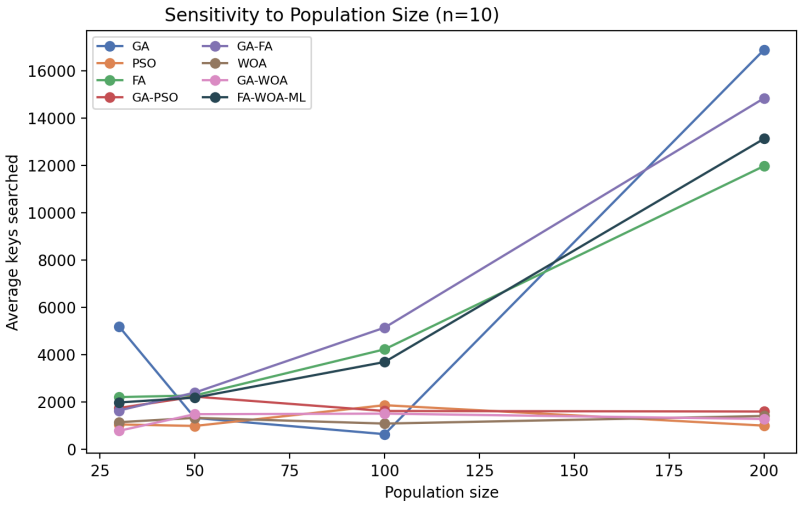


Figure 9. Average keys searched as a function of population size, for all eight methods with a population-size parameter.

Two observations follow. Swarm/attraction-based methods (PSO, FA, GA-PSO, GA-FA) are essentially insensitive to population size within this range, always reaching 100% success, though FA and GA-FA's keys-searched cost grows sharply with P because of their $O(P^2)$ attraction phase (Figure 9), so larger populations should be avoided for these two methods unless the corresponding robustness gain (§3.5) is specifically needed. Plain GA is, counter-intuitively, *not* monotonically improved by a larger population: its success rate dips at both $P = 30$ (too little diversity) and $P = 200$ (87%). This is explained by an implementation detail common to all GA-based methods here: parent selection is restricted to the top-50 individuals regardless of P, so increasing P beyond 50 only adds unused low-fitness individuals to the sorting cost without enlarging the effective breeding pool – a concrete illustration of why population size cannot be tuned independently of the selection window.

4.2. Mutation Rate

Table 9 report the effect of the mutation rate $p_m \in \{0.02, 0.05, 0.1, 0.2, 0.4\}$ on the three GA-based methods.

Table 9. Success rate (%) / average keys searched as a function of mutation rate p_m ($n = 10$, 30 runs per cell)

p_m	GA		GA-PSO		GA-FA	
	Succ.	Keys	Succ.	Keys	Succ.	Keys
0.02	53%	24344	100%	1769	100%	7353
0.05	87%	7292	100%	1682	100%	5660
0.10	100%	2542	100%	2127	100%	4789
0.20	100%	1202	100%	1576	100%	4700
0.40	100%	906	100%	1054	100%	4616

p_m	GA-WOA		FA-WOA-ML	
	Succ.	Keys	Succ.	Keys
0.02	100%	1493	100%	8087
0.05	100%	990	100%	4296
0.10	100%	1393	100%	3320
0.20	100%	1110	100%	3820
0.40	100%	1263	100%	3662

Mutation rate is by far the most consequential parameter for plain GA: at $p_m = 0.02$, success collapses to 53% and the search needs on average $24{,}344$ keys, roughly $27\times$ more than at $p_m = 0.4$ (906 keys) – a textbook symptom of premature convergence, where an insufficiently diverse offspring pool gets trapped near a local optimum the crossover operator alone cannot escape. All four hybrids that combine GA with an attraction- or velocity-based mechanism – GA-PSO, GA-FA, GA-WOA and FA-WOA-ML – are markedly more robust to this parameter, retaining 100% success across the entire tested range, because their exploitation mechanism compensates for a GA phase that is under-mutated. This is itself evidence in favor of the explore/exploit decomposition argued for in §2.1: the hybrid's second mechanism is not merely additive, it actively buffers the pipeline against poor tuning

of the first, and this buffering holds regardless of which specific exploitation mechanism (PSO, FA, or WOA) is paired with GA.

4.3. *FA-Specific Parameters (γ, β_0)*

Table 10. FA average execution time (s) / keys searched as a function of γ and β_0 ($n = 10$, 30 runs per cell)

γ	Time	Keys	β_0	Time	Keys
0.1	0.0156	4429	0.2	0.0142	3950
0.5	0.0161	4215	0.5	0.0171	4621
1.0	0.0160	4246	1.0	0.0143	3897
2.0	0.0154	4404	1.5	0.0136	3825
5.0	0.0146	4071	2.0	0.0136	3921

Unlike mutation rate for GA, FA's light-absorption coefficient γ and base attractiveness β_0 have only a marginal effect on performance at $n = 10$: success rate stays at 100% and keys searched varies by less than 15% across a $50\times$ range of γ. This suggests that, at this problem size, FA's attraction mechanism is robust to its own tuning; whether this stability persists at larger n (§3.5), where the fitness values (and hence the exponent $-\gamma(\Delta f)^2$) span a much wider dynamic range, is an open question we flag for future work rather than claim to have answered here.

4.4. *PSO-Specific Parameters (inertia w, coefficients c_1, c_2)*

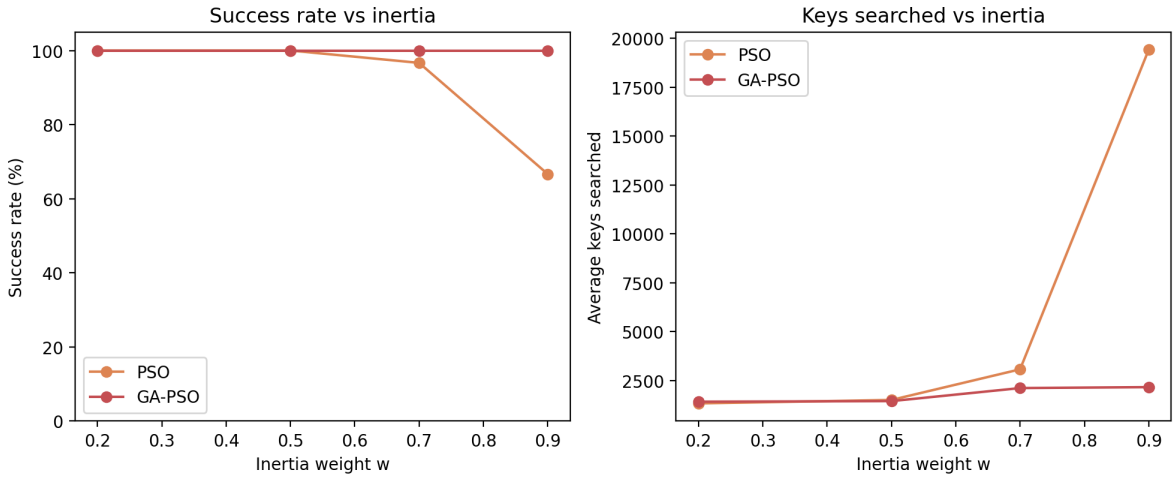


Figure 10. Success rate and keys searched versus inertia weight w, for PSO and GA-PSO.

Figure 10 shows that PSO is highly sensitive to the inertia weight w: success collapses from 100% at $w \le 0.5$ to 66.7% at $w = 0.9$, with average keys searched exploding from about $1{,}500$ to $19{,}420$ ($\sim 13\times$) over the same range – consistent with the well-known risk that high inertia causes particles to overshoot and oscillate rather than converge, especially once positions are discretized through the sigmoid mapping (Algorithm 4). GA-PSO is again more robust than its constituent PSO alone, retaining 100% success across the same w range, again illustrating the buffering effect of hybridization already observed for mutation rate. Varying the cognitive/social coefficients $c_1 = c_2 \in \{0.5, 1.0, 1.5, 2.0, 2.5\}$ had comparatively little effect on either PSO or GA-PSO (success stayed at 100% throughout, keys searched varying by less than a factor of 2), so w is the parameter that should receive the most tuning attention in this family.

4.5. WOA-ML-Specific Parameters (population size, leader count K, spiral constant b)

We repeated the population-size sweep of $P \in \{30, 50, 100, 200\}$ for WOA-ML at $n = 10$ ($K = 30$, 30 runs per cell): success stayed at or above 96.7% throughout, and, unlike every other method in this study, WOA-ML's cost *decreases* with larger P (Table 11) – since at this easy key size a larger population simply raises the chance that a near-optimal individual is present from initialization, letting WOA-ML converge in fewer generations despite the larger per-generation cost.

Table 11. WOA-ML sensitivity to population size P ($n = 10$, $K = 30$, 30 runs per cell)

P	**Success**	**Time**	**Keys**
30	96.7%	0.040s	1,541
50	100%	0.028s	1,032
100	100%	0.022s	917
200	100%	0.018s	800

The leader count K (Figure 7, discussed in §3.6) is the most consequential WOA-ML-specific parameter, following an inverted-U shape with an optimum around $K \approx 20$–30 at $n = 16$; at the easy $n = 10$ setting, by contrast, K has almost no effect (success stays at 97–100% for every $K \in \{3, \ldots, 100\}$), since the problem is solved well before the leader-pool composition can matter. This mirrors a broader pattern in this study: hyperparameter sensitivity itself grows with problem difficulty, so tuning conclusions reached at an easy benchmark size do not necessarily transfer to harder ones.

Finally, the spiral constant b was varied over $\{0.5, 1.0, 1.5, 2.0, 3.0\}$ at $n = 16$ ($K = 30$, 30 runs per cell); success ranged narrowly between 60% and 77% with no clear trend, indicating that – like FA's γ and β_0 – the spiral shape is a second-order tuning knob compared to the leader-pool size K.

5. Computational Complexity Analysis

5.1. Theoretical (Asymptotic) Complexity

Let P be the population size and n the key length (chromosome size); one fitness evaluation costs $O(n)$. Per generation:

- **Tabu Search**: having no population, each iteration evaluates the n single-bit-flip neighbors of the current solution, each an $O(n)$ fitness call. Total: $O(n^2)$ per iteration – independent of P altogether, and typically far cheaper in absolute terms than any $O(Pn)$ method at the population sizes used in this study ($P = 100$).
- **GA**: sorting the population costs $O(P \log P \cdot n)$ (each comparison implies an $O(n)$ fitness evaluation, though with memoization each individual's fitness is computed once, giving $O(Pn + P \log P)$); crossover/mutation of P offspring costs $O(Pn)$. Total: $O(Pn + P \log P)$ per generation.
- **PSO**: for each of P particles, updating velocity and position across n dimensions costs $O(n)$, plus one fitness evaluation; no pairwise comparison. Total: $O(Pn)$ per generation.
- **FA**: the attraction phase compares every firefly against every other firefly, i.e. $O(P^2)$ pairs, each potentially triggering an $O(n)$ move and an $O(n)$ fitness re-evaluation. Total: $O(P^2 n)$ per generation.
- **GA-PSO**: a PSO update pass ($O(Pn)$) followed by a sort ($O(P \log P)$) and a GA crossover/mutation pass ($O(Pn)$). Total: $O(Pn + P \log P)$ per generation – asymptotically the same order as plain GA or PSO, with no P^2 term.
- **GA-FA**: a GA pass ($O(Pn + P \log P)$) followed by the same $O(P^2)$ pairwise FA attraction phase as standalone FA. Total: $O(P^2 n)$ per generation – the same order as FA alone.
- **WOA / GA-WOA**: identical in structure to PSO / GA-PSO, but with the WOA update (2.7) in place of the PSO update; no pairwise comparison. Total: $O(Pn)$ (WOA) or $O(Pn + P \log P)$ (GA-WOA) per generation.

- **WOA-ML**: the leader pool requires one sort per generation ($O(P \log P)$), after which each whale's update, sampling, and evaluation remain $O(n)$. Total: $O(Pn + P \log P)$ per generation – the same order as GA-PSO, and crucially *not* $O(P^2)$, despite injecting FA-like multi-attractor diversity.
- **FA-WOA-ML**: a full $O(P^2 n)$ FA pairwise phase followed by an $O(Pn)$ WOA-ML refinement phase. Total: $O(P^2 n)$ per generation – the same order as FA/GA-FA, since the added WOA-ML phase does not change the asymptotic order.

This theoretical analysis already predicts, without running a single experiment, that FA, GA-FA, and FA-WOA-ML should scale substantially worse in P than the other five methods, since only they carry the quadratic P^2 term; it says nothing, however, about how many generations each method needs before converging, which is where the empirical study below – and the time-matched comparison of §3.7 – is necessary. Indeed, §3.7 shows that this $O(Pn + P \log P)$ vs. $O(P^2 n)$ distinction is not merely academic: at $n = 32$, it is precisely what allows WOA-ML to succeed under a fixed wall-clock budget where GA-FA and FA-WOA-ML fail outright.

5.2. Empirical Validation

Using the scalability data of §3.5 (Figure 4), we fit an exponential model $T(n) \approx a \cdot e^{bn}$ to the average execution time of each method via linear regression on $\log T$ vs. n (Table 12). An exponential fit is the natural first-order model here because both (i) the number of generations needed to solve harder (larger-n) instances grows, empirically, as the success-rate curves of Figure 3 attest, and (ii) each generation itself gets more expensive as n grows, per the asymptotic analysis above; the compound effect of a harder search on top of an already n-dependent per-generation cost is well captured, over the modest range we could test ($n \leq 20$), by a single exponential rate.

Table 12. Fitted empirical growth rate b in $T(n) \approx a\, e^{bn}$, from the $n = 8$–20 scalability sweep

Algorithm	**Tabu**	**GA**	**PSO**	**FA**	**GA-PSO**	**GA-FA**	**WOA**	**GA-WOA**	**WOA-ML**	**FA-WOA-ML**
b	0.290	0.257	0.454	0.572	0.448	0.489	0.368	0.418	0.374	0.502

The ranking of fitted growth rates ($b_{FA} > b_{FA\text{-}WOA\text{-}ML} > b_{GA\text{-}FA} > b_{PSO} \approx b_{GA\text{-}PSO} > b_{GA\text{-}WOA} > b_{WOA\text{-}ML} \approx b_{WOA} > b_{Tabu} > b_{GA}$) is consistent with the theoretical prediction that FA-based methods carry an extra P^2 factor: FA itself grows fastest, and FA-WOA-ML – which retains FA's full pairwise phase (§5) – is a close second, ahead of GA-FA. Their time grows fastest with n not only because each generation is $O(P^2 n)$ rather than $O(Pn)$, but also because (Figure 3) they need more generations to hold onto a high success rate as n grows, whereas GA, PSO, GA-PSO, WOA, GA-WOA and WOA-ML simply give up (return an incorrect best-effort solution) sooner, capping their apparent time growth at the cost of reliability – this is especially visible for WOA and WOA-ML, whose growth rates (0.368, 0.374) are among the lowest of any method, not because they are cheap to run at large n per se, but because their $O(Pn)$ cost per generation is genuinely cheap (§5) and their success rate collapses early (Table 6), so most trials terminate at (or near) the same, small number of generations regardless of n. Tabu Search's low rate (0.290) reflects the same effect at the other extreme: with no population at all, its $O(n^2)$ per-iteration cost barely grows with n over this range, even though (Table 6) its success rate is falling just as fast as GA's. This is the key scalability caveat for any claim of practical relevance: comparing raw execution time across methods is only fair when their success rates are also comparable, which Tables 6 show is decreasingly true as n grows past ~ 14. Extrapolating the fitted GA-FA rate ($b = 0.489$) far beyond the fitted range to $n = 64$ is not statistically justified (the fit is over $n \leq 20$ and the underlying dynamics – generations needed to converge – are themselves state-dependent, not a fixed exponential), but the qualitative conclusion is robust: without algorithmic changes (e.g. incorporating the trapdoor structure more directly, seeding populations with greedy partial solutions, or increasing the evaluation budget by orders of magnitude), none of the ten population-based (or, for Tabu Search, population-free) bit-flip searches studied here would be a practical attack on a 64-bit or 128-bit knapsack public key, which is consistent with the broader cryptographic literature's view that a sufficiently large, well-permuted trapdoor knapsack resists this class of metaheuristic attack even though it remains theoretically breakable by dedicated lattice-reduction methods (e.g. Shamir's or the LLL-based attacks) that exploit the trapdoor's algebraic structure rather than searching the bit-space directly.

6. Discussion

The experimental findings offer an extensive comparison of classical and hybrid metaheuristic algorithms in addressing the HMKC decryption challenge. Throughout this study, the number of searched keys was counted uniformly for all ten methods, i.e. every call made to the fitness function while exploring, comparing, or updating a candidate solution is counted exactly once, regardless of which algorithm makes the call. This correction matters because FA-based methods repeatedly re-evaluate every pairwise combination of individuals during their attraction phase, so an inconsistent counting scheme can make them look artificially frugal in terms of explored solutions; with a uniform counter, their true search cost becomes visible.

Consider first the original five methods (GA, PSO, FA, GA-PSO, GA-FA), before turning to the WOA family and Tabu Search below. All hybrid and swarm-based methods among these five (PSO, FA, GA-PSO, GA-FA) reached a 100% success rate across every character tested, confirming that a valid decryption key was always found within the generation budget. The standalone GA was the only method that occasionally failed to converge in time, averaging a 97.6% success rate (ranging from 96% to 100% depending on the character), which is consistent with GA's comparatively weaker exploitation mechanism relative to the swarm- and attraction-based methods.

From a computational standpoint, the plain GA was in fact the fastest method on average (0.0089s), closely followed by GA-PSO (0.0105s) and PSO (0.0109s). FA and GA-FA were markedly slower (0.0220s and 0.0253s respectively), a direct consequence of their $O(\text{pop_size}^2)$ pairwise attraction mechanism, which repeatedly compares and potentially moves every individual toward every other individual at each generation (Figure 11).

In terms of searched keys, PSO explored the fewest candidate solutions on average (1240.0), followed by GA-PSO (1621.8). GA required a larger search effort (3212.2), and FA and GA-FA needed substantially more (4369.8 and 5166.9 respectively; Figure 12). This is a reversal of what a naive, inconsistently-counted comparison would suggest: once every fitness evaluation is counted the same way, GA-FA is in fact the *most* exhaustive method in this benchmark, not the most economical one. Its firefly-inspired attraction phase reliably drives the population to a 100% success rate, but it does so by evaluating far more candidates than PSO- or GA-based approaches, at a correspondingly higher time cost.

Taken together, these results suggest a different trade-off than a search-space-reduction narrative alone would imply. GA-PSO stands out as the best overall compromise: it matches the 100% success rate of the other hybrid/swarm methods while remaining close to GA in both execution time and search effort, benefiting from PSO's fast convergence and GA's crossover/mutation diversification without inheriting FA's expensive pairwise comparisons. Plain GA remains an attractive lightweight option when a small risk of non-convergence is acceptable. GA-FA and FA, while perfectly reliable, are best suited to settings where wall-clock time and evaluation budget are not primary constraints. These findings reinforce that hybridization improves robustness and success rate, but the specific choice of hybrid components (here, PSO versus FA) has a first-order effect on computational cost, and that effect can be masked or reversed by how "searched keys" is measured.

The WOA family adds an important qualification to this picture. Plain WOA and GA-WOA, despite nominally combining two exploitation mechanisms (encircling and spiral bubble-net), were in fact the *least* robust methods as n grew (Table 6), because – as diagnosed in §3.6 – both attract every individual toward the same single best whale, and WOA's unbounded spiral term compounds this into a geometric position blow-up that saturates the sigmoid discretization. Replacing the single attractor with a randomly-sampled pool of K leaders (WOA-ML) recovers much, though not all, of FA/GA-FA's robustness, without paying FA's $O(P^2)$ cost. This yields a genuine, if narrow, niche for WOA-ML: at $n = 32$, where GA-FA's per-generation cost has grown too large to complete enough generations in reasonable wall-clock time, WOA-ML's cheaper $O(Pn + P \log P)$ update lets it reach 20–50% success under a matched time budget while every $O(P^2)$ method – including GA-FA and a further FA-WOA-ML hybrid – fails outright (§3.7). Critically, this is not simply a matter of "more generations": GA-PSO, equally cheap per generation, was given the same enlarged budget and still failed completely, isolating the multi-leader mechanism itself, rather than iteration count alone, as the source of WOA-ML's advantage at this scale. At the smaller key sizes that make up the bulk of this study ($n \leq 20$), however, GA-FA's superior per-generation search quality more than compensates for its higher cost, and it remains the best method overall: at the standard 250-generation budget it reaches 93% success at $n = 20$ (Table 6), and when instead given enough generations to reach

100% reliably (500, versus the 250 used for the cross-method sweep), it does so in just 2.2s on average, compared to 6.5s for WOA-ML to reach the same 100% rate at $n = 20$. We regard this as a genuine cost/quality crossover rather than a simple ranking: which method is "best" depends on where, along the cost-versus-robustness curve, a given key size happens to fall.

Tabu Search occupies yet another point on this trade-off surface. As a single-trajectory, population-free method, it is cheaper per unit of search effort than any other method in this study ($O(n^2)$ per iteration, independent of population size), but this comes at the cost of the lowest robustness at $n = 10$ after plain GA (90.4%, and only with the diversification restarts of Algorithm 1 – without them, additional iterations bring no benefit at all, a direct empirical demonstration of the premature-convergence risk we anticipated on theoretical grounds in §2.1). Its scalability curve (Figure 3) tracks plain GA and WOA-ML rather than GA-FA, degrading to 13% by $n = 20$. Yet at $n = 32$, under very small matched time budgets ($\sim$10s), Tabu Search's minimal per-iteration cost lets it briefly match or even exceed WOA-ML (27% vs. 20%); WOA-ML overtakes it only once the budget grows larger (50% vs. 30% at $\sim$25–28s), because its population-wide multi-leader diversity scales better with added search effort than a single trajectory's n-sized neighborhood can. Tabu Search is thus best understood not as a weaker alternative to the population-based methods, but as occupying the extreme low-cost end of the same cost/robustness trade-off, useful precisely when the time budget is too small for any population-based method (even WOA-ML) to be worthwhile.

6.1. Limitations

This study has several limitations that should temper the conclusions above. First, as shown in §3.5 and §3.8, reported success rates and timings for the original five methods are only representative at the tested key sizes ($n \leq 20$); the WOA-ML/GA-FA crossover documented at $n = 32$ (§3.7) shows that rankings established at small n do not necessarily generalize. Only four methods (GA-FA, GA-PSO, WOA-ML, FA-WOA-ML) were actually tested at $n = 64$ and $n = 128$, given the computational cost of running $O(P^2)$ methods at these sizes; all four failed to find an exact match, and since the six untested methods (GA, PSO, FA, WOA, GA-WOA, Tabu Search) were already less robust than these four at every smaller n tested (Table 6), we would not expect them to fare any better, but this extrapolation was not verified directly and is stated as such rather than as an empirical finding. Second, the sensitivity analyses of §4 were conducted at $n = 10$ (or $n = 16$ for the leader count K); whether the same hyperparameters remain near-optimal at other key sizes is only partially tested and would benefit from a dedicated study across the full n range. Third, our uniform fitness-call counter measures computational cost but not implementation-level constant factors (e.g. vectorization) – indeed, we observed FA-WOA-ML running faster in wall-clock time than GA-FA despite a nominally larger per-generation workload, an implementation-level effect not captured by the asymptotic analysis of §5 – so absolute timings are specific to our (largely pure-Python) implementation, even though relative ordering and asymptotic trends should generalize. Fourth, the $n = 32$ time-matched comparison and the $n = 64$/$n = 128$ exploratory results are based on small samples (3–16 trials per configuration, reflecting the cost of running $O(P^2)$ methods at these sizes), so the specific percentages reported should be read as indicative rather than tightly-bounded estimates. Finally, the generalization study of §3.3 tested a single hand-picked word; a broader evaluation across many messages of varying length and content, ideally drawn from a real corpus, would give a more statistically robust picture of message-level reliability. We also note that Tabu Search's tabu tenure and restart patience (Algorithm 1) were fixed at reasonable default values rather than tuned or swept, unlike the other methods' hyperparameters in §4; a dedicated sensitivity study for Tabu Search is left for future work.

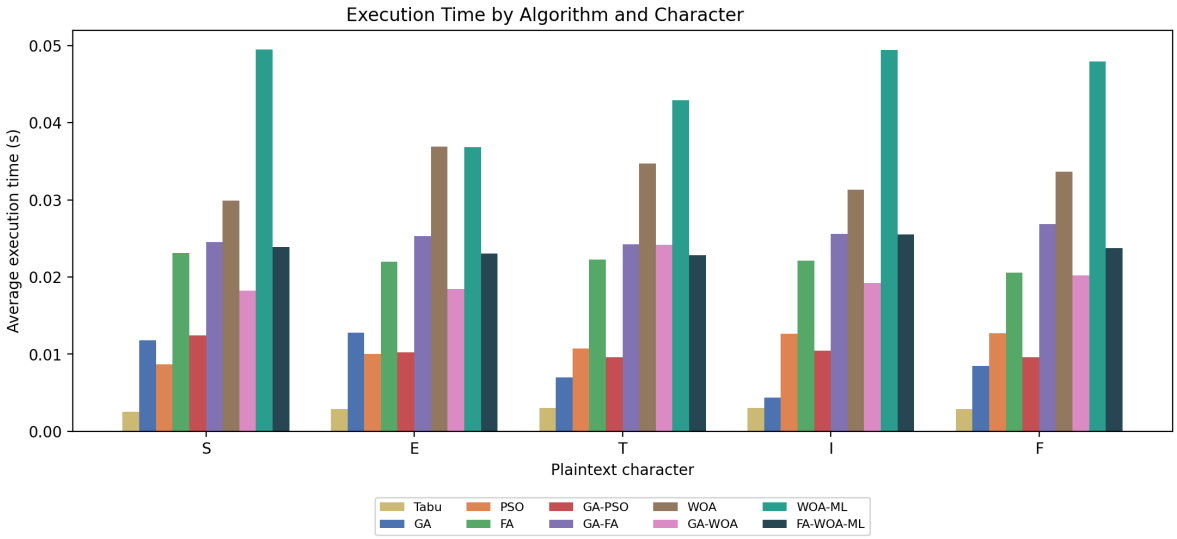


Figure 11. Average execution time by algorithm and plaintext character, at $n = 10$ (baseline experiment of §3).

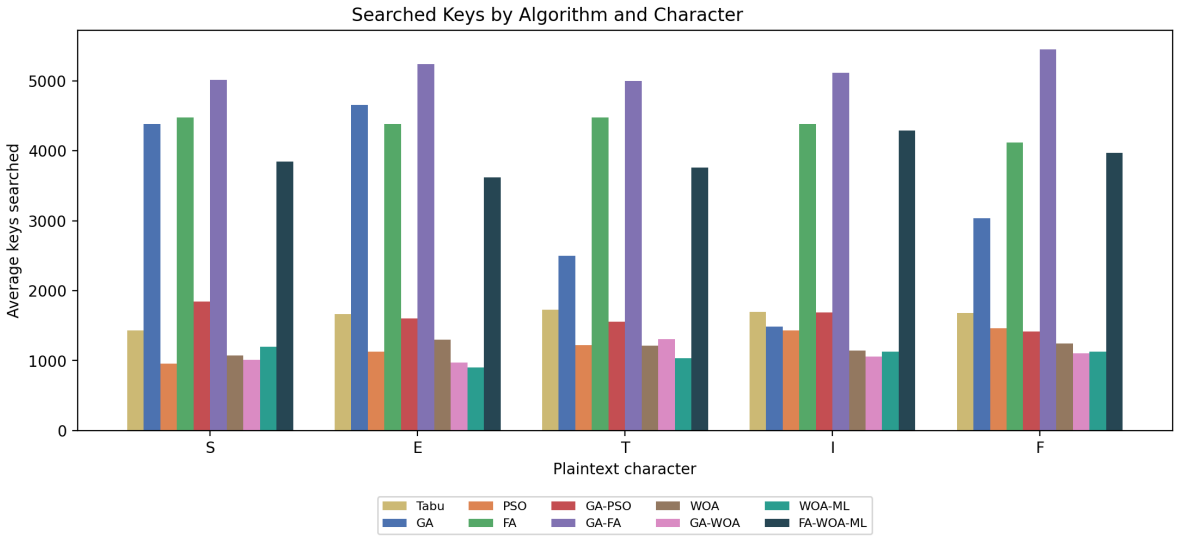


Figure 12. Average number of keys searched by algorithm and plaintext character, at $n = 10$ (baseline experiment of §3).

7. Conclusion

For this study, we developed GA-FA, a hybrid between GA and FA, for the cryptanalysis of the Hellman-Merkle cryptosystem, which represents a difficult NP-hard combinatorial optimization problem. This hybrid approach uses a genetic algorithm enhanced by an attraction-based local-search mechanism to improve the fitness of individuals; GA-FA is able to maintain high population diversity while minimizing the probability of premature convergence. GA-FA was compared with other state-of-the-art approaches – GA, PSO, FA, GA-PSO, and, further, WOA, GA-WOA, a novel multi-leader variant WOA-ML, an FA-WOA-ML hybrid, and Tabu Search – using a uniform, call-for-call fitness-evaluation counter, so that ”searched keys” means the same thing for every method, and justified

on complexity-theoretic grounds (§2.1) as an explore-then-exploit decomposition well matched to the rugged, exponentially-scaled fitness landscape of the public knapsack.

From the comparison experiments, all hybrid and swarm-based methods reach a 100% success rate at $n = 10$, while GA-PSO achieves this reliability with the lowest computational overhead among the hybrids, making it the recommended default when both success rate and efficiency matter at small key sizes. Extending the evaluation to the 5-character word "SETIF" (§3.3) showed that per-character success rates compound non-trivially over longer messages, and the key-size scalability study (§3.5) revealed a robustness ordering invisible at $n = 10$: GA-FA is the only method that remains reliable ($\geq 93\%$ success) up to $n = 20$ bits, while GA, PSO, and GA-PSO degrade sharply beyond $n \approx 14$, at the cost of a confirmed $O(P^2 n)$ per-generation complexity (§5). Plain WOA and its direct GA hybrid degraded even faster than GA/PSO, which a diagnostic analysis traced to an unbounded position blow-up that saturates their sigmoid discretization once every individual is pulled toward a single shared attractor (§3.6). Replacing that single attractor with a randomly-sampled pool of leaders (WOA-ML) substantially repaired this weakness without incurring FA's quadratic cost, and, at a larger key size ($n = 32$, §3.7), this cheaper $O(Pn + P \log P)$ update allowed WOA-ML to reach non-trivial success rates under a matched wall-clock budget precisely where every $O(P^2)$ method – GA-FA and FA-WOA-ML included – failed outright, isolating a genuine crossover point between per-generation cost and per-generation search quality. The parameter sensitivity study (§4) further showed that this robustness picture is conditional on reasonable hyperparameter choices: GA's success rate is highly sensitive to its mutation rate, PSO's to its inertia weight, and WOA-ML's to its leader-pool size K (itself following an inverted-U relationship), whereas hybridization generally buffers these individual weaknesses.

Taken together, GA-FA remains the most reliable choice across the key sizes where a full evaluation was feasible ($n \leq 20$), GA-PSO is preferable when speed matters at modest problem sizes, WOA-ML is the only method that remains productive, in a time-matched sense, once GA-FA's cost becomes prohibitive at larger n, and Tabu Search – despite the lowest robustness of any method besides plain GA – offers the lowest possible per-iteration cost and briefly rivals WOA-ML at very small time budgets before being overtaken as the budget grows. As our exploratory attempts at $n = 64$ and $n = 128$ confirm honestly for the four methods tested there (GA-FA, GA-PSO, WOA-ML, FA-WOA-ML), none constitutes a practical attack at cryptographically realistic key sizes, and none of the ten methods studied in this paper would be expected to fare better; closing this gap is the main direction for future work. Our ongoing research accordingly involves extending the scalability and time-matched-budget studies of §3.5–§3.7 toward such realistic key sizes, using greedy-seeded populations and adaptive, self-tuning hyperparameters informed by §4 (including an adaptive leader-pool size for WOA-ML), and further exploring whether the multi-leader mechanism can be combined with FA at less than FA's full $O(P^2)$ cost, for instance by restricting pairwise attraction to a leader subset rather than the whole population. We are also using evolutionary algorithms to attack other cryptographic schemes based on reference data sets, and we plan to design and implement a parallel hybrid evolutionary algorithm with the aim of discovering valid encryption keys more efficiently.

Acknowledgement

This work was completed with the support of DGRSDT-MESRS (Algeria).

REFERENCES

1. R. C. Merkle, and M. E. Hellman, *Hiding information and signatures in trapdoor knapsacks*, IEEE Transactions on Information Theory, vol. 24, no. 5, pp. 525–530, 1978. DOI: 10.1109/TIT.1978.1055927
2. R. Spillman, *Cryptanalysis of knapsack ciphers using genetic algorithms*, Cryptologia, vol. 17, no. 4, pp. 367–377, 1993. DOI: 10.1080/0161-119391867999
3. P. Garg, A. Shastri, and D. C. Agarwal, *An enhanced cryptanalytic attack on knapsack cipher using genetic algorithm*, Transaction on Engineering, Computing and Technology, vol. 1, no. 12, pp. 4071–4074, 2007. DOI: 10.5281/zenodo.1330067
4. M. F. AbdulHalim, B. A. Attea, and S. M. Hameed, *A binary particle swarm optimization for attacking knapsack cipher algorithm*, in Proc. 2008 Int. Conf. on Computer and Communication Engineering, Kuala Lumpur, Malaysia, pp. 77–81, 2008. DOI: 10.1109/ICCCE.2008.4580572

5. A. Jain, and N. S. Chaudhari, *Cryptanalytic results on knapsack cryptosystem using binary particle swarm optimization*, in Proc. Int. Joint Conf. SOCO'14-CISIS'14-ICEUTE'14, Advances in Intelligent Systems and Computing, vol. 299, Springer, Cham, 2014. DOI: 10.1007/978-3-319-07995-0_37
6. S. N. Sinha, S. Palit, M. A. Molla, A. Khanra, and M. Kule, *A cryptanalytic attack on knapsack cipher using differential evolution algorithm*, in Proc. 2011 IEEE Recent Advances in Intelligent Computational Systems, Trivandrum, India, pp. 317–320, 2011. DOI: 10.1109/RAICS.2011.6069326
7. S. Palit, S. N. Sinha, M. A. Molla, A. Khanra, and M. Kule, *A cryptanalytic attack on the knapsack cryptosystem using binary firefly algorithm*, in Proc. 2011 2nd Int. Conf. on Computer and Communication Technology (ICCCT), Allahabad, India, pp. 428–432, 2011. DOI: 10.1109/ICCCT.2011.6075143
8. M. Abdel-Basset, D. El-Shahat, I. El-henawy, A. K. Sangaiah, and S. H. Ahmed, *A novel whale optimization algorithm for cryptanalysis in Merkle-Hellman cryptosystem*, Mobile Networks and Applications, vol. 23, pp. 723–733, 2018. DOI: 10.1007/s11036-018-1024-4
9. N. Kantour, and S. Bouroubi, *Cryptanalysis of Merkle-Hellman cipher using parallel genetic algorithm*, Mobile Networks and Applications, vol. 25, pp. 211–222, 2020. DOI: 10.1007/s11036-019-01216-8
10. T. Mandal, and M. Kule, *An improved cryptanalysis technique based on Tabu search for knapsack cryptosystem*, International Journal of Control Theory and Applications, vol. 16, no. 9, pp. 8295–8302, 2016.
11. H. Singh, *Contravening esotery: cryptanalysis of knapsack cipher using genetic algorithms*, International Journal of Computer Applications, vol. 140, no. 6, pp. 26–33, 2016. DOI: 10.5120/ijca2016909333
12. A. Shamir, *A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem*, IEEE Transactions on Information Theory, vol. 30, no. 5, pp. 699–704, 1984. DOI: 10.1109/TIT.1984.1056964
13. J. C. Lagarias, and A. M. Odlyzko, *Solving low-density subset sum problems*, Journal of the ACM, vol. 32, no. 1, pp. 229–246, 1985. DOI: 10.1145/2455.2461
14. N. Hafsi, H. Hachimi, D. Benterki, and Y. Slimani, *Genetic-Firefly Algorithm Based Approach in Image Steganography*, in Proc. 2025 11th Int. Conf. on Optimization and Applications (ICOA), Kenitra, Morocco, 2025. DOI: 10.1109/ICOA66896.2025.11236921
15. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
16. X. S. Yang, *Firefly algorithms for multimodal optimization*, in Stochastic Algorithms: Foundations and Applications (SAGA 2009), Lecture Notes in Computer Science, vol. 5792, pp. 169–178, 2009. DOI: 10.1007/978-3-642-04944-6_14
17. S. Lukasik, and S. Zak, *Firefly algorithm for continuous constrained optimization tasks*, in Proc. ICCCI 2009, Lecture Notes in Artificial Intelligence, vol. 5796, pp. 97–100, 2009. DOI: 10.1007/978-3-642-04441-0_8
18. X. S. Yang, *Firefly algorithm, stochastic test functions and design optimisation*, International Journal of Bio-Inspired Computation, vol. 2, no. 2, pp. 78–84, 2010. DOI: 10.1504/IJBIC.2010.032124
19. X. S. Yang, *Biology-derived algorithms in engineering optimization*, in Handbook of Bioinspired Algorithms and Applications, S. Olariu and A. Y. Zomaya, Eds., Chapman and Hall/CRC, Boca Raton, FL, 2005.
20. R. Eberhart, and J. Kennedy, *A new optimizer using particle swarm theory*, in Proc. 6th Int. Symposium on Micro Machine and Human Science (MHS'95), Nagoya, Japan, pp. 39–43, 1995. DOI: 10.1109/MHS.1995.494215
21. M. Aly, and A. S. Alotaibi, *Hybrid Butterfly–Grey Wolf Optimization (HB-GWO): A Novel Metaheuristic Approach for Feature Selection in High-Dimensional Data*, Statistics, Optimization and Information Computing, vol. 13, no. 6, pp. 2575–2600, 2025. DOI: 10.19139/soic-2310-5070-2617
22. B. Meryem, A. Toulaoui, I. Hafidi, and Y. Rochd, *OGA-Apriori: An Optimized Genetic Algorithm Approach for Enhanced Frequent Itemset Mining*, Statistics, Optimization and Information Computing, vol. 14, no. 6, pp. 3144–3161, 2025. DOI: 10.19139/soic-2310-5070-2320
23. G. Basheer, L. Mohammed, and Z. Algamal, *Solving 0–1 Knapsack Problem by an Improved Binary Monarch Butterfly Algorithm*, Statistics, Optimization and Information Computing, vol. 15, no. 4, pp. 2382–2396, 2025. DOI: 10.19139/soic-2310-5070-2965
24. N. Hafsi, H. Hachimi, and D. Benterki, *A New Hybrid Genetic with Firefly Algorithm for Solving Cyber-Criminal's Attacks*, in Proc. 2022 8th Int. Conf. on Optimization and Applications (ICOA), Genoa, Italy, 2022. DOI: 10.1109/ICOA55659.2022.9934759
25. C. R. Reeves, and J. E. Rowe, *Genetic Algorithms – Principles and Perspectives: A Guide to GA Theory*, Kluwer Academic Publishers, Dordrecht, 2003.
26. G. Ward, *Moby Word Lists*, Project Gutenberg, 1996. Available: https://www.gutenberg.org/ebooks/3201
27. P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. K. Greene, and M. F. Theofanos, *Digital Identity Guidelines: Authentication and Lifecycle Management*, NIST Special Publication 800-63B, National Institute of Standards and Technology, 2017. DOI: 10.6028/NIST.SP.800-63b