Comparing AI Algorithms for Optimizing Elliptic Curve Cryptography Parameters in e-Commerce Integrations: A Pre-Quantum Analysis

Felipe Tellez, Jorge Ortíz
Department of Systems and Industrial Engineering, National University of Colombia, Bogotá, Colombia 111321

***Abstract*—This paper presents a comparative analysis between the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), two vital artificial intelligence algorithms, focusing on optimizing Elliptic Curve Cryptography (ECC) parameters. These encompass the elliptic curve coefficients, prime number, generator point, group order, and cofactor. The study provides insights into which of the bio-inspired algorithms yields better optimization results for ECC configurations, examining performances under the same fitness function. This function incorporates methods to ensure robust ECC parameters, including assessing for singular or anomalous curves and applying Pollard's rho attack and Hasse's theorem for optimization precision. The optimized parameters generated by GA and PSO are tested in a simulated e-commerce environment, contrasting with well-known curves like secp256k1 during the transmission of order messages using Elliptic Curve-Diffie Hellman (ECDH) and Hash-based Message Authentication Code (HMAC). Focusing on traditional computing in the pre-quantum era, this research highlights the efficacy of GA and PSO in ECC optimization, with implications for enhancing cybersecurity in third-party e-commerce integrations. We recommend the immediate consideration of these findings before quantum computing's widespread adoption.**

***Keywords*—Artificial intelligence; genetic algorithms; particle swarm optimization; elliptic curve cryptography; e-commerce; third-party integrations; pre-quantum computing**

I. Introduction

This paper explores the field of Elliptic Curve Cryptography (ECC), a form of public key cryptography that uses the mathematics of elliptic curves to secure transactions, specifically focusing on its application within e-commerce transactions executed through third-party integrations during a pre-quantum computing era. The aim is to identify the most efficient and effective Artificial Intelligence (AI) algorithm for optimizing parameters essential to ECC's successful operation within this context. The two leading algorithms being examined are Genetic Algorithms (GA), and Particle Swarm Optimization (PSO).

A. Background

Elliptic Curve Cryptography (ECC), an important and widely used form of public-key cryptography [1][2], provides enhanced security with shorter key lengths [3], making it ideal for resource-constrained environments like e-commerce platforms [34]. Effective ECC operation hinges on the careful selection of parameters such as curve coefficients, base point, prime modulus, and others [4] to [11]. Optimizing these parameters can enhance ECC's security and efficiency, which is crucial in e-commerce transactions.

Artificial Intelligence (AI) has shown tremendous potential in this parameter optimization [12] to [26]. Notably, two AI algorithms, -GA, and PSO- stand out [12][15]. They represent different subsets and categories of AI algorithms: GA, an Evolutionary Algorithm [18], and PSO, a Swarm Intelligence [19], belong to Population-based Optimization. These algorithms are recognized for their problem-solving and optimization capabilities.

E-commerce transactions often involve integrations with third-party solutions such as Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) systems, payment gateways, data analytics solutions, among others [27][28]. These transactions need to be securely encrypted, making ECC an excellent choice, especially with optimized parameters.

B. Objective

The objective of this paper is to compare the efficacy of GA and PSO in optimizing ECC parameters for e-commerce transactions involving third-party integrations within binary computing. Specifically, the comparison aims to determine which AI algorithm is most efficient in terms of its behavior during the optimization process, and which AI algorithm is most effective in terms of the quality of the results it produces.

C. Scope and Limitations

While ECC can be used in contexts other than e-commerce third-party integrations, such as Web browsers, customer authentication, administrative user authentication, and database persistence, the focus of this article is on transaction integrations with third parties. Interactions between e-commerce systems and backend or third-party solutions such as ERPs, CRMs, payment gateways, and billers are all part of it.

This research uses a simulated e-commerce environment that incorporates a business process that entails creating orders in an emulated ERP. The information for these orders is sent from the e-commerce solution through third-party integrations using web services to imitate real-world conditions. The research relies on API simulations for outbound connections utilizing datasets relevant to these types of scenarios rather than a whole e-commerce solution.

The research also limits its scope to the pre-quantum computing era. Although quantum computing promises significant advances, its implications for ECC and AI algorithms are beyond this study's scope. The research also excludes other AI

techniques like Simulated Annealing, Ant Colony Optimization or Artificial Neural Networks, despite their applicability to ECC optimization, to keep the research focused.

D. Structure and Contributions

The rest of this paper is organized as follows: Section II reviews related work on ECC optimization, AI techniques, e-commerce integrations, and pre-quantum developments. Section III details materials and methods, including ECC parameters and optimization criteria. Section IV describes the simulation environment design and implementation. Section V presents results and analysis, covering AI algorithm execution, e-commerce simulation, and comparison based on ECC criteria. Section VI discusses future improvements and limitations, including parameter tuning, parallelization, hybrid algorithms, alternative AI techniques, fitness function improvements, diverse cryptographic threats, and quantum computing implications. Section VII concludes with future work and recommendations. Unique contributions include a detailed comparison of GA and PSO for ECC optimization in e-commerce, a novel fitness function, and an evaluation framework for pre-quantum computing.

II. Literature Review

A. ECC Parameter Optimization

The complexity of Elliptic Curve Cryptography (ECC) optimization is central to research on ECC systems' effectiveness, security, and efficiency. Koblitz [1] and Miller [2] independently introduced Elliptic Curves in public-key cryptography, stressing the careful choice of parameters for enhanced security. Washington [3] emphasized optimizing ECC parameters such as curve coefficients, base point, prime modulus, and key sizes, all influencing ECC's performance.

Lenstra and Verheul [4] advocated ECC's use in cryptographic systems, with a focus on proper parameter selection, especially prime modulus. Blake, Seroussi, and Smart [5] delved into the details of ECC parameter selection, highlighting the selection of the base point, curve coefficients, and prime modulus.

A significant aspect of ECC optimization is speeding up point multiplication, a core ECC operation. Hankerson, Menezes, and Vanstone [6] outlined strategies for this focusing on ECC's computational aspects. Other researchers have examined ECC optimization in MANET and Sensor networks, where hardware plays a significant role. In [7], we can find a state of the art of ECC optimizations in these types of scenarios.

In general, the literature review emphasizes ECC parameter optimization's importance and complexity, covering domains such as elliptic curves' mathematical foundations, intricate parameter selection, implementation optimizations, and AI algorithms for multi-objective optimization [8] to [11]. This substantial knowledge forms the foundation of this study.

B. AI Techniques for ECC Parameter Optimization

Artificial Intelligence (AI) exhibits vast potential in ECC parameter optimization, with prominent techniques like Genetic Algorithms (GA) [12] to [14] and Particle Swarm Optimization (PSO) [15] to [17]. These methods contribute distinctive strengths to ECC optimization.

GA, inspired by biological evolution, is known for its effectiveness in exploring complex search spaces, particularly in seeking optimal solutions within intricate landscapes like ECC parameter optimization [18]. On the other hand, PSO, modeled after the social behaviors of birds and fishes, is celebrated for its simple implementation and intrinsic ability to avoid local optima [19]. These methods, by emulating natural processes, present unique solutions to ECC's challenges, highlighting the connection between nature's complexity and technological innovation.

Besides these techniques, others like Simulated Annealing (SA) [20], an Stochastic Optimization inspired by the annealing process in metallurgy, is known for adaptability and robustness in solving optimization issues [21], including ECC. Evolutionary Algorithms (EA), similar to GA, involve mechanisms like reproduction and mutation, showing promise in optimizing Elliptic Curves [22][23]. Machine Learning (ML), where algorithms evolve through data usage, has been applied to Elliptic Curve factorization problems [24]. Also, Tabu Search has been used to enhance ECC operations and multimedia encryption [25] [26].

Each of these named AI methods offers unique benefits in the optimization of ECC parameters; nevertheless, as we mentioned in the introduction section, the focus of our analysis will be on GA, and PSO.

C. E-commerce and Third-party Integrations

The growth of e-commerce has fostered an interconnected technological network involving various third-party entities such as payment gateways, ERP systems, CRM systems, billers, web services, and custom solutions [27][28]. They exchange vital operational data, including inventory status, billing data, orders information and more.

1) Types of e-commerce integrations: To better understand e-commerce integrations, it's crucial to consider their directionality, distinguishing between inbound and outbound integrations [29] [30].

- *Inbound Integrations:* SaaS-based e-commerce platforms [31] usually offer an integration layer based on Application Programming Interfaces (APIs). These APIs, typically RESTful web services [32][33], are exposed for third-party consumption. They are provided "out of the box," ready to be used by external requesters or legacy systems such as ERPs.
- *Outbound Integrations:* These emanate from e-commerce platforms to an external entity and are usually executed through webhooks. These triggers send detailed order information to third-party entities such as the ERP system.

2) The Role of AI in ECC and Third-Party Integrations: As e-commerce evolves, secure and efficient third-party integrations are essential. ECC maintains data security and integrity across these integrations [34]. AI techniques optimize ECC parameters, boosting transaction speed, data security, and overall user experience, offering a competitive edge in e-commerce operations.

D. Pre-Quantum Developments in ECC Optimization

The advent of quantum computing ushers in a new era with its potential to solve complex problems more efficiently than classical computers [35]. However, the implications of this quantum leap for Elliptic Curve Cryptography (ECC) and its parameter optimization using AI algorithms remain largely speculative, as quantum computing is yet to become mainstream. The pre-quantum era, thus, serves as the current framework within which ECC optimization techniques are developed and implemented, focusing on the capabilities of classical computing.

E. Limitations of Similar Research

Although previous studies have made significant contributions to the field of ECC cryptanalysis and security, they primarily focus on specific techniques like Pollard's Rho, DNA-based cryptography, PSO/Cuckoo Search for key generation, and power optimization for mobile devices. These studies do not explore other AI techniques for optimizing ECC parameters, particularly in the context of e-commerce integrations. Furthermore, there is a lack of consideration for the practical applications of these optimizations in real-world scenarios. The limitations of these studies are summarized in Table I.

TABLE I. LIMITATIONS OF SIMILAR RESEARCH

Study	Limitations
[12]	Focuses on cryptanalysis rather than optimization of ECC parameters for practical applications. Does not explore other AI techniques or their integration with e-commerce systems.
[13]	Concentrates on multi-cloud security using DNA and HECC techniques but does not explore other AI techniques like GA or PSO for ECC optimization. Lacks practical implementation details for e-commerce integrations.
[16]	Focuses on mobile devices and optimizing power consumption using PSO and Simplified Swarm Optimization. Does not provide a comprehensive comparison with other AI techniques like GA for ECC optimization. The study's focus on mobile device constraints limits its applicability to broader e-commerce integrations.
[17]	Explores PSO and Cuckoo Search Algorithm for ECC key selection but does not provide a comprehensive comparison with other AI techniques like GA. Focuses more on key generation rather than overall ECC parameter optimization in e-commerce contexts.

III. MATERIALS AND METHODS

This section highlights our research methodology, focusing on the ECC parameters to optimize and the criteria for the evaluation of AI techniques.

A. ECC Optimization Parameters

Elliptic Curve Cryptography parameters play distinctive roles, and they can be carefully tuned to improve ECC without sacrificing security. The parameters that will be analyzed for this study are as follows:

1) Choice of elliptic curve: The curve's equation $E : y^2 = x^3 + ax + b$ and specific constants a and b (curve coefficients) determine the system's efficiency and security.

2) Field size: Represented by a prime number (p), the field size affects security and computational load. Larger fields enhance security but need careful balancing with efficiency.

3) Generator point G: The method used for representing points (x, y) on the curve affects computation speed.

4) Scalar multiplication: Techniques like the Montgomery ladder [36] or sliding window method [37] enhance ECC operations. The operation $Q = kP$, where P is a point on the curve and k is scalar, can be optimized for efficiency.

5) Group order n: This represents the number of points on the elliptic curve and plays a vital role in the security of the ECC system.

6) Cofactor h: The ratio between the number of points on the curve and the group order n. It's essential in defining the subgroup that is used for cryptographic purposes.

The parameters mentioned above represent only a fraction of the many that can be considered [1] to [11]. Other aspects, such as Hash Function, Pairing Function, Random Number Generation, protocol parameters, use of special curves, batch operations, endomorphism $(\phi : E \to E)$, parallelism, efficient arithmetic libraries, hardware acceleration, and more, will not be discussed to maintain the focus of the study.

B. ECC Optimization Criteria

The efficiency and effectiveness, collectively referred to as the efficacy, of the selected AI algorithms in optimizing the ECC parameters, are assessed based on multiple criteria acknowledged as vital evaluation measures by the broader research community. These criteria encompass various aspects that together represent the complete performance of ECC. Below, we outline these criteria:

1) Evaluation of the AI Algorithms (efficiency):

a) Performance: Speed, convergence rate, computational time.

b) Flexibility: Ability to adapt to different problems or changes in the landscape.

c) Robustness: Sensitivity to initial conditions, parameter settings, and noise.

d) Scalability: Ability to handle increasing complexity or problem size.

e) Comparability: Fairness and alignment in comparing the two algorithms.

2) Evaluation of the ECC Parameters Generated (effectiveness):

a) Security: Resistance against attacks, adherence to cryptographic best practices.

b) Optimality: How close the parameters are to the theoretical best solution.

c) Generalization: Effectiveness across different curve configurations and real-world scenarios.

d) Validity: Compliance with mathematical and cryptographic requirements, such as avoiding singular or anomalous curves.

e) Practicality: Consideration of real-world applications, computational performance, and compatibility with existing systems.

The two aspects of efficiency and effectiveness, are interconnected in efficacy but evaluate different dimensions of the problem. Efficiency focuses on the algorithms themselves and how they perform as optimization techniques [38] to [43], while effectiveness concentrates on the quality and characteristics of the ECC parameters they produce [1] to [11].

IV. Simulation Environment Design and Implementation

The implementation of our simulation consists of an environment of applications and software modules (hereafter referred to as components), built using the Python programming language. These are divided into two main groups: "ECC Params Optimization" and "e-commerce Simulation". The architecture of this environment is illustrated in Fig. 1.

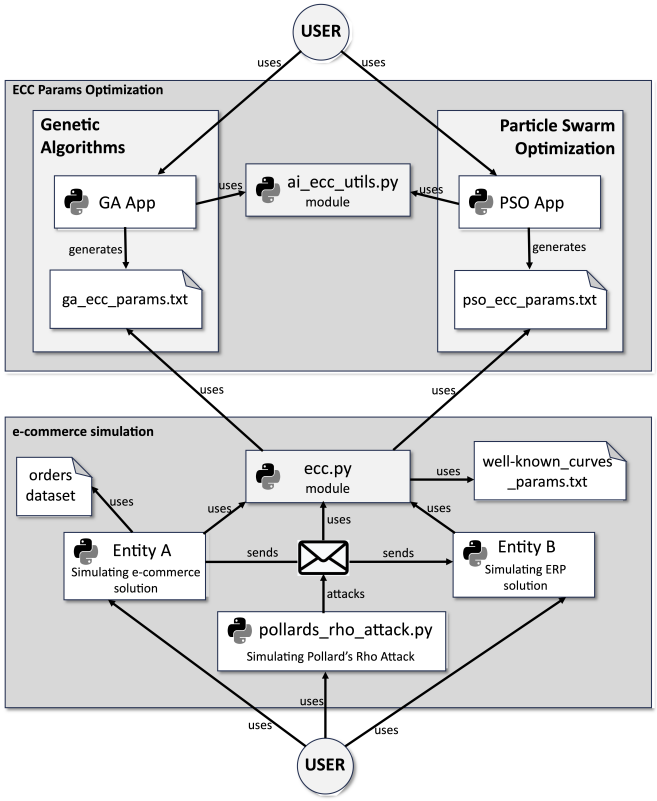

Fig. 1. A high-level diagram of the environment's architecture.

The libraries used in the project include, but are not limited to, 'numpy', 'pandas', 'matplotlib', 'deap', 'gmpy2', 'requests', and 'tinyec'. These libraries, along with the detailed source code, can be found at: github.com/cftellezc/GA_PSO_ECC_parameter_Optimization

The components of our simulation environment are described in more detail below.

A. ECC Params Optimization Group

1) Genetic algorithm: The GA.py script or GA App, employs the DEAP (Distributed Evolutionary Algorithms in Python) library to implement a genetic algorithm for ECC parameter optimization. It initiates a population of individuals, where each individual is a list representing potential elliptic curve parameters in ECC. This parameters are the constants a and b, the prime number p, the generator point G representing points (x, y), the group order n and the cofactor h. Through iterative genetic operations like selection, crossover, and mutation, new generations of individuals are produced.

The script uses a custom mutation function that receives the individual, an independent probability *indpb* (the chance of each attribute to be mutated), and the mutation rate, all of this to mutate the individuals generating prime numbers or perturbing parameters using a Gaussian distribution as follows (Algorithm 1):

Algorithm 1 Custom Mutation Function

```
function CUSTOMMUTATION(individual, indpb, mutation_rate)
    degree_of_mutation ← 5
    if mutation_rate > 0.5 then
        degree_of_mutation ← 10
    else
        degree_of_mutation ← 2
    end if
    if random value between 0 and 1 < mutation_rate then
        for i = 0 to length of individual − 1 do
            if random value between 0 and 1 < indpb then
                if i = 2 then
                    individual[i] ← generate prime number for p
                else if individual[i] is a tuple then
                    individual[i] ← find generator point using individual[0], individual[1], individual[2]
                else
                    individual[i] ← individual[i] + round(value from Gaussian distribution with mean 0 and standard deviation degree_of_mutation)
                end if
            end if
        end for
    end if
    return individual,
end function
```

The script defines several global constants that control the behavior of the genetic algorithm, as summarized in Table II:

TABLE II. Genetic Algorithm Constants

Constant	Value	Description
POP_SIZE	500	Population size
CXPB	0.5	Crossover probability
MUTPB	0.2	Mutation probability
NGEN	40	Number of generations
MULTIPARENT_CXPB	0.1	Multi-parent crossover probability
ELITISM_RATE	0.1	Elitism rate

They were set up and tuned for better performance using some techniques like grid search. They can be adjusted to tune the performance of the algorithm.

Executed as the main module, the script's primary function initializes the population, assesses their fitness, and enters a loop for generating and evaluating new individuals over a specified number of generations (*NGEN*) as follows (Algorithm 2):

Algorithm 2 Main Genetic Algorithm

Initialize: $pop \leftarrow$ toolbox.population()
for all individual in pop **do**
    Evaluate fitness and assign to individual
end for
Initialize: $elitism\_number \leftarrow$ round$(len(pop) \times ELITISM_RATE)$
$mutation\_rate \leftarrow MUTPB$
for $g = 0$ to $NGEN - 1$ **do**
    Log: Starting Generation: $g + 1$
    $elites \leftarrow$ select top individuals from pop
    $offspring \leftarrow$ select next generation from pop
    Clone: offspring
    for $i = 0$ to $len(offspring) - 3$ **step** 3 **do**
        Apply three-point crossover if random() $< MULTIPARENT\_CXPB$
    end for
    for all pair $child1, child2$ in offspring **do**
        Apply crossover to $child1, child2$ if random() $< CXPB$
    end for
    for all mutant in offspring **do**
        mutate mutant with rate $mutation\_rate$
    end for
    Evaluate and set fitness of invalid individuals
    $offspring \leftarrow offspring + elites$
    $pop \leftarrow offspring$
end for

Summarizing, we initialize the mutation degree utilizing the standard deviation of a Gaussian distribution, and adjusting it according to the *mutation_rate*. This procedure entails creating random floats to determine the probability of mutation, which is influenced by both the mutation and *indpb* rates.

Individual parameters are subject to potential mutation, adhering to unique criteria for distinct cases: the third parameter (p) entails deriving a fresh prime number utilizing the *BITS_PRIME_SIZE* constant — currently set to 256 bits — from the *ai_ecc_utils.py* module. Tuple parameters representing generator points (G) necessitate fabricating a new point using the prevailing values of a, b, and p.

The remaining parameters (a and b) undergo modifications through Gaussian perturbations, with an imperative to retain integer attributes facilitated by the round function. This mechanism ensures compatibility with DEAP, expecting mutated individuals to be returned as single-item tuples.

The script uses tournament selection and two-point crossover from the DEAP library. It also implements elitism, ensuring that the best individuals from each generation are carried over to the next.

The script evaluates the fitness of the individuals using a function from the *ai_ecc_utils.py* module (which will be explained later), which calculates the fitness based on the ECC parameters represented by the individual. The script logs the progress of the genetic algorithm, including the statistics of each generation and the best individual from the final generation.

In the context of *ECC*, the individuals in the population represent different elliptic curves, and the fitness function assesses how well they meet the desired criteria. The genetic algorithm identifies the best-fitting elliptic curve and generates a file named *ga_ecc_params.txt*, containing the optimal parameters for *ECC* optimization.

2) Particle swarm optimization: The PSO.py module or PSO App, uses a Particle Swarm Optimization (PSO) algorithm to fine-tune ECC parameters. It initializes particles, where each particle is a list representing potential elliptic curve parameters in ECC, updates their velocities and positions, evaluates fitness, and identifies the best ECC parameter set through the optimal particle. The ECC parameters are the same as those assessed in GA: (a, b, p, G, n, h).

The *update_velocity* function calculates the new velocity of a particle. It balances global and local exploration using a dynamic inertia weight that linearly decreases from 0.9 to 0.4 over iterations. Two components contribute to the velocity: a cognitive component based on the particle's best-known position $C1$, and a social component based on the swarm's best-known position $C2$. Special handling is done for the generator point of the elliptic curve, as shown in Algorithm 3.

Algorithm 3 Update Velocity of a Particle

procedure UPDATEVEL(part, vel, best_part, glob_best_part, iter, max_iter)
    $new\_vel \leftarrow []$
    $w_{\text{max}} \leftarrow 0.9$
    $w_{\text{min}} \leftarrow 0.4$
    $w \leftarrow w_{\text{max}} - (w_{\text{max}} - w_{\text{min}}) \cdot \frac{\text{iter}}{\text{max\_iter}}$
    for $i = 0$ **to** len(part) $- 1$ **do**
        $v \leftarrow$ vel$[i]$
        $r1, r2 \leftarrow$ random(0, 1)
        if $i = 3$ **then**
            $cog \leftarrow$ calc_cog$(best_part[i], part[i], r1)$
            $soc \leftarrow$ calc_soc$(glob_best_part[i], part[i], r2)$
            if is_tuple(v) **then**
                $new\_v \leftarrow$ calc_new_v_tuple(v, w, cog, soc)
            else
                $new\_v \leftarrow$ calc_new_v(cog, soc)
            end if
        else
            $cog \leftarrow C1 \cdot r1 \cdot (best\_part[i] - part[i])$
            $soc \leftarrow C2 \cdot r2 \cdot (glob\_best\_part[i] - part[i])$
            $new\_v \leftarrow w \cdot v + cog + soc$
            if $i \neq 3$ **then**
                $new\_v \leftarrow$ abs(new_v)
            end if
        end if
        Append new_v **to** new_vel
    end for
    return new_vel
end procedure

The *update_position* function calculates the new position of a particle based on its velocity. It ensures that coordinates remain positive and integers, and specific attention is given to update the generator point. It also makes use of the external utility functions from *ai_ecc_utils.py* to update the prime number and generator point, as shown in Algorithm 4.

Algorithm 4 Update Position of a Particle

procedure UPDATEPOSITION(particle, velocity)
    new_particle ← empty list
    for $i = 0$ to len(particle) $- 1$ **do**
        $p \leftarrow$ particle$[i]$
        $v \leftarrow$ velocity$[i]$
        if $i = 3$ **then**
            new_p ← tuple($|\text{int}\,(\text{round}(p_i + v_i))|$ for p_i, v_i in zip(p[:2], v[:2]))
        else
            new_p ← $|\text{int}\,(\text{round}(p + v))|$
        end if
        Append new_p to new_particle
    end for
    new_particle[2] ← ai_ecc_utils.get_prime_for_p()
    a, b, p ← new_particle[:3]
    new_particle[3] ← ai_ecc_utils.find_generator_point(a, b, p)
    return new_particle
end procedure

The script specifies several global constants that regulate how the Particle Swarm Optimization algorithm behaves, as summarized in Table III:

TABLE III. PARTICLE SWARM OPTIMIZATION CONSTANTS

Constant	Value	Description
SWARM_SIZE	500	Number of particles in the swarm
MAX_ITERATIONS	40	Maximum number of iterations
C1	1.0	Cognitive parameter (influence of particle's best-known position)
C2	2.5	Social parameter (influence of swarm's best-known position)
MAX_ITERATIONS_ WITHOUT_ IMPROVEMENT	20	Used for an early stopping feature

The implementation includes a parameter grid for tuning the PSO constants.

The *main function* initializes the swarm of particles, their velocities, and their best-known positions. Global best-known positions are also identified. The *main loop* iterates through the swarm, updating velocities and positions using the previously defined functions. Fitness is evaluated for each particle using the same fitness function from the *ai_ecc_utils.py* module that is used by *GA.py* (which will be explained later), and best-known positions are updated as necessary. If there is no improvement in global best fitness for 20 iterations, the algorithm stops early, and at the end, statistics regarding fitness values are calculated and printed. The best particle is selected, and its details are printed and written to the *pso_ecc_params.txt* file.

3) ga_ecc_params.txt: This file contains the best parameters found by the Genetic Algorithm (GA) for ECC parameter optimization.

4) pso_ecc_params.txt: Represents the file with the best ECC parameters found by the Particle Swarm Optimization (PSO) technique.

5) ai_ecc_utils.py: It is a utility module that aids AI algorithms like GA and PSO in the process of ECC parameter optimization. The module's primary purpose is to facilitate the creation of elliptic curves and their associated parameters, as shown in Algorithm 5.

Algorithm 5 Elliptic Curve Parameter Generation

procedure GENERATE_CURVE
    signal.signal(signal.SIGALRM, handler)
    while True **do**
        $p \leftarrow$ get_prime_for_p()
        while True **do**
            logging.info("a, b generation")
            $a \leftarrow$ random.randint$(0, p-1)$
            $b \leftarrow$ random.randint$(0, p-1)$
            if $(4 \cdot a^3 + 27 \cdot b^2) \mod p \neq 0$ and not is_singular(a, b, p) **then**
                break
            end if
        end while
        try:
        signal.alarm(TIMEOUT_SECONDS)
        $G \leftarrow$ find_generator_point(a, b, p)
        logging.info("$G :$ ", G)
        signal.alarm(0)
        break
        except NoGeneratorPointException, TimeoutError :
        continue
    end while
    $n \leftarrow p - 1$
    $h \leftarrow 1$
    return (a, b, p, G, n, h)
end procedure
procedure GET_PRIME_FOR_P
    return getPrime(BITS_PRIME_SIZE)
end procedure
procedure IS_SINGULAR(a, b, p)
    discriminant ← $(4 \cdot a^3 + 27 \cdot b^2) \mod p$
    return discriminant == 0
end procedure
procedure FIND_GENERATOR_POINT(a, b, p)
    for x in 0 to $p - 1$ **do**
        rhs ← $(x^3 + a \cdot x + b) \mod p$
        if legendre_symbol(rhs, p) == 1 **then**
            $y \leftarrow$ tonelli_shanks(rhs, p)
            return (x, y)
        end if
    end for
    raise NoGeneratorPointException
end procedure

It includes functions for generating prime numbers and finding generator points on the elliptic curve using mathematical functions, such as the Legendre symbol and Tonelli-Shanks algorithm [44]. The Legendre symbol determines whether a number is a quadratic residue modulo a prime, essential for finding valid points on the elliptic curve. The Tonelli-Shanks algorithm finds the square root of a number modulo a prime,

crucial for computing the y-coordinates of the points on the curve. These methods ensure the generated points are valid and lie on the elliptic curve, as shown in Algorithm 6.

Algorithm 6 Legendre Symbol and Tonelli-Shanks Algorithm

procedure LEGENDRE_SYMBOL(a, p)
    $ls \leftarrow \text{pow}(a, (p-1) \div 2, p)$
    return -1 if $ls == p-1$ else ls
end procedure
procedure TONELLI_SHANKS(n, p)
    assert legendre_symbol(n, p) == 1,"n is not a quadratic residue modulo p"
    $q \leftarrow p-1$
    $s \leftarrow 0$
    while $q \mod 2 == 0$ **do**
        $q \div = 2$
        $s \leftarrow s+1$
    end while
    if $s == 1$ **then**
        return $\text{pow}(n, (p+1) \div 4, p)$
    end if
    for z from 2 to $p-1$ **do**
        if legendre_symbol$(z, p) == -1$ **then**
            break
        end if
    end for
    $m \leftarrow s$
    $c \leftarrow \text{pow}(z, q, p)$
    $t \leftarrow \text{pow}(n, q, p)$
    $r \leftarrow \text{pow}(n, (q+1) \div 2, p)$
    while $t \neq 1$ **do**
        $i \leftarrow 0$
        $t_i \leftarrow t$
        while $t_i \neq 1$ **do**
            $t_i \leftarrow \text{pow}(t_i, 2, p)$
            $i \leftarrow i+1$
        end while
        $b \leftarrow \text{pow}(c, 2^{m-i-1}, p)$
        $r \leftarrow r \cdot b \mod p$
        $t \leftarrow t \cdot b \cdot b \mod p$
        $c \leftarrow b \cdot b \mod p$
        $m \leftarrow i$
    end while
    return r
end procedure

The ai_ecc_utils.py module validates ECC parameters, checking cofactor, prime p, point validity, handling generator point exceptions, matching order and cofactor, and confirming non-singular, anomalous, or supersingular characteristics [45], as shown in Algorithm 7. These validations ensure the integrity and security of the ECC parameters used in the cryptographic system.

The module also implements Pollard's rho attack [45] to evaluate the security of the generated ECC parameters. This attack is a well-known method for finding discrete logarithms in elliptic curves, making it an essential tool for assessing the resilience of the cryptographic system against specific types of attacks. It employs functions to add two points on an elliptic curve, apply the "double and add" method for point multiplication, and check if a point is "distinguished"

Algorithm 7 Validation and Properties of Elliptic Curve

procedure VALIDATE_CURVE(a, b, p, G, n, h)
    if $h < 1$ **then**
        log "The cofactor h is less than 1, which makes it invalid."
        return False
    end if
    if $p == 0$ **then**
        log "The prime p can't be zero."
        return False
    end if
    if len(G) == 2 **then**
        $x, y \leftarrow G$
        if $(y \cdot y - x \cdot x \cdot x - a \cdot x - b) \mod p \neq 0$ **then**
            log "The point G is not on the curve!"
            return False
        end if
        $field \leftarrow$ SubGroup with (p, G, n, h)
        if No generator point in $field$ **then**
            log "No generator point found!"
            return False
        end if
        $curve \leftarrow$ Curve with $(a, b, field,$ "random_curve")
    else
        log "Invalid generator point provided. Skipping curve creation."
        return False
    end if
    $order \leftarrow n$
    if $h \neq field.h$ **then**
        log "The cofactor does not match the expected cofactor!"
        return False
    end if
    if IS_SINGULAR(a, b, p) **then**
        log "The curve is singular!"
        return False
    end if
    if IS_ANOMALOUS$(p, order)$ **then**
        log "The curve is anomalous!"
        return False
    end if
    if IS_SUPERSINGULAR$(p, order)$ **then**
        log "The curve is supersingular!"
        return False
    end if
    return True
end procedure
procedure IS_SINGULAR(a, b, p)
    $discriminant \leftarrow (4 \cdot a^3 + 27 \cdot b^2) \mod p$
    return $discriminant == 0$
end procedure
procedure IS_ANOMALOUS(p, n)
    return $p == n$
end procedure
procedure IS_SUPERSINGULAR(p, n)
    if $p \in [2, 3]$ or not isprime(p) **then**
        return False
    end if
    return $(p+1-n) \mod p == 0$
end procedure

by having t trailing zeros in its *x-coordinate*, as shown in Algorithm 8.

Algorithm 8 Pollard's Rho Attack on an Elliptic Curve

function P_RHO_ATTACK($G, a, b, p,$ order$, t,$ max_iter)
  $Q_a, Q_b \leftarrow G, G$
  $a, b \leftarrow 0, 0$
  power_of_two $\leftarrow 1$
  iterations $\leftarrow 0$
  while iterations $<$ max_iterations **do**
    for _ in range(power_of_two) **do**
      $i \leftarrow Q_a[0] \mod 3$
      if $i = 0$ **then**
        $Q_a \leftarrow$ add_points(Q_a, G, a, p)
        $a \leftarrow (a + 1) \mod$ order
      else if $i = 1$ **then**
        $Q_a \leftarrow$ double_and_add$(2, Q_a, a, p)$
        $a \leftarrow (2 \cdot a) \mod$ order
      else
        $Q_a \leftarrow$ double_and_add$(2, Q_a, a, p)$
        $a \leftarrow (2 \cdot a) \mod$ order
        $Q_a \leftarrow$ add_points(Q_a, G, a, p)
        $a \leftarrow (a + 1) \mod$ order
      end if
      if is_distinguished(Q_a, t) **then**
        return a, Q_a
      end if
    end for
    for _ in range(2) **do**
      Repeat the same steps for Q_b
       , but twice per iteration
    end for
    iterations $\leftarrow$ iterations $+ 1$
    if $Q_a = Q_b$ **then**
      power_of_two $\leftarrow$ power_of_two $\times 2$
      $Q_b \leftarrow Q_a$
      $b \leftarrow a$
    end if
  end while
  logging.info($''$No collision found within the)
  specified maximum number of iterations.$''$)
  return None
end function

Lastly, the ai_ecc_utils.py module calculates the fitness function, incorporating all the elliptic curve validations. It evaluates the fitness of a candidate, whether an "individual" in the GA population or a "particle" in the PSO swarm, as shown in Algorithm 9.

The fitness function extracts the elliptic curve parameters (a, b, p, G, n, h). *Curve Validation* checks if the parameters form a valid curve, returning a fitness of 0 if not. The expected order of the curve is calculated, checks Hasse's theorem bounds [44] and the Hasse score is computed to evaluate how close the actual order is to the expected. *Pollard's Rho Attack* is attempted, with a longer execution time indicating higher resistance. An *Attack Resistance Score* is assigned. The *Final Fitness Calculation* includes 40% weight to the natural logarithm of the curve's order, 20% to the Hasse score (weighted by the logarithm of the order), 20% to the execution time score of the attack, and 20% to the resistance score. The

Algorithm 9 Function to evaluate the fitness of a candidate in GA or PSO

function EVALUATE(candidate)
  Extract a, b, p, G, n, h from candidate
  if not VALIDATE_CURVE(a, b, p, G, n, h) **then**
    return 0
  end if
  $expected\_order \leftarrow p + 1 - 2 \cdot \sqrt{p}$
  $upper\_bound \leftarrow expected\_order + 2 \cdot \sqrt{p}$
  $hasse\_score \leftarrow \max\left(0, \frac{upper\_bound - |n - expected\_order|}{upper\_bound - lower\_bound}\right)$
  $start\_time \leftarrow$ current time
  $rho\_attack\_result \leftarrow$ P_RHO_ATTACK$(G, a, b, p, expected_order)$
  $execution\_time \leftarrow$ current time $- start\_time$
  $max\_time \leftarrow 10.0, min\_time \leftarrow 0.1$
  $execution\_score \leftarrow \max\left(0, \min\left(1, \frac{execution\_time - min\_time}{max\_time - min\_time}\right)\right)$
  $attack\_resistance\_score \leftarrow 1$ if rho_attack_result is None else 0
  $fitness \leftarrow 0.4 \cdot \log(n) + 0.2 \cdot hasse\_score \cdot \log(n) + 0.2 \cdot execution\_score + 0.2 \cdot attack\_resistance\_score$
  return fitness
end function

cumulative fitness score is returned, reflecting the candidate's elliptic curve suitability.

These are several global constants that are defined for *ai_ecc_utils.py* module, as summarized in Table IV:

TABLE IV. AI_ECC_UTILS.PY MODULE CONSTANTS

Constant	Value	Description
BITS_PRIME_SIZE	256	Size of the prime in bits. Generates a n-bit prime number
POLLARDS_RHO_TRIALS	20	Number of trials for the Pollard's Rho function
POLLARDS_RHO_MAX_ITER	10**2	Maximum iterations for each trial in the Pollard's Rho function

This module is designed to be used in conjunction with other modules that implement bio-inspired algorithms, such as PSO and GA, and given how it was designed, other AI algorithms that are capable of optimizing elliptical curves may use it in the future.

B. E-commerce Simulation Group

1) well-known_curves_params.txt: Represents parameters for standard elliptic curves used in cryptography, such as *secp256k1* and *brainpoolP256r1* [46].

2) ecc.py: This utility module includes classes and functions for elliptic curve cryptography. A key function reads ECC parameters from files like *ga_ecc_params.txt*, *pso_ecc_params.txt*, *secp256k1.txt*, or *brainpoolP256r1.txt* (the latter two as *well-known_curves_params.txt)*, creating a structured set of parameters to be used in the e-commerce simulation, as shown in Algorithm 10.

The *ecc.py* module has support functions like *ec_addition* for adding points on an elliptic curve, and *ec_scalar_multiplication* for multiplying a point by a scalar using the double-and-add method, as shown in Algorithm 11.

These utility functions facilitate key generation by creating a random private key, an integer within the range $[1, n-1]$, and producing the corresponding public key, which is computed by

Algorithm 10 Initialization of ECC Parameters

function INITIALIZE_PARAMS(option)
  Select $filename$ based on $option$:
  if option = "1" **then**
    $filename \leftarrow$ "$ga\_ecc\_params.txt$"
  else if option = "2" **then**
    $filename \leftarrow$ "$pso\_ecc\_params.txt$"
  else if option = "3" **then**
    $filename \leftarrow$ "$secp256k1.txt$"
  else if option = "4" **then**
    $filename \leftarrow$ "$brainpoolP256r1.txt$"
  else
    $filename \leftarrow$ "$secp256k1.txt$" ▷ default
  end if
  Open $filename$ for reading as f
  Initialize $params\_dict$ as an empty dictionary
  for each line in f **do**
    Split line into key, $value$ and store in $params\_dict$
    Convert $value$ to integer
  end for
  Extract parameters p, a, b, G_x, G_y, n, h from $params\_dict$
  $G \leftarrow$ ECPoint(G_x, G_y)
  return ECCParameters(p, a, b, G, n, h)
end function

Algorithm 11 Point Addition and Scalar Multiplication

function EC_ADDITION(P, Q, p)
  if P is None or inf **then return** Q
  end if
  if Q is None or inf **then return** P
  end if
  if $P.x = Q.x$ **then**
    if $P.y = -Q.y \mod p$ **then return** ECPoint(None, None) ▷ Infinity
    end if
    $m \leftarrow (3P.x^2 + a) \cdot (2P.y)^{-1} \mod p$
  else
    $m \leftarrow (Q.y - P.y) \cdot (Q.x - P.x)^{-1} \mod p$
  end if
  $x \leftarrow m^2 - P.x - Q.x \mod p$
  $y \leftarrow m(P.x - x) - P.y \mod p$ **return** ECPoint(x, y)
end function
function EC_SCALAR_MUL.(P, s, p)
  $r \leftarrow$ ECPoint(None, None)
  $c \leftarrow P$
  while s **do**
    if $s \& 1$ **then**
      $r \leftarrow$ is None ? c : ec_addition(r, c, p)
    end if
    $c \leftarrow$ ec_addition(c, c, p)
    $s >>= 1$
  end while
  Print $r.x, r.y$ **return** r
end function

multiplying the base point G by the private key, as shown in Algorithm 12.

Algorithm 12 Private and Public Key Generation

function GENERATE_PRIVATE_KEY(params)
  return randbelow(params.$n - 1$)
end function
function GENERATE_PUBLIC_KEY(private_key, params)
  $result \leftarrow$ ec_scalar_multiplication(params.G, private_key, params)
  return $result$
end function

Additionally, within the *ecc.py* module, the functions *ec_addition* and *ec_scalar_multiplication* are utilized to encrypt and decrypt messages, as shown in Algorithm 13.

Lastly, *ecc.py* uses the *hmac* Python library to generate and verify a Hash-based Message Authentication Code (HMAC) for a given message and key, ensuring the integrity and authenticity of messages.

3) Orders dataset: This e-commerce dataset includes invoices generated by an authorized online retailer [47]. These have been pre-processed and curated as orders for practical simulation purposes, converting them into order data. This data feeds *Entity A* and is then sent to *Entity B* in the e-commerce simulation.

4) EntityA.py: This component emulates an e-commerce solution, and communicates with a simulated ERP server (*EntityB.py*) via Elliptic Curve Cryptography (ECC). It includes functionalities for server connections, ECC parameters, key management, order message generation, and transaction handling. The system reads data from a spreadsheet file (Orders dataset), encrypts messages with ECC and HMAC using the module *ecc.py*, sends them to the ERP server, and employs Elliptic Curve Diffie-Hellman (ECDH) for key agreement, as shown in Algorithm 14.

5) EntityB.py: It is a simulated ERP server that interacts with the emulated e-commerce solution (*EntityA.py*) using *Flask* (Python web framework). It handles orders, key retrievals, initiates ECC and ECDH keys, and allows users to select the type of ECC parameters (such as GA, PSO, or well-known curves) to be used throughout the simulation. Functions for decrypting orders and verifying HMACs are included.

The ECC parameters employed in the simulation are selected by the user and loaded from the corresponding txt file. The *ecc.py* module handles all cryptographic operations throughout the process.

6) pollards_rho_attack.py: To evaluate the ECC parameters in the simulation, a component is designed to attack the communication between *EntityA.py* and *EntityB.py*. This Python script executes *Pollard's rho attack* [36][45] on the e-commerce simulation, employing the *"tortoise and hare"* technique to implement the attack logic. By leveraging multiprocessing for parallelization and handling collisions to determine the private key, it interacts with the ERP server (*EntityB.py*) to gather essential data such as the public key of the targeted entity. The script employs various methods like

Algorithm 13 ECC Encryption and Decryption

```
function ENCRYPT_MESSAGE(message, public_key, params)
    if not is_valid_point(public_key, params) then
        raise ValueError("Public key is not a valid point on the elliptic curve")
    end if
    k ← generate_private_key(params)
    if not is_valid_scalar(k, params) then
        raise ValueError("Invalid scalar value")
    end if
    C1 ← ec_scalar_multiplication(params.G, k, params)
    C2 ← ec_scalar_multiplication(public_key, k, params)
    message_bytes ← message.encode('utf − 8')
    encrypted_message ← []
    for byte in message_bytes do
        if C2.x is None then
            raise ValueError("Encryption failed: kQ resulted in the point at infinity")
        end if
        encrypted_byte ← byte ⊕ (C2.x&0xFF)
        encrypted_message.append(encrypted_byte)
    end for
    return C1, encrypted_message
end function
function DECRYPT_MESSAGE(C1, encrypted_message,) private_key, params)
                                        ▷ Try block starts here
    C2 ← ec_scalar_multiplication(C1, private_key, params)
                                        ▷ Catch block starts here
    print(f"An error occurred during decryption: {str(e)}")
    return None
                                        ▷ Catch block ends here
    decrypted_message_bytes ← bytearray()
    for encrypted_byte in encrypted_message do
        byte ← encrypted_byte ⊕ (C2.x&0xFF)
        decrypted_message_bytes.append(byte)
    end for
    decrypted_message ← decrypted_message_bytes.decode('utf − 8')
    return decrypted_message
                                        ▷ Try block ends here
end function
```

Algorithm 14 Main Function Procedure for Entity A

```
procedure MAIN
    Initialize ServerConnection with server URL
    Initialize ECCParams with ServerConnection
    Request ECC parameters from server
    Initialize ECCKeys with ECC parameters
    Generate ECC private and public keys
    Initialize RetailMessage with MS Excel file path
    Initialize TransactionManager with necessary objects
    Run transactions until a predetermined end time
end procedure
```

scalar multiplication and point addition on elliptic curve points to successfully carry out the attack, as shown in Algorithm 15.

It is important to note that within the *ai_ecc_utils.py* component, there is a function that leverages the logic of Pollard's Rho attack to evaluate ECC parameters. While this function shares core mathematical principles with those in *pollard_rho_attack.py*, their objectives differ significantly. Specifically, the function in the first code seeks to find collisions in points to assess the security of ECC and facilitate fitness calculation. The second code aims to find the private key. It carries additional logic to compute the private key from the collision, and employs the tortoise and hare approach, standard in Pollard's rho. Its goal is to attack the e-commerce scenario by finding the private key, a computationally challenging task.

The following image is the UML sequence diagram illustrating the interaction between *Entity A* (the emulated e-commerce component) and *Entity B* (the emulated ERP server), as shown in Fig. 2.

V. Results, Findings, and Analysis

To summarize our research findings, we divided the results into three stages. First, we ran the GA and PSO artificial intelligence algorithms. Next, we evaluated them within the e-commerce integration simulation. Finally, we compared the results between GA and PSO using the ECC Optimization Criteria detailed in earlier sections of this document.

A. Execution of AI Algorithms for ECC Optimization

Each run of the GA and PSO algorithms generates different ECC parameters. This is advantageous for implementation in e-commerce settings or any scenario requiring frequent ECC parameter changes. Despite these differences, fitness function values remain remarkably consistent across runs for both GA and PSO. Below are examples of results from each AI algorithm, as shown in Tables V and VI:

TABLE V. GA Results

Metric	Value
Attack	0
Min	0.0
Max	$3.0181340473967544 \times 10^{39}$
Avg	$3.012097779301965 \times 10^{39}$
Std	$1.3484001529034777 \times 10^{38}$
Best Individual Parameters	
Parameter a	259478422709058278976591280391547878163230073421011406267083100946720514 3790
Parameter b	401369886095925990916576587864580990140837811240502450758226668579221529 1693
Parameter p	115720213769277544236445373063821935816701441977156565361337888165594796740319
Parameter G	0, 729843922999420305306886530467207607647642966966880656658886834963804381391 49
Parameter n	115720213769277544236445373063821935816701441977156565361337888165594796740318
Parameter h	1

As observed, both GA and PSO produce 256-BIT-based parameters, advantageous for security. However, the fitness function results appear superior in GA ($3.0181340473967544 \times 10^{39}$) compared to PSO ($1.5946572521224025 \times 10^{39}$). The *"Fitness Evolution"* figures emphasize this distinction by showcasing a line graph that tracks the fitness progression over generations or iterations, highlighting the algorithm's

Algorithm 15 Pollard's Rho Attack on Elliptic Curve Cryptography

procedure P_RHO(init_value, G, public_key, params, manager_dict
    Print start message with init_value
    tortoise ← ec_scalar_multiplication(G, init_value, params)
    hare ← tortoise
    tortoise_scalar ← hare_scalar ← init_value
    for i = 1 to 2 × params.n + 1 **do**
        tortoise, tortoise_scalar ← step(tortoise, tortoise_scalar, G, public_key, params)
        hare, hare_scalar ← step(hare, hare_scalar, G, public_key, params)
        hare, hare_scalar ← step(hare, hare_scalar, G, public_key, params)
        if tortoise = hare and tortoise ≠ None **then**
            scalar_difference ← gmpy2.f_mod((tortoise_scalar - hare_scalar), params.n)
            if scalar_difference = 0 **then**
                continue
            end if
            scalar_difference_inverse ← gmpy2.invert(scalar_difference, params.n)
            secret_key ← gmpy2.f_mod((scalar_difference_inverse * hare_scalar), params.n)
            if ¬ manager_dict['found_flag'] **then**
                manager_dict['found_flag'] ← True
                Print found secret key
            end if
            break
        end if
        if manager_dict['found_flag'] **then**
            break
        end if
    end for
    Print result message
end procedure
procedure STEP(point, scalar, G, public_key, params)
    Define a function to move and update point and scalar
    Handle different cases based on x-coordinate of point
    return new point and corresponding scalar
end procedure
function GETECCPARAMSFROMSERVER
    Retrieve ECC parameters from server
end function
function GETPUBLICKEYFROMENTITYB
    Retrieve public key from Entity B
end function
procedure POLLARDSRHOATTACKONENTITYB(params)
    Carry out Pollard's rho attack on Entity B in parallel
end procedure
procedure MAIN
    Retrieve ECC parameters
    Initialize parameters
    Carry out Pollard's rho attack on Entity B
end procedure
if name = "main" **then**
    Call Main()
end if

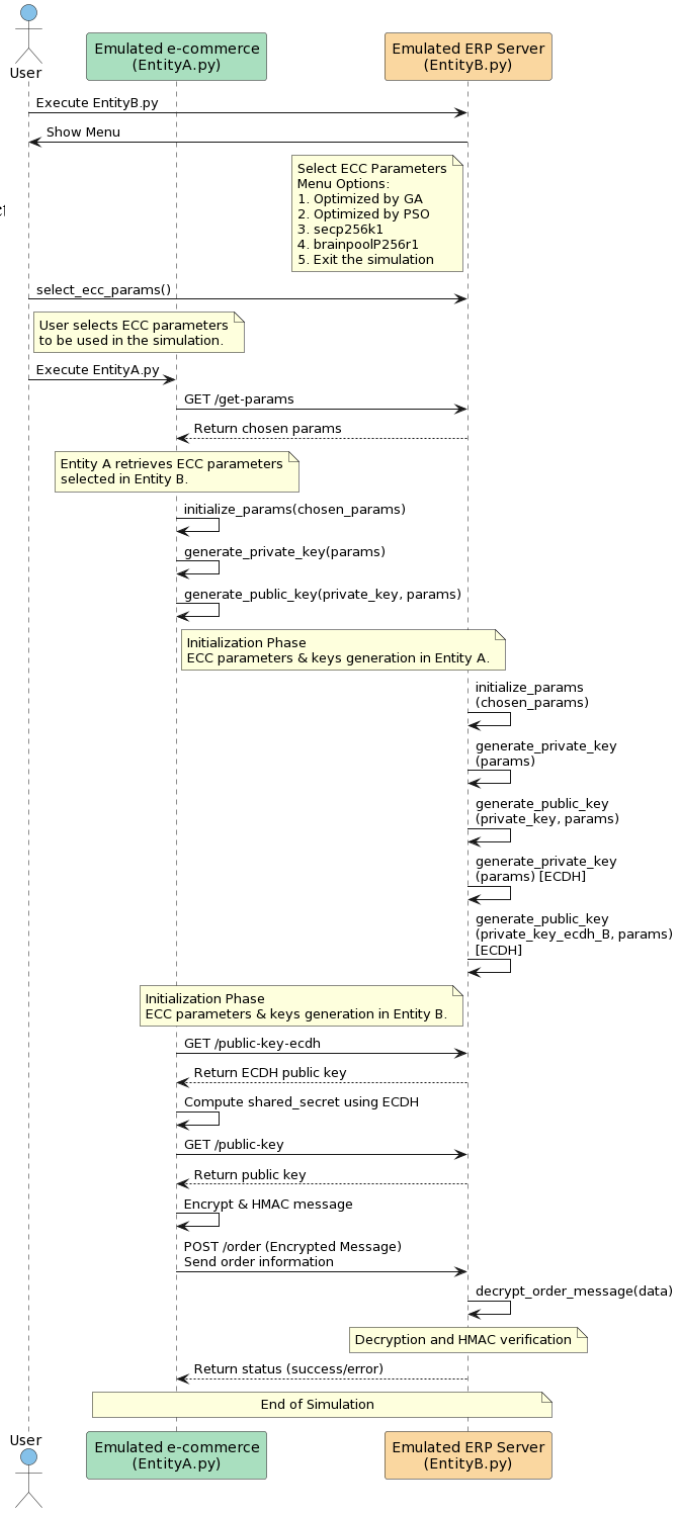

Fig. 2. Interaction between *Entity A* (e-commerce) and *Entity B* (ERP server).

convergence. In these graphs, the average fitness value for PSO seems to degrade over time, despite optimizing the Cognitive and Social parameters using grid search. Conversely, GA consistently refines its fitness function, demonstrating a more stable trend.

TABLE VI. PSO RESULTS

Metric	Value
Attack	0
Min	73.92932346203426
Max	$1.5946572521224025 \times 10^{39}$
Avg	$3.189314504244805 \times 10^{36}$
Std	$7.1243889397520655 \times 10^{37}$
Best Particle Parameters	
Parameter a	73916884511138539486074209032992425010519602193355593404983790531383100702 72
Parameter b	18466552033214128305449972063322860268226711933935550031901591713321110732 8
Parameter p	83920875675429201076002743705901489967077637562817356440692877235699677907597
Parameter G	2, 52816158108397424543262331025570826905013942926608742347720195343450586800572
Parameter n	11577418207255264997910984803085607312498477086787200556690772670901016487526 4
Parameter h	1

The subsequent charts illustrate these trends in Fig. 3 and 4:

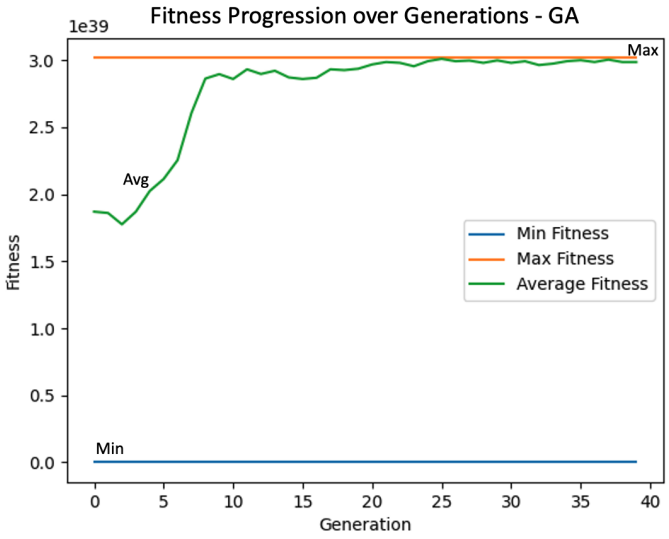

Fig. 3. Fitness progression over generations - GA.

It's worth noting that both algorithms are utilizing the same fitness function. On the other hand, GA generates results much faster than PSO, even though PSO has a feature that allows for early stopping if there's no improvement in the global best fitness for 20 iterations. While GA completes its optimization task in roughly 2 minutes, PSO takes about 22 minutes to finish its task. The tests were conducted on a 2019 15-inch MacBook Pro with the following specifications: Processor: 2.3 GHz 8-Core Intel Core i9; RAM: 16 GB 2400 MHz DDR4; Graphics: Radeon Pro 560X 4 GB and Intel UHD Graphics 630 1536 MB.

B. Execution of e-commerce Integration Simulation

In the simulation testing the transmission of order information in third-party integrations, from the simulated e-commerce to the ERP, the results are promising for both GA and PSO when compared to well-known curves like *secp256k1.txt* and *brainpoolP256r1.txt*. The speed of encrypting and decrypting messages was nearly identical among the four curve types,

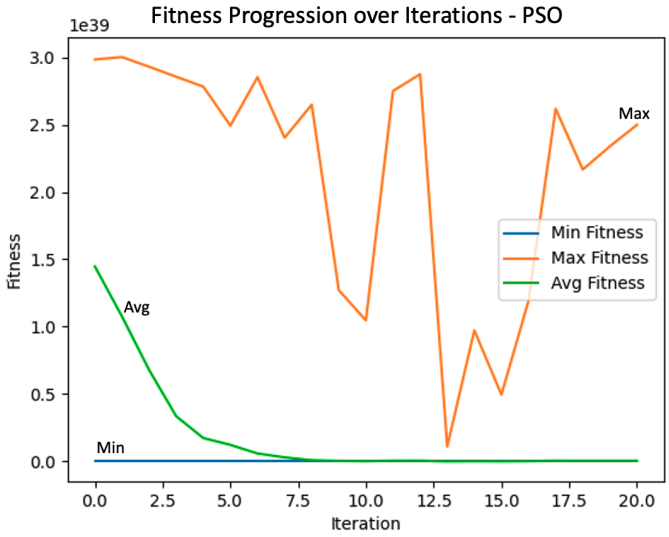

Fig. 4. Fitness progression over iterations - PSO.

ranging from fractions of a second to less than 2 seconds. Moreover, a successful Pollard's Rho attack could not be executed in any scenario, given the computationally intensive task of determining the private key, even when the simulation ran for a week. This approach emphasizes frequent changes in ECC parameters. Compared to well-known curves, GA and PSO offer an advantage by utilizing novel curves without known values for attackers.

C. Comparison based on ECC Optimization Criteria

The table below displays the winning algorithm based on the acceptance criteria previously outlined in this paper, as shown in Table VII:

TABLE VII. COMPARISON BETWEEN GA AND PSO

Efficiency		
Criteria	**Winning Algorithm**	**Justification**
Performance	GA	Faster generation of ECC parameters.
Flexibility	GA	Consistent evolution of the fitness function even when increasing the number of generations.
Robustness	GA	Greater tolerance to noise in initial constants definition.
Scalability	GA	Remains faster and more efficient even when increasing generations and parallelizing calculations.
Comparability	GA	Lower computational complexity.
Effectiveness		
Criteria	**Winning Algorithm**	**Justification**
Security	Both GA and PSO	Neither faced successful attacks in the e-commerce integration scenario. However, during parameter generation, PSO experienced more successful attacks than GA when searching for point collisions.
Optimality	GA	Consistently closer to expected maximum values and showed improvement over time.
Generalization	Both GA and PSO	Quick in both encryption and decryption processes in the simulated scenario, and neither faced successful attacks.
Validity	Both GA and PSO	ECC parameters were free from singular or anomalous curves.
Practicality	GA	Superior in performance while being compatible with real-world existing systems.

Based on the previous table and the information in this section, Genetic Algorithms (GA) outperform Particle Swarm Optimization (PSO) in ECC parameter optimization, not just for third-party e-commerce integrations but also, in terms of efficiency, across any Elliptic Curve Cryptography scenario.

VI. Future Improvements and Known Limitations

Here are some areas we believe can be improved in the future. These improvements can be considered as potential next steps or future work:

A. Advanced Parameter Tuning

The current scripts employs grid search for fine-tuning the initial constants. Employing intricate parameter tuning techniques like random search, Bayesian optimization, or metaheuristic algorithms might pinpoint superior parameter values, enhancing the efficacy of the GA and PSO algorithms.

B. Parallelization

For PSO, the fitness evaluation along with position and velocity updates for individual particles can be executed concurrently, given their independence. Incorporating parallel processing could drastically curtail computational duration, particularly with larger swarms or increased iteration counts.

C. Hybrid Algorithms

Merging PSO with alternative optimization algorithms can spawn a hybrid model that capitalizes on the strengths of each algorithm. For instance, integrating GA to evolve the swarm while utilizing PSO for refining solutions could augment solution quality and the robustness of the optimization.

D. Exploration of Alternative AI Techniques

In light of the promising results yielded by the GA and PSO algorithms in this research, it stands to reason that the exploration of alternative artificial intelligence techniques could further optimize ECC parameter generation. Leveraging the same fitness function and utility components established in this study would serve as a foundational bedrock for the integration of techniques such as deep learning, reinforcement learning, or swarm intelligence variations different from PSO, facilitating a seamless transition and a consistent basis for performance evaluation. Such endeavors could potentially unearth novel approaches that are more efficient, secure, and robust, pushing the boundaries of what can be achieved in cryptographic parameter optimization and ensuring a forward momentum in ECC security research.

E. Improved Fitness Function

The ai_ecc_utils.evaluate function, employed as the script's fitness function, ascertains the security of an elliptic curve. This function could undergo enhancement or be supplanted with an alternate fitness function to steer the PSO algorithm search more effectively. For instance, the fitness function might integrate additional security parameters or be tailored to prioritize specific elliptic curve types.

F. Integrating Diverse Cryptographic Threat Evaluations

In the future, addressing diverse threats to elliptic curves beyond just Pollard's rho attack is crucial. With historical cryptographic vulnerabilities exposed by techniques like the Pohlig-Hellman method and Baby-step Giant-step [45], and the impending rise of quantum computing introducing threats like Shor's algorithm, modern ECC methods may be at risk [35]. Integrating and evaluating these varied attacks in the fitness function will be essential for fortified defenses.

G. Quantum Computing Implications

It's crucial to consider the advent of quantum computers and their potential impact on cryptographic algorithms [35]. As quantum computing technology evolves, both GA and PSO algorithms' performance and security measures need re-evaluation to ensure they remain resilient against quantum threats. Additionally, the designed fitness function for this study, which evaluates the security and efficiency of ECC parameters, could be implemented in quantum environments since it is based on optimization through artificial intelligence, allowing for the study of its behavior in such contexts.

VII. Conclusions

In light of our comprehensive research and systematic evaluation, it is clear that Genetic Algorithms (GA) are more efficient than Particle Swarm Optimization (PSO) in the optimization of Elliptic Curve Cryptography (ECC) parameters. This assertion is based on the adept fitness function we designed and utilized in our research. GA's superiority is evident in third-party e-commerce integrations. Both algorithms are robust, successfully withstanding attacks in e-commerce integration tests. Nevertheless, GA consistently delivers quicker and more reliable performance, integrating seamlessly with real-world systems. This benefit, along with its efficient evolution and rapid ECC parameter generation, underscores GA's dominance in this field. While PSO does offer distinct advantages, its potential is hampered by its relative inefficiency, especially regarding computational speed. Consequently, we strongly advise stakeholders aiming to optimize ECC parameters to prioritize the GA approach.

References

[1] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203-209, Jan 1987.

[2] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *LNCS, Advances in Cryptology - CRYPTO '85: Proceedings*, ed: Springer Berlin / Heidelberg, 1986, p. 417.

[3] L. C. Washington, *Elliptic Curves: Number Theory and Cryptography*, 2nd ed, 2008.

[4] Lenstra, Arjen & Verheul, Eric, "Selecting Cryptographic Key Sizes," *Journal of Cryptology*, vol. 14, pp. 255-293, 2001. https://doi.org/10.1007/s00145-001-0009-4.

[5] I. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*, LMS Lecture Notes 265, Cambridge University Press, 1999. https://doi.org/10.1017/CBO9781107360211.

[6] D. Hankerson, A. Menezes, S. Vanstone, "Elliptic curve arithmetic," in *Guide to Elliptic Curve Cryptography*, Springer, New York, 2004. https://doi.org/10.1007/0-387-21846-7_3.

[7] Z. Liu, H. Seo, J. Großschädl, and H. Kim, "Efficient Implementation of NIST-Compliant Elliptic Curve Cryptography for 8-bit AVR-Based Sensor Nodes," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 7, pp. 1385-1397, July 2016. https://doi.org/10.1109/TIFS.2015.2491261.

[8] M. Lochter, J. Merkle, J. Schmidt, and T. Schütze, "Requirements for Elliptic Curves for High-Assurance Applications," 2015.

[9] S. R. Singh, A. K. Khan, and T. S. Singh, "On the performance of Elliptic Curve public cryptosystem," 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT), Pune, India, 2016, pp. 24-29. https://doi.org/10.1109/ICACDOT.2016.7877545.

[10] D. Kobrle and R. Lorencz, "Optimization of elliptic curve operations for ECM using double & add algorithm," 2015 Forth International Conference on e-Technologies and Networks for Development (ICeND), Lodz, Poland, 2015, pp. 1-4. https://doi.org/10.1109/ICeND.2015.7328534.

[11] H. Lv, H. Li, J. Yi, and H. Lu, "Optimal implementation of elliptic curve cryptography," Proceedings of 2013 IEEE International Conference on Service Operations and Logistics, and Informatics, Dongguan, China, 2013, pp. 35-39. https://doi.org/10.1109/SOLI.2013.6611377.

[12] T. Ribaric and S. Houghten, "Genetic programming for improved cryptanalysis of elliptic curve cryptosystems," 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 2017, pp. 419-426. https://doi.org/10.1109/CEC.2017.7969342.

[13] S. Selvi, M. Gobi, M. Kanchana, and S. Mary, "Hyper elliptic curve cryptography in multi cloud-security using DNA (genetic) techniques," 2017, pp. 934-939. https://doi.org/10.1109/ICCMC.2017.8282604.

[14] P. Sethuraman, P. S. Tamizharasan, and K. Arputharaj, "Fuzzy Genetic Elliptic Curve Diffie Hellman Algorithm for Secured Communication in Networks," *Wireless Pers Commun*, vol. 105, pp. 993–1007, 2019. https://doi.org/10.1007/s11277-019-06132-4.

[15] N. Chandnani and C. N. Khairnar, "A Novel Secure Data Aggregation in IoT using Particle Swarm Optimization Algorithm," 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), Bhopal, India, 2018, pp. 1-6. https://doi.org/10.1109/ICACAT.2018.8933784.

[16] Mullai, A. & Mani, K., "Enhancing the security in RSA and elliptic curve cryptography based on addition chain using simplified Swarm Optimization and Particle Swarm Optimization for mobile devices," *International Journal of Information Technology*, vol. 13, 2020. https://doi.org/10.1007/s41870-019-00413-8.

[17] Kota, S., Padmanabhuni, V.N., Budda, K. *et al.*, "Authentication and Encryption Using Modified Elliptic Curve Cryptography with Particle Swarm Optimization and Cuckoo Search Algorithm," *J. Inst. Eng. India Ser. B*, vol. 99, pp. 343–351, 2018. https://doi.org/10.1007/s40031-018-0324-x.

[18] Holland, J. H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, University of Michigan Press, 1975.

[19] Eberhart, R., & Kennedy, J., "A new optimizer using particle swarm theory," in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39-43, Nagoya, Japan: IEEE, 1995.

[20] Silambarasan, S., & Savitha Devi, M., "Hybrid Simulated Annealing with Lion Swarm Optimization Algorithm with Modified Elliptic Curve Cryptography for Secured Data Transmission Over Wireless Sensor Networks (WSN)," 2022.

[21] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 1983. https://doi.org/10.1126/science.220.4598.671.

[22] M. Wang, G. Dai, H. Hu and L. Pen, "Selection of Security Elliptic Curve Based on Evolution Algorithm," 2009 International Conference on Computational Intelligence and Natural Computing, Wuhan, China, 2009, pp. 55-57. https://doi.org/10.1109/CINC.2009.205.

[23] X. Zhou, "Elliptic Curves Cryptosystem Based Electronic Cash Scheme with Parameter Optimization," 2009 Pacific-Asia Conference on Knowledge Engineering and Software Engineering, Shenzhen, China, 2009, pp. 182-185. https://doi.org/10.1109/KESE.2009.55.

[24] G. Vostrov and I. Dermenzhy, "The Concept of Machine Learning and Elliptic Curves United Approach in Solving of the Factorization Problem," 2019 XIth International Scientific and Practical Conference on Electronics and Information Technologies (ELIT), Lviv, Ukraine, 2019, pp. 87-91. https://doi.org/10.1109/ELIT.2019.8892318.

[25] Laue, R., Huss, S.A., "Parallel Memory Architecture for Elliptic Curve Cryptography over GF(p) Aimed at Efficient FPGA Implementation," *J Sign Process Syst Sign Image*, vol. 51, pp. 39–55, 2008. https://doi.org/10.1007/s11265-007-0135-9.

[26] N. Jayapandian, "Cloud Dynamic Scheduling for Multimedia Data Encryption Using Tabu Search Algorithm," *Wirel. Pers. Commun.*, vol. 120, no. 3, pp. 2427–2447, Oct 2021. https://doi.org/10.1007/s11277-021-08562-5.

[27] Bin, P., Tao, Z., & Yu, W., "The Integration Strategy of E-commerce Platform and ERP Based on Cooperative Application," 2010 International Conference on E-Business and E-Government, IEEE, 2010.

[28] Krithika, L. B., Prabadevi, B., Deepa, N., & Bhavanasi, S., "Integration of E-Commerce System with Various ERP Tools," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), IEEE, 2020.

[29] Kuno, H., Lemon, M., & Karp, A., "Transformational interactions for P2P e-commerce," Proceedings of the 35th Annual Hawaii International Conference on System Sciences, IEEE, 2002.

[30] H. Wang and Q. Hu, "Research and Application of an Integration Platform for E-Commerce System Based on SOA," 2009 *International Conference on Management of e-Commerce and e-Government*, IEEE, 2009.

[31] J. Daigler, S. Shen, P. Gillespie, M. Lowndes, A. Vasudevan, and Y. Dharmasthira, "Gartner Magic Quadrant for Digital Commerce," Aug. 2022. [Online]. Available: https://www.gartner.com/en/documents/4017524 and https://greywolf.com/wp-content/uploads/2022/09/Magic-Quadrant-for-Digital-Commerce.pdf

[32] A. Fujii, M. Nakayama, K. Tanaka, and K. Nagamura, "EDI support system over RESTful Web API," *IEEE 8th International Symposium on Intelligent Systems and Informatics*, IEEE, 2010.

[33] N. Kulkarni, S. Kumar, K. Mani, and S. Padmanabhuni, "Web services: e-commerce partner integration," *IT Professional*, vol. 7, no. 2, IEEE, 2005.

[34] G. Shen and X. Zheng, "Research on Implementation of Elliptic Curve Cryptosystem in E-Commerce," *Proceedings of the 2008 International Symposium on Electronic Commerce and Security*, IEEE, 2008.

[35] H. Zhang, Z. Ji, H. Wang, and W. Wu, "Survey on quantum information security," *China Communications*, vol. 16, no. 10, Magazine Article, IEEE, 2019.

[36] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, pp. 243–264, 1987.

[37] C. K. Koc, "Analysis of sliding window techniques for exponentiation," *Computers and Mathematics with Applications*, vol. 30, no. 10, pp. 17–24, Nov. 1995.

[38] R. Tabbussum and A. Q. Dar, "Performance evaluation of artificial intelligence paradigms—artificial neural networks, fuzzy logic, and adaptive neuro-fuzzy inference system for flood prediction," *Environ Sci Pollut Res*, vol. 28, pp. 25265–25282, 2021. [Online]. Available: https://doi.org/10.1007/s11356-021-12410-1

[39] Zhang, T., Xiao, W., & Hu, P. "Design of Online Learning Early Warning Model Based on Artificial Intelligence". *Wireless Communications and Mobile Computing*, 2022(1), 1-11. Hindawi. https://doi.org/10.1155/2022/3973665

[40] A. He et al., "A Survey of Artificial Intelligence for Cognitive Radios," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 4, pp. 1578-1592, May 2010, doi: 10.1109/TVT.2010.2043968.

[41] R. Hamon, H. Junklewitz, and I. Sanchez, "Robustness and explainability of Artificial Intelligence," *Publ. Off. Eur. Union*, Luxembourg, 2020.

[42] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798-1828, Aug. 2013, doi: 10.1109/TPAMI.2013.50.

[43] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[44] T. Icart, "How to Hash into Elliptic Curves," in *Advances in Cryptology - CRYPTO 2009*, S. Halevi, Ed. Lecture Notes in Computer Science, vol 5677, Springer, Berlin, Heidelberg, 2009, https://doi.org/10.1007/978-3-642-03356-8_18.

[45] S. Ullah, J. Zheng, N. Din, M. T. Hussain, F. Ullah, and M. Yousaf, "Elliptic Curve Cryptography; Applications, challenges, recent advances, and future trends: A comprehensive survey," *Computer Science Review*, vol. 47, 100530, 2023, https://doi.org/10.1016/j.cosrev.2022.100530.

[46] M. Kramer, F. Gerstmayer, and J. Hausladen, "Evaluation of Libraries and Typical Embedded Systems for ECDSA Signature Verification for Car2X Communication," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin, Italy, 2018, pp. 1123-1126, doi: 10.1109/ETFA.2018.8502595.

[47] "Online Retail," *UCI Machine Learning Repository*, 2015, https://doi.org/10.24432/C5BW33.