Received July 23, 2021, accepted August 21, 2021, date of publication August 31, 2021, date of current version September 9, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3109235

Analysis of Swarm Intelligence Based ANN Algorithms for Attacking PUFs

AHMED OUN, (Graduate Student Member, IEEE), NOOR AHMAD HAZARI, AND MOHAMMED Y. NIAMAT, (Life Member, IEEE)
Department of Electrical Engineering and Computer Science, The University of Toledo, Toledo, OH 43606, USA
Corresponding author: Ahmed Oun (ahmed.oun@rockets.utoledo.edu)

ABSTRACT Physical Unclonable Functions (PUFs) are used for authentication and generation of secure cryptographic keys. However, recent research work has shown that PUFs, in general, are vulnerable to machine learning modeling attacks. From a subset of Challenge-Response Pairs (CRPs), the remaining CRPs can be effectively predicted using different machine learning algorithms. In this work, Artificial Neural Networks (ANNs) using swarm intelligence-based modeling attacks are used against different silicon-based PUFs to test their resiliency to these attacks. Amongst the swarm intelligence algorithms, the Gravitational Search Algorithm (GSA), Cuckoo Search Algorithm (CS), Particle Swarm Optimizer (PSO) and the Grey Wolf Optimizer (GWO) are used. The attacks are extensively performed on six different types of PUFs; namely, Configurable Ring Oscillator, Inverter Ring Oscillator, XOR-Inverter Ring Oscillator, Arbiter, Modified XOR-Inverter Ring Oscillator, and Hybrid Delay Based PUF. From the results, it can be concluded that the first four PUFs under study are vulnerable to ANN swarm intelligence-based models, and their responses can be predicted with an average accuracy of 71.1% to 88.3% for the different models. However, for the Hybrid Delay Based PUF and the Modified XOR-Inverter Ring Oscillator PUF, which are especially designed to thwart machine learning attacks, the prediction accuracy is much lower and in the range of 9.8% to 14.5%.

INDEX TERMS Hardware security, FPGA, PUF, artificial neural network, swarm intelligence, GSA, CS, PSO, GWO, machine learning attacks.

I. INTRODUCTION

In recent years, the use of programmable devices such as Field Programmable Gate Arrays (FPGAs) and custom designed Application Specific Integrated Circuits (ASICs) have increased rapidly. The increased deployment of these devices in mission critical computing systems include, but are not limited to, communication networks, smart grids, defense equipment, and internet of things, has led hackers to continually devise new techniques to breach the security of these devices. Examples of such attacks include disabling or degrading the function of these chips in systems like radars and missiles. Other attempts include implanting malicious electronic circuitry in the chips, known as Trojans, to steal vital information for cyber-attacks. These tampered chips can subsequently act as 'spy chips' by collecting confidential data for adversaries and hackers. To counter such attacks, chip designers have embedded additional layers of security in these devices [1], [2]. Although researchers have long tried to secure hardware-based systems with both software and hardware-based approaches, this paper explicitly focuses on techniques based on hardware-oriented security and trust [3], [4]. These approaches mainly involve generation of unique hardware-based cryptographic keys in the form of Challenge-Response Pairs (CRPs). In order to generate hardware-based unique keys, different structures of physical unclonable functions (PUFs) have been proposed in the past [5], [6]. Essentially, a PUF utilizes manufacturing process variation, which is an inherent property of silicon chips, to generate unique and unclonable CRPs. Amongst the different types of PUFs available, the delay-based PUFs are widely studied in CMOS-based silicon devices. The most investigated PUFs on silicon-based devices are the Ring Oscillator PUF (ROPUF) and the Arbiter PUF (APUF). Most of the delay-based PUFs are strong candidates for not only ASICs but also for FPGAs [7], [8]. The significant advantage of using PUFs as security measures is that it does not require on-chip memory to generate and store keys; thus,

The associate editor coordinating the review of this manuscript and approving it for publication was Junaid Arshad.

it eliminates the use of on-chip memory for the security of the hardware-based system. Another very significant feature of the PUF is that the keys generated by the PUF are device specific. Further, the keys change with the specific location and placement of the PUF inside the chip, since they depend on the random manufacturing process variations [9], [10]. It should be noted that the behavior of PUFs rely on the random manufacturing process variations related to several components that are used to construct it. These components are sometimes linearly interrelated to the number of CRPs. Because of these limitations and linearity, an attacker may try all challenges and know the corresponding responses within an extended period of time [6]. This kind of brute force approach, however, generally fails because of the time required and because of the fact that the exact location of the PUF is unknown. It is further complicated in FPGAs, since the location of a PUF mapped onto an FPGA, unlike an ASIC, can be frequently changed by the designer by changing the bit-stream file.

PUF produces a device-specific unique response for a given challenge. This property of the PUF makes it suitable for different applications including, authentication, IP protection, random key generation, remote attestation, and secured supply chain, etc. Once the CRPs can be predicted by an attacker; as a consequence, the whole concept of cryptographic primitive for hardware security applications, including PUF as an authenticator, is in jeopardy. Though PUFs are considered unclonable, researchers have shown that they are vulnerable to machine learning-based modeling attacks. An attacker can perform different types of attacks, including side-channel attacks, cloning, reverse engineering, Probably Approximately Correct (PAC) based attacks, and eavesdropping for predicting the CRPs [11]–[15]. Side-channel based attacks can be performed by monitoring the voltage, current, and power values during runtime. If an attacker wants to authenticate using PUF CRPs without getting any access to the PUF, the attacker would be able to do so if the attacker has the responses available for the challenges, which can be done by eavesdropping on some of the CRPs. Hackers can eavesdrop by using MITM attacks by recording the network data packets and extracting the information of the CRPs when the system is in operation. Thus, after acquiring a set of CRPs, a PUF can be modeled using machine learning. Side-channel based attacks can be performed by monitoring the voltage, current, and power during runtime. PUFs have been successfully attacked using machine learning algorithms such as Logistic Regression, Probably Approximately Correct learning, Evolutionary Strategy, Quick Sorting, etc. [13]–[15]. In Rührmair *et al.*'s research [13], the authors used quick sorting for modeling RO PUFs. In J. Delvaux's work [14], the authors performed modeling attacks on APUF, PolyPUF, OB-PUF, RPUF, LHS-PUF, and PUF FSM protocols. The Probably Approximately Correct (PAC) learning algorithm has been used for predicting ROPUF CRPs in [15]. In their work, the number of CRPs required to learn the models is on a scale of ten thousand which is high for an attacker to obtain from the CRP set. Fault injection-based modeling attacks on APUFs are performed in [16]. In this attack model, an attacker must have physical access to the PUF. Logical Approximation and Global Approximation attacks are performed on different structures of Arbiter PUFs using ANN methods of RMSProp and Gradient Descent Optimizer [17]. In this technique, the number of CRPs required is also high. Different side channel-based modeling attacks have been performed in [18] and [19], which also requires physical access to the PUF device. Khalafalla and Gebotys [20] performed deep learning attacks on a Double Arbiter PUF. In their work, the authors performed Logistic Regression-based deep learning attacks. The number of CRPs required to perform such an attack is very high and requires more than a million pairs of CRPs that are difficult for an adversary to obtain in order to attack the PUF.

Genetic Algorithms have also been used to predict CRPs for the ROPUF [21], [22]. In the Genetic Algorithm-based modeling, CRPs are generated by crossover, mutation, and then the attacks are performed, which is not consistent for different models of ROPUFs. Mathematical modeling of different PUFs including the Arbiter PUF and the Ring Oscillator PUF has been performed in [23]. In this work, the authors describe a mathematical model for the ROPUF and perform Logistic Regression-based modeling attacks on the Arbiter PUF and the DCMUX PUF. In this approach, the drawback is that the CRPs depend on the different structures of the ROPUF. ANN-based modeling attacks on a small set of CRPs using different optimizations including RMSprop, Adam, Nadam, etc., have been performed in [24]. However, the prediction accuracy needs improvement.

Different metaheuristics algorithms exist in the literature to solve optimization problems. Metaheuristics algorithms can be classified into different categories including, Evolutionary, Physics-based, and Swarm Intelligence-based Algorithms. The Genetic Algorithm (GA) is the most popular Evolutionary based algorithm proposed by [25], which works on an initial random solution and optimizes the solution based on generations and mutations. Other popular Evolutionary based algorithms are Genetic Programming (GP) [26], Evolutionary Strategies (ES) [27], Differential Evolution (DE) [28] etc. The popular physics-based algorithms is the Gravitational Search Algorithm (GSA) [29] which works based on the law of gravity, and the best solution is reached after the iteration can produce specific agents that achieve certain fitness. Ultimately, the heavier; the mass is, the closer the optimum points will be. Other physics-based algorithms include Big-Bang Big-Crunch (BBBC) [30], Central Force Optimization (CFO) [31], Galaxy-based Search Algorithm (GbSA) [32], Gravitational Local Search (GLSA) [33], Charged System Search (CSS) [34] etc. Swarm Intelligence (SI) based algorithms are a subset of the bio-inspired algorithms. SI is a nature-inspired algorithm produced by a group of animals or birds acting together, and the algorithm is based on how these animals act or behave to adapt to the different scenarios occurring in their surroundings [35]. In Particle Swarm Optimization (PSO) the particles chase the position of the best particle and reach their own best position so that the overall best solution of the swarm is obtained [36]. Other popular swarm intelligence-based algorithm includes Ant Colony Optimization (ACO) [37], Cuckoo Search (CS) [38], Grey Wolf Optimizer (GWO) [39], etc., which are inspired by hunting and searching behavior.

In 2014, Mirjalili *et al.* introduced the Grey Wolf Optimizer (GWO), which is a metaheuristic algorithm that simulates the hierarchical superiority-based hunting mechanism of Grey wolves for hunting down prey. This arrangement benefits them to preserve stability and support each other throughout hunting. Wolves have a strict social hierarchy consisting of the alpha (α), beta (β), delta (δ), and omega (ω) wolves [39]. The GWO algorithm takes these features of Grey Wolf to search optimized solution of a problem utilizing exploitation and exploration; therefore, in the searching process, the best solution position can be comprehensively estimated by three solutions. Thus, the algorithm can significantly decrease the probability of falling into the local optimum. The properties of metaheuristics algorithms have motivated their use to solve different engineering problems such as embedded systems, electric power system [40], scheduling Energy Storage Unit problems [41], communication network and Distributed Compressed Sensing (DCS) problem [42]. Hence, the research on the swarm intelligence optimization algorithms has an academic advantage and practical importance.

In our earlier work [43], we presented an analytical study of the vulnerability of the Configurable Ring Oscillator PUF and the XOR-Inverter Ring Oscillator PUF against Feed-Forward Neural Network (FNN) attacks using the Dragonfly Algorithm. That limited study showed that both designs are vulnerable to this type of attack. In this paper, Artificial Neural Networks are trained using different swarm intelligence algorithms, namely: GSA, CS, PSO and GWO to study the vulnerability and resistance of various PUF structures against machine learning modeling attacks. It is assumed that an adversary is able to get hold of a subset of the CRPs and then attempts to predict the remaining set of CRPs by performing modeling attacks.

The contributions of this paper are listed as follows:

- Use of Artificial Neural Network based modeling attacks on various PUFs using different Swarm Intelligence algorithms, namely: The Gravitational Search Algorithm (GSA), Cuckoo Search Algorithm (CS), Particle Swarm Optimization and the Grey Wolf Optimizer. To the best of our knowledge, these algorithms have not been used in studying the vulnerability of PUFs to ANN-based attacks.
- Development of a comparative study and statistical analysis for the different Swarm Intelligence optimization attack models' results with respect to other machine learning attack models. It is found that the ANN-based Grey Wolf Optimizer approach produces better accuracy results than the other methods.

The rest of the paper is organized as follows: Section II describes current research related to PUFs, and Section III describes the structure of the Artificial Neural Network. Section IV presents an introduction to the Gravitational Search Algorithm (GSA), Cuckoo Search Algorithm (CS), Particle Swarm Optimization and the Grey Wolf Optimizer algorithm. Section V describes the proposed method and approach. In Section VI, experimental and simulation results are discussed. Section VII provides concluding remarks.

II. RESEARCH BACKGROUND: DIFFERENT PUF STRUCTURES

A. BASIC SILICON PUFs

Ring Oscillator Physical Unclonable Functions (ROPUFs) and Arbiter Physical Unclonable Functions (APUFs) are the two most commonly used silicon-based PUFs [44], [45]. The basic ROPUF design is described first. Fig. 1 shows the structure of the Ring Oscillator PUF [46]. The design relies on delay loops, which can be produced using an odd number of inverters. As can be seen from the figure, the output bit is generated by the random selection of a pair of ring oscillators. Because of the process manufacturing variations inherent in the chip, ROs that are mapped at different locations of the chip produce different frequencies (f_a *and* f_b). These two frequencies (f_a *and* f_b) are compared. If the frequency of the first RO is greater than the second, then the output is 1; otherwise it is 0.

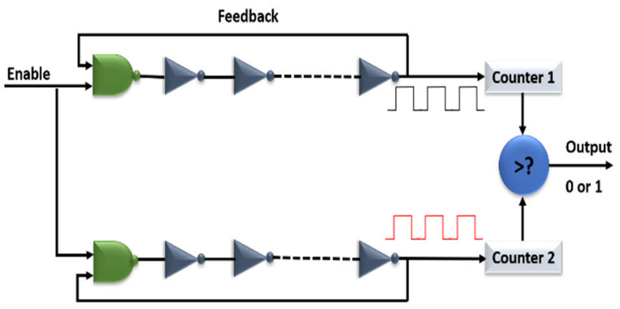

FIGURE 1. Ring oscillator PUF circuit.

A response bit (r_{ab}) is thus produced by a simple comparison as shown in equation (1):

$$r_{ab} = \begin{cases} 1, & \text{if } f_a > f_b, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The basic structure of the Arbiter PUF is shown in Fig. 2. The circuit produces a race among two delay paths with an arbiter at the end [47]. In APUF, a rising edge signal travels through two paths simultaneously. Due to process variations, the signal on one path travels faster than the other and generates a 1 or a 0 response. The challenge bits consist of K external bits ($C_1 = b_1.b_2 \ldots . b_k$) for K number of stages. Thus, for challenge bits C_1, C_2, C_n, a response of $R_1, R_2 \ldots R_n$ is obtained.

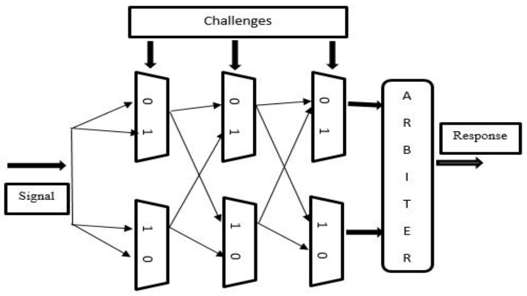

FIGURE 2. Arbiter PUF.

B. CONFIGURABLE RING OSCILLATOR PUF

The Configurable ROPUF design shown in Fig. 3 was presented in our earlier work [9]. This c-ROPUF design was implemented on a Spartan 3E FPGA board, which was divided into eight regions. In each region, sixteen ring oscillators were placed in forty configurable logic blocks. The oscillators can be selected based on the challenges provided to the programmable XOR gates. The responses were collected using the Agilent 16801A logic analyzer. The advantage of this design is that it can generate a large number of CRPs from a small chip area.

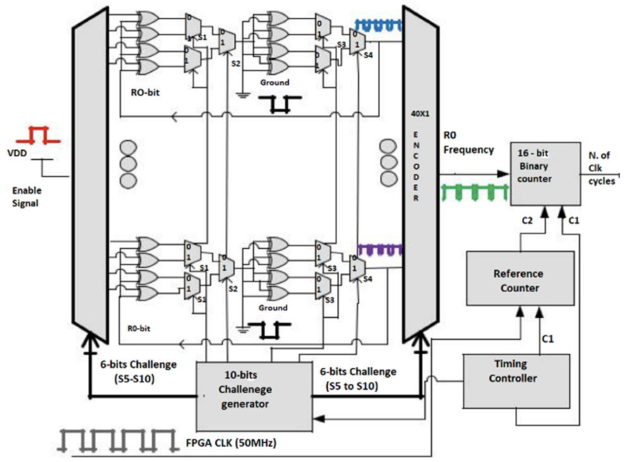

FIGURE 3. Configurable ROPUF design.

C. INVERTER RING OSCILLATOR PUF

A 5-stage Inverter Ring Oscillator PUF, as shown in Fig. 4, is used in this study. The PUFs are mapped on Five different Spartan 3E Xilinx boards. Each PUF consists of 512 Ring Oscillators.

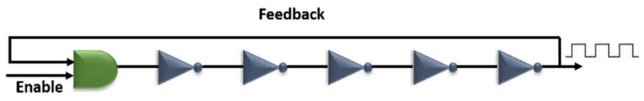

FIGURE 4. Five stage NOT based Ring Oscillator.

D. XOR-INVERTER ROPUF

The XOR-Inverter based Ring Oscillator PUF is shown in Fig. 5. This design consists of NAND, XOR, and Inverter

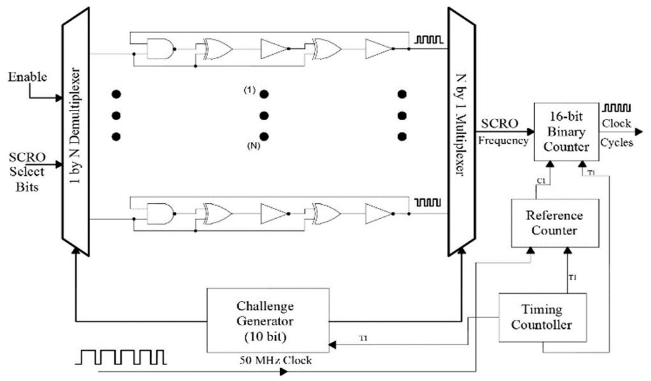

FIGURE 5. XOR-Inverter based ROPUF design.

gates and has been implemented on ten different Xilinx FPGA boards in our research lab [24]. The design has been implemented using hard macros so that the oscillator provides fixed routing, and the frequencies are not affected by routing delays. The ROs are enabled for a certain period of time to generate a response for a fixed challenge. For different challenges applied through the challenge generator, the frequencies at the output are collected through the frequency counter. Each challenge generates a single bit of response by comparing frequencies between the two oscillators.

E. MODIFIED XOR-INVERTER ROPUF

This design, shown in Fig. 6, is a modification of the XOR-Inverter ROPUF introduced in our earlier work for thwarting machine learning modeling attacks [24]. As shown in the figure, the new challenges are generated from the challenge generator which consists of an XOR and a Linear Feedback Shift Register (LFSR) network. The design has been modified in a way that the ring oscillators are selected in a pair with the same routing. The difference between the two oscillator frequencies should lie within a specific threshold frequency to avoid bit-flips. If the oscillators do not meet this criterion, they are moved to another CLB slice.

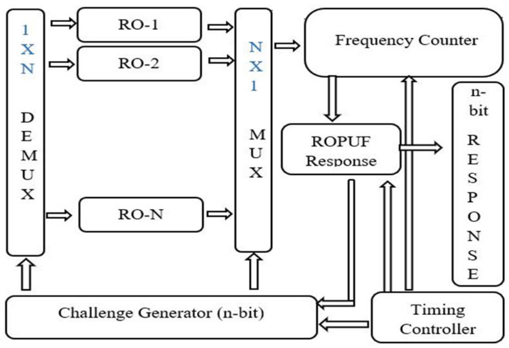

FIGURE 6. Modified design of XOR-Inverter based ROPUF.

F. HYBRID DELAY BASED PUF

The Hybrid Delay based AROPUF (Arbiter-Ring Oscillator PUF), shown in Fig. 7, was proposed in our earlier work [48].

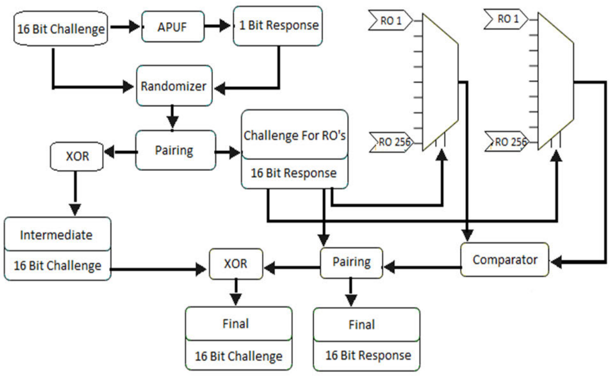

FIGURE 7. Hybrid Delay based AROPUF design.

The hybrid model was also designed to prevent machine learning based modeling attacks. This design is a combination of the Arbiter PUF and the Ring Oscillator PUF. A one-bit response is generated by providing a n-bit challenge to the APUF. The CRPs are randomized using the Mersenne Twister Random Number Generator [49]. The randomized CRPs are paired sequentially to form n-bit responses. The final output of the architecture is an n-bit response corresponding to a n-bit challenge.

III. ARTIFICIAL NEURAL NETWORK

An Artificial Neural Network (ANN) is a network structure of connected artificial neurons that can model complex relationships between inputs and outputs using computational and statistical data modeling tools. The neural networks consist of different layers termed as the input layer, output layer, and hidden layer. The first layer from where the network takes the input is known as an input layer, whereas the last layer of the network is termed as an output layer. The layers in between are termed as hidden layers. The number of hidden layers varies depending on the design [50]. The structure of the neural network is shown in Fig. 8.

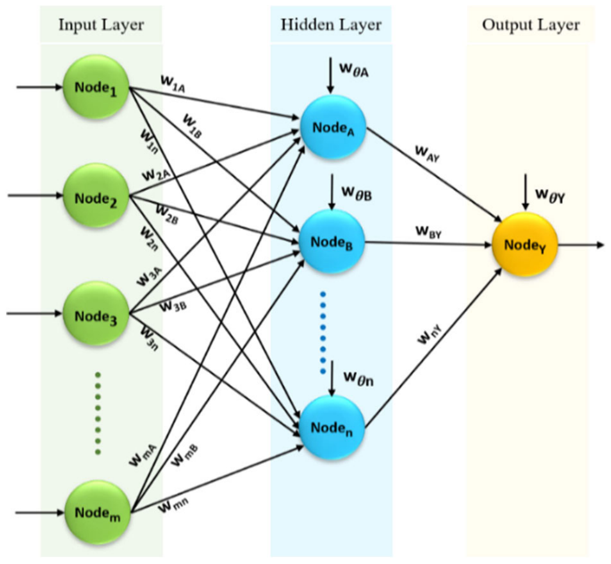

FIGURE 8. Artificial neural network structure.

The input layer is connected and assigned a weight to the hidden layers. Similarly, the hidden layer is connected to the output layers; consequently, the output of any input layer act as an input of the next layer. For each node, weights are assigned and adjusted based on the input-output relationship. The output of a 3-layer feed-forward neural network can be given by:

$$Y_j = b_j + \sum_{i=1}^{3} w_{i,j} x_i \tag{2}$$

where, Y_j is the output, b_j is base, $w_{i,j}$ is the weights, and x_i is the input. The input of a hidden layer is modified by some nonlinear function, sigmoid, which is the activation function:

$$Sigmoid\ (z) = \frac{1}{1 + e^{-z}} \tag{3}$$

The weights are updated and calculated according to the difference obtained from model output and the actual output of the training data [51]. This difference is calculated using loss or cost function, which is minimized until the training loss is minimum or goes very close to zero.

IV. SWARM INTELLIGENCE

Swarm Intelligence (SI) is a cooperative system based on a group of agents that achieve a common goal by cooperating according to their behavior and system organization. The fundamental concept behind swarm intelligence techniques is the replication of the behavior of the natural collective system [52]. Amongst the many available swarm intelligence algorithms, the Gravitational Search Algorithm (GSA), Cuckoo Search Algorithm (CS), Particle Swarm Optimization (PSO), and the Grey Wolf Optimizer (GWO) algorithms are frequently used. These algorithms, in general, are simple and computationally efficient. Specifically, these algorithms have higher viability, robustness, stability, and search efficiency, and have a fast convergence rate [53]. Moreover, these algorithms are able to optimize a vast search space with a fixed size population to solve different complex design optimization problems. For a detailed comparative analysis of the various swarm intelligence algorithms, the reader is again referred to [53].

A. GRAVITATIONAL SEARCH ALGORITHM

Gravitational Search Algorithm (GSA) is a swarm optimization technique proposed by Rashedi *et al.*, based on gravity concepts and different masses' interaction [29]. In this algorithm, the solutions of different agents' populations interact with one another via the theory of Newtonian gravity force and the laws of motion. The solution's performance is measured by different masses. Due to gravitational force, the masses are dragged towards each other, which creates a global movement of all objects approaching the objects with greater masses. The exploration step occurs when a mass moves towards a heavier mass, and the exploitation is when heavier masses move slowly. Accordingly, each mass can convey information with different masses and see their situation through the gravitational force. The mass's position compares to a problem's solution; then, the best solution is achieved with the heavier agent. The initial population is generated randomly, and the position of the agents are defined as:

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n) \quad \text{for } i = 1, 2, \ldots, \mathrm{N} \tag{4}$$

where x_i^d presents the position of i^{th} agent in the d^{th} dimension. The gravitational search algorithm sets the initial value of the constant G:

$$G(t) = G_0 e^{-\propto t/T} \tag{5}$$

where G_0 and $\propto$ is initialized at the beginning of the iteration and T is the total number of iterations. The agents update the velocity and the position according to these equations:

$$v_i(t+1) = rand_i \times v_i(t) + a_i(t) \tag{6}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{7}$$

where, $rand_i$ is a uniform random variable in the interval [0, 1]. This random number is used to give a randomized characteristic to the search.

The total force acting on agent i at iteration t, was calculated as follows:

$$F_i^d(t) = \sum_{j \in K_{best}, j \neq i} rand_j F_{ij}^d(t) \tag{8}$$

where K_{best} represents the set of k agents with best fitness and biggest mass.

The pseudo code for the GSA is shown below [29]:

Algorithm 1 Gravitational Search Algorithm

```
1.  Objective function $f(x)$, $x = (x_1, x_2, \ldots x_d,)^T$
2.  Initialize the population of $n$ agents $x_i$
3.  while $t <$ Max of Iterations do
4.      Evaluate the fitness for each agent
5.      for each searching
6.          Update the $G(t)$, $best(t)$, $worst(t)$ and $M_i(t)$
7.      end for
8.      Calculation of the total force in different directions.
9.      Calculation of acceleration and velocity.
10.     Updating agents' position.
11.     $t = t + 1$
12. end while
13. Return the best solution
```

B. CUCKOO SEARCH ALGORITHM

Cuckoo search algorithm (CS) is a nature-inspired optimization algorithm proposed by Yang in 2009 to solve optimization problems based on the cuckoo bird's breeding behavior and search approach of laying its eggs in the best host nest [38]. The CS algorithm is based on the brood parasitism of cuckoo birds. The species lay their eggs in other host bird nests to be brooded by the proxy mother bird and use the host bird assistance to hatch their eggs. The hatching probability of similar eggs to the host bird's eggs is high. In some cases, the other bird recognizes the different eggs, so they throw the eggs away, destroy them, and even leave their nests to build another one in a distinct location. The CS algorithm uses better solutions to substitute not-so-good solutions in the nests. As a result, it can enhance search capabilities to improve the relationship between exploration and exploitation. The CS algorithm is performed through the following three rules: first: each cuckoo bird chooses a random nest to lay only one egg; second, the best nests with a good quality of eggs will carry over for the next population; third, a host bird can detect a different egg with a probability of pa $\in$ [0, 1] for a constant number of available host nests. Hence, the host bird may either throw the different eggs or leave the nest and build a new one. One of the essential CS features is Lévy flights to generate new candidate solutions rather than a simple random walk. The following Lévy flight is performed to generate new solutions $x^{(t+1)}$ for the i^{th} cuckoo:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \text{Lévy}(\lambda) \tag{9}$$

where $\alpha > 0$ is the step size. The product $\oplus$ means entry-wise multiplications. The Lévy step size probability distribution is represented by:

$$\text{Lévy} \sim u = t^{-\lambda}, \quad (1 < \lambda \leq 3) \tag{10}$$

which has an infinite variance with an infinite mean. The pseudo-code of CS algorithm is shown below [38]:

Algorithm 2 Cuckoo Search Algorithm

```
1.  Objective function $f(x)$, $x=(x_1, x_2, \ldots x_d,)^T$
2.  Initialize the population of $n$ host nests $x_i$
3.  while $t <$ Max of Iterations do
4.      Get a cuckoo randomly by Lévy flights
5.      Evaluate its fitness $f_i$
6.      Randomly choose $n$ nest $f_j$
7.      If ($f_i > f_j$)
8.          Replace $j$ by the new solution
9.      End if
10.     Abandon a fraction of $p_a$ of worse nests and
        build new ones at new locations via Lévy flights
11.     Keep the best solutions
12.     Rank the solutions and find the current best
13.     $t = t + 1$
14. end while
15. Return the best solution
```

C. PARTICALE SWARM OPTIMIZATION

In 1995, Eberhart and Kennedy proposed the Particle Swarm Optimization (PSO), which mimics the social behavior and search techniques of a swarm of animals, or a flock of birds, or a school of fish, when they adapt their environment to search for their food [36]. The particles communicate and shares its information to find the optimum path to reach its

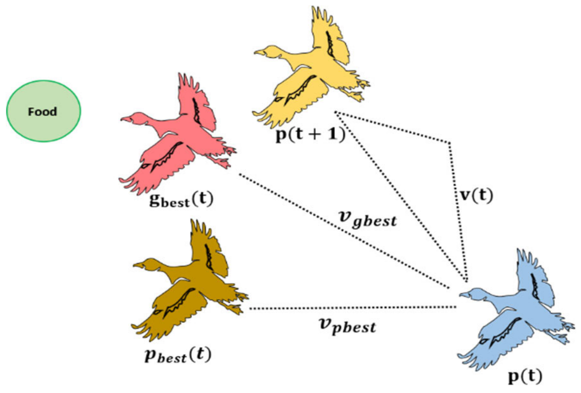

FIGURE 9. Update Particle position and velocity in the search space.

food sources. The shortest path followed is the particle's best position. Based on the current positions of the local and global positions in the search space, each particle identifies and updates its position until the global-optimum position is achieved. Fig. 9 shows how the particle changes its position within the search space to obtain food [54].

Particle movements affect all other individuals within the group, each one of them has its position and velocity defined by equation (11), which presents the best position achieved with respect to all neighbor's best position.

$$p(t+1) = p(t) + v(t+1) \tag{11}$$

Here $p(t+1)$ denotes the updated location of the particle in the swarm, *gbest* defines the global best, $p(t)$ represents the current location of the particle in the swarm, and $v(t+1)$ is the new velocity of the particle in the swarm based on the location of the *gbest*. Based on the current velocity and position of each particle, its own best position *pbest* and the entire population's best position *gbest*, the particle's new velocity and position can be determined as:

$$v(t+1) = \omega * v(t) + c_1 * r_1 * [p_{best}(t) - p(t)] + c_2 * r_2 * [g_{best}(t) - p(t)] \tag{12}$$

where *pbest* is the best position of the particle, *gbest* is the best position of the swarm, $v(t)$ is the current velocity, and r_1, r_2 are random numbers from uniform distribution. Both c_1, c_2 are acceleration coefficients and ω is the inertia weight. The pseudo-code of the Particle Swarm Optimization (PSO) algorithm is shown below [54]:

Algorithm 3 Particle Swarm Optimization Algorithm

```
1.  Initialization Particle's Position
2.  Initialization Particle's velocity
3.  Calculate the fitness values of each particle
4.  while $t < Max\ of\ Iterations$ do
5.      Update the position according to Equation 11
6.      Update the velocity according to Equation 12
7.      Choose the particle having the best fitness value as
        the g-best
8.      Compare P-best of each particle with g-best of swarm
9.      t = t + 1
10. end while
11. Return g-best particle
```

D. GREY WOLF OPTIMIZER

In 2014, Mirjalili and others introduced the Grey Wolf Optimizer (GWO), which is an algorithm that illustrates the Grey Wolf's hierarchical hunting pattern based on how wolves obey a strict social hierarchy [39]. This pattern maintains the stability and assists other wolves during the hunt. The complete wolf pack must follow the orders of the wolf with the most durability and fighting ability. Fig.10 shows the classification of the social hierarchy in a grey wolf pack consisting of the alpha (α), beta (β), delta (δ) and omega (ω) wolves:

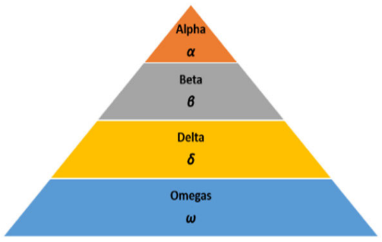

FIGURE 10. Grey wolf social hierarchy.

1) ALPHA (α)

The leader of the pack, at the top of the hierarchy, is mostly responsible for making decisions because it is considered the most qualified wolf among the pack.

2) BETA (β)

The adviser wolf at the second level in the hierarchy, which helps the alpha in decision-making or other pack activities. A beta follows the leader's directions to maintain discipline over the pack.

3) DELTA (δ)

Stands at the third level in the hierarchy, Delta follows the orders of alpha and beta wolves, but dominates and leads the omegas.

4) OMEGA (ω)

The lowest level in the grey wolf social hierarchy, the omega wolves, always follow the commands of all the other dominant wolves in the social hierarchy.

The hunting behavior of the GWO algorithm is guided by the three wolves α, β, and δ, while the ω wolves follow them. Fig.11 illustrates how the position can be updated in the search space for the three wolves. Alpha is the closest location in the search space to prey X_α, which is considered as the first best wolf, X_β is the second-best location for beta wolf, and delta is the third best wolf location X_δ. The rest of

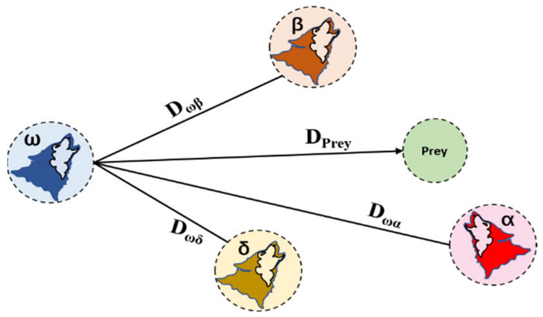

FIGURE 11. Wolves position surrounding the prey.

the pack, omega wolves, will update their positions according to alpha, beta, and delta positions. The locations of wolves are updated as follows:

$$\vec{D} = \left|\vec{C} \cdot \vec{X}_P(t) - \vec{X}(t)\right| \tag{13}$$

$$\vec{X}(t+1) = \vec{X}_P(t) - \vec{A} \cdot \vec{D} \tag{14}$$

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \tag{15}$$

$$\vec{C} = 2 \cdot \vec{r}_2 \tag{16}$$

where, t represents iteration, $\vec{A}$ *and* $\vec{C}$ are coefficient vectors, $\vec{X}_P$ is the prey position vector, $\vec{X}$ is the wolf positions, $\vec{a}$ is the linear coefficient, and $\vec{r}_1$ and $\vec{r}_2$ are random vectors located in the scope [0, 1]. The calculation of distances between the position of current individual and individual of *alpha, beta*, and *delta* are:

$$\vec{D}_\alpha = \left|\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}\right| \tag{17}$$

$$\vec{D}_\beta = \left|\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}\right| \tag{18}$$

$$\vec{D}_\delta = \left|\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}\right| \tag{19}$$

where $\vec{X}_\alpha$, $\vec{X}_\beta$, $\vec{X}_\delta$ are the position vectors, $\vec{C}_1$, $\vec{C}_2$, $\vec{C}_3$ are randomly generated vectors, $\vec{X}$ represents the position vector of current individual. Therefore, the mathematical models for grey wolf hunting are calculated by:

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha \tag{20}$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta \tag{21}$$

$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta \tag{22}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{23}$$

where $\vec{A}_1, \vec{A}_2, \vec{A}_3$ are randomly generated vectors.

The pseudo code for the GWO is shown below [39]:

Algorithm 4 Grey Wolf Optimization Algorithm

```
1.  Initialize the population of the Grey Wolves
2.  Initialize for a, A, and C
3.  Calculate the fitness values of each wolf $X_\alpha$, $X_\beta$, and $X_\delta$
4.  while $t$ < Max of Iterations do
5.      for each searching wolf
6.          Update position using equation 6
7.      end for
8.  Update a, A, and C
9.  Calculate the fitness values of all wolves
10. Update the positions of $X_\alpha$, $X_\beta$, and $X_\delta$
11.     $t = t + 1$
12. end while
13. Return $X_\alpha$
```

V. PROPOSED METHOD

During modeling the vulnerability of PUFs, researchers have used many techniques such as Fault injection-based modeling attacks, Genetic Algorithm, Genetic Programming, and Evolutionary Strategies to model PUF CRPs. From their results, it has been observed that these algorithms require a large number of CRPs to model PUF characteristics; moreover, the attacker may need to have physical access to the PUFs. The motivation for choosing swarm intelligence algorithms is that SI algorithms have fewer parameters than evolutionary methods to adjust, which makes them flexible, robust, and distributive. SI algorithms are easy to implement, more reliable for finding solutions to many complex problems, and converge faster than other algorithms. Also, SI optimizers maintain a large search space of candidate information throughout the iterations. Furthermore, the mathematical model's implementation mechanism is very well developed to avoid local optimization and improve performance, making it easier to combine with practical engineering problems. The Swarm Optimization algorithms are used to build ANN models to analyze the vulnerability of the different PUFs described earlier for modeling attacks. These training algorithms adjust the weights and biases of the ANN until the highest response prediction accuracy can be obtained by finding the optimum set of weights and biases. Based on the objective (loss/error) function for the SI algorithms, the weights are adjusted in each iteration in order to minimize the loss/error function that is used in neural networks to minimize the training error. The Mean Square Error function (MSE), which is the most commonly used parameter for the evaluation of the neural network, is defined in equation (24).

$$Mean\_Square\_Error = \frac{1}{n}\sum_{i=1}^{n}\left(Y_{exp} - Y_{obs_i}\right)^2 \tag{24}$$

where the performance of the network is evaluated based on the difference between the predicted responses (Y_{obs_i}) and the actual responses (Y_{exp}). The average MSE obtained from all training samples is based on the best solutions of the previous iteration. While the current weights and biases of the neural network are updated, the MSE gradually decreases; therefore, after enough iterations, the algorithms can achieve the best solution. For a challenge vector $C = [C_1, C_2, \ldots, C_m]$ of size m, Configurable Ring Oscillator, Inverter based Ring Oscillator, XOR Inverter based ring oscillator, and the Arbiter PUF will generate a response vector $R = [r_1, r_2, \ldots, r_m]$.

The CRPs are fed to the ANN network, where each bit of the challenge vector represents one neuron, and the response bit is the outcome of the neural network. For modeling of the PUFs, it is assumed that if an attacker gets hold of a small set of CRPs $(C, r) = [(C_1, r_1), (C_2, r_2) \ldots, (C_m, r_m)]$, then it can be modeled by the ANN-based models using swarm optimization, GSA, CS, PSO and GWO algorithms to predict the remaining set of CRPs. For modeling the Modified XOR-Inverter ROPUF and the Hybrid Delay based PUF, the challenge vector is defined as $C = [C_1, C_2, \ldots, C_m]^T$, and the response matrix for the individual PUF is given as:

$$R = \begin{bmatrix} r_{11} & r_{12} & \ldots & r_{1n} \\ r_{21} & r_{22} & \ldots & r_{2n} \\ \vdots & \vdots & \ldots & \vdots \\ r_{m1} & r_{m2} & \ldots & r_{mn} \end{bmatrix} \tag{25}$$

where both the challenge and response bits are of the same size. The prediction accuracy for n bit response can be calculated as:

$$Prediction\ accuracy = \sum_{r=1}^{n} nCr/number\ of\ challenges \tag{26}$$

Algorithm 5 outlines the steps of how the model trains the Artificial Neural Network based on Swarm Intelligence Algorithms.

Algorithm 5 Training ANN Using SI Algorithms

```
1.  Initialize all the parameters of Swarm Algorithm
2.  Constricting of ANN learning structure
3.  Initialize the weights and biases of the Neural Network
4.  while $t < Max\ of\ Iterations$ do
5.     Map the Challenges Vector C = [$C_1, C_2, \ldots, C_m$]
         into the input layer of AN
6.     Calculate the predicted response R
7.     Compare the predicted response R to the actual
         response R
8.     Calculate the new global optimum value of weights
         and biases using swarm algorithm
9.     Update the weights and biases of the ANN using
         Swarm Intelligence optimizers
10.    t = t + 1
11. end while
12. Return weights, biases and predicted accuracy
```

A. COMPUTATIONAL COMPLEXITY ANALYSIS

The computational complexity of the proposed algorithm (Algorithm 5) accounts for the execution time of the algorithm based on its structure. For Algorithm 5, the computational complexity for each step can be described as follows:

- The computational complexity of initialization the weights and biases is O(N $\times$ dim) time, where *N* represents the population size and *dim* represents the dimension of the problem.
- In step 5 (mapping the challenge vector) of the proposed algorithm, for each iteration, the challenge vector is mapped into the input layer of ANN with constant computational complexity of $O(1)$; the iteration loop technically runs in O(Iter), therefore the final time complexity for mapping is O(Iter).
- The calculation of predicting the response and comparing it with the actual response in steps 6 and 7 have a computational complexity of (Iter $\times$ L), where L is the total number of training CRPs.
- For step 8, in each iteration the computational complexity represents the calculation of the new global optimum value of weights and biases (Iter $\times$ N).
- Step 9 represents the updated weights and biases values with computational complexity of (Iter $\times$ N $\times$ dim).
- Since the total number of iterations is not more than Iter_{Max}, the total time complexity is: $O(\text{Iter}_{\text{Max}}) + O(\text{Iter}_{\text{Max}} \times \text{L}) + O(\text{Iter}_{\text{Max}} \times \text{N}) + O(\text{Iter}_{\text{Max}} \times \text{N} \times \text{dim}) + O(\text{dim} \times \text{N})$.

As seen from the above analysis, the proposed algorithm's computational complexity depends on the size of population (N), dimension (dim), and iterations (Iter).

VI. EXPERIMENTAL RESULTS ANALYSIS AND DISCUSSIONS

To analyze the vulnerability of the various PUFs to ANN-based attacks using Swarm Intelligence algorithms, a subset of the randomly chosen CRPs is used as the training set. An accuracy score evaluates the attack resistance in terms of the percentage of successful response predictions. The Swarm Intelligence algorithms used to train the ANN network to predict PUF CRPs are implemented using Python 3.5 (64 Bit) frameworks. For training the CRPs, a 2.3 GHz PC with 16 GB RAM and 2GB Graphics card is used. The response prediction accuracy is determined by using cross-validation of ten blocks K-fold method [55]. One of these ten partitions is used as the test set, while the other nine cumulatively serve as the training set. The ANN learning network structure used in the experiment is a 3- Multi-Layer Perceptron (MLP) with 33 nodes in the hidden layer. It is observed that the ANN method dramatically improves the learning rate for the first four PUFs, but still fails to learn the last two dual-mode PUFs. Different parameters and hyperparameters used in ANN for training and prediction of the CRPs are listed in Table 1. In order to verify the performance of the proposed method, we chose well-known ANN-based optimization algorithms (RMSprop, Adadelta, Adam, and Nadam) for the purpose of comparison under the same experimental environment and the same platform for a fair comparison. Furthermore, we used the same ANN structure in terms of the number of hidden layers, nodes, and activation functions. The number of individuals that have been used for all the algorithms is 100, and each run stops when the maximal number of 1000 iterations is achieved. Finally, a statistical analysis of the method's results has been performed.

TABLE 1. Parameters values used.

Parameters	Values
Number of training CRPs	10000
Number of k folds	10
Loss Functions	MSE
Number of nodes in hidden layer	33
Hidden Layer Activation Function	ReLU
Output Layer Activation Function	Sigmoid
Learning rate	0.01

A. MACHINE LEARNING ANN-BASED MODELING ATTACKS

ANN-based models using four well-known optimization algorithms are used to perform attacks on different PUFs. These models are RMSprop, Adadelta, Adam, and Nadam optimizations. The ANN structure in terms of the number of hidden layers, nodes, and activation functions are shown in Table 1. Moreover, the training conditions of the network-based model are given in table 2.

TABLE 2. Initial parameters set in ANN optimizers.

Algorithm	Parameter	Default Value
Adam	Alpha (α)	0.001
	Beta1 (β1)	0.9
	Beta2 (β2)	0.999
	Number of iterations	1000
RMSprop	Discounting factor (rho)	0.9
	Momentum	0.0
	Centered	False
	Number of iterations	1000
Nadam	Alpha (α)	0.001
	Epsilon (ε)	1e-08
	amsgrad	False
	Number of iterations	1000
Adadelta	Discounting factor (rho)	0.9
	Epsilon (ε)	1e-08
	**kwargs	clipvalue
	Number of iterations	1000

As shown in Table 3, it is observed that the best accuracy for response prediction is 85.0% for the ANN-based modeling with Adam on the Configurable ROPUF.

TABLE 3. ANN-based prediction accuracy for PUFs.

Type of PUF	Adadelta	RMSprop %	Adam %	Nadam %
Inverter ROPUF	75.8	77.1	78.3	79.4
Config. ROPUF	83.8	84.4	85.0	84.1
Xor-ROPUF	68.0	69.2	70.3	70.0
Arbiter PUF	69.3	70.1	71.9	72.1
Hybrid PUF	7.5	8.1	8.9	9.7
Modified PUF	9.1	9.7	10.3	10.7

In the case of the Hybrid Delay based PUF and the Modified XOR-Inverter ROPUF, the two PUFs which are specially designed to thwart machine learning attacks, it is noted that the models are unable to predict the responses with higher prediction accuracy. The best prediction accuracy of 10.7% is observed for the Nadam optimization. Figs. 12(a), (b), (c), (d), (e), (f) show plots of the prediction accuracies versus the number of iterations for the Inverter ROPUF, Configurable ROPUF, XOR-Inverter ROPUF, Arbiter PUF, Hybrid Delay based PUF, and Modified ROPUF, respectively. It can be concluded from these plots that the prediction accuracy of the Nadam algorithm is higher than the other algorithms. Fig. 13 shows the loss function of the different ANN-based optimization algorithms. It is observed from this figure that the Nadam optimizer converges faster than the other optimization algorithms.

B. SWARM INTELLIGENCE BASED MODEL ATTACKS

In this section, we describe how the Swarm Intelligence algorithms are used to train the ANN. However, for these algorithms to reach their maximum performance and achieve the best results, proper settings of the initial parameters are required. The parameters chosen for SI algorithms to simulate the GSA, CS, PSO, and GWO algorithms are selected based on references [29], [38], [39], [54], and are given in Table 4. ANN-based models are trained for 1000 iterations and the algorithms are tested with an initial population of individuals in the range of 5-150. However, no improvement in prediction accuracy is achieved by increasing the number of individuals to more than 100; therefore, the number of individuals is kept at 100.

TABLE 4. Initial parameters set in swarm algorithms.

Algorithm	Parameter	Default Value
CS	Detection probability (p_a)	0.25
	Step length control	0.01
	Number of iterations	1000
	Number of bird nests	100
	Dimension	595
GSA	Initial gravitational constant (G_0)	100
	Constant values initialization (α)	20
	Number of iterations	1000
	Population Size	100
	Dimension	595
PSO	Cognitive influence (C1)	2
	Social influence (C2)	2
	Inertia weight (ω)	[0.2, 0.9]
	Number of iterations	1000
	Number of particles	100
	Dimension	595
GWO	Decreases linearly ($\vec{a}$)	[2, 0]
	Vector contains random values ($\vec{A}$)	[-2$\vec{a}$, 2$\vec{a}$]
	Vector contains random values ($\vec{C}$)	[0, 2]
	Number of iterations	1000
	Number of wolves	100
	Dimension	595

Table 5 lists experimental results for the accuracy, standard deviation, and runtime for four different PUFs using the

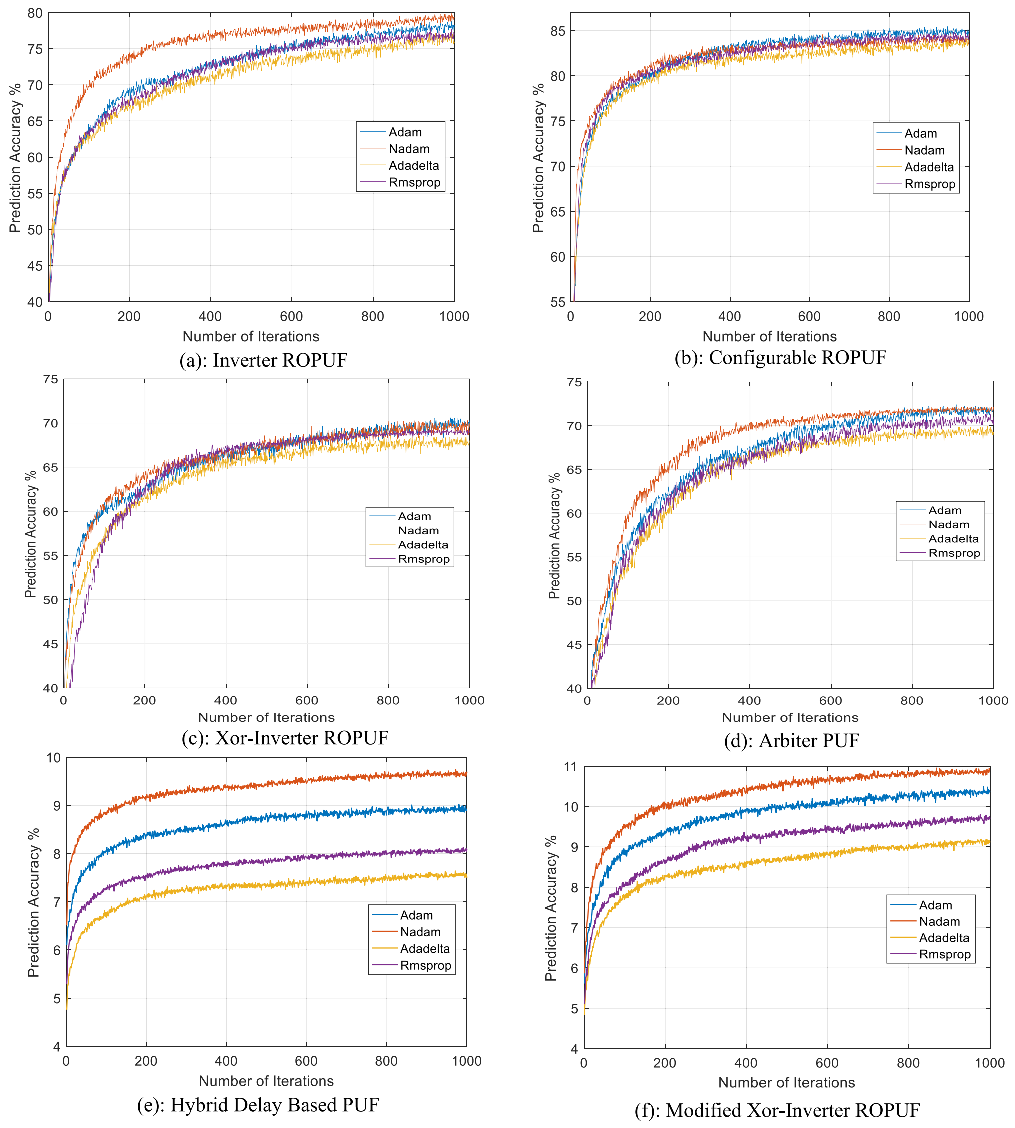

FIGURE 12. **ANN-based prediction accuracy vs. number of Iteration for different PUFs.**

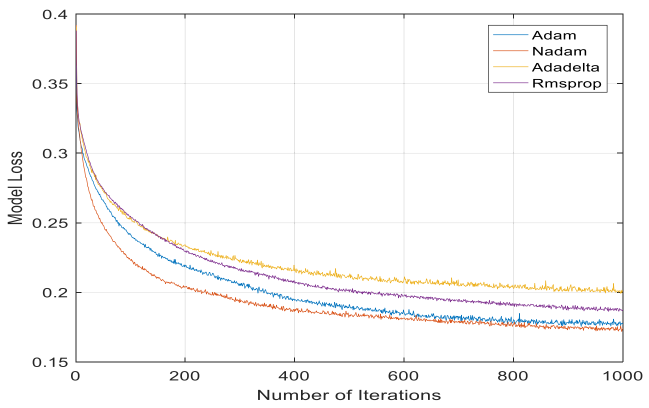

FIGURE 13. **Loss function vs. number of Iteration for different ANN optimizers.**

GSA, CS, PSO and GWO Swarm Intelligence algorithms. From the table, it is evident that the PUF structures are vulnerable to Swarm Intelligence-based model attacks with prediction accuracies ranging from 71.1% - 88.3%. In contrast, for the machine learning ANN-based models, the prediction accuracies range from 68.0% to 85.0 %. Also, it is found from Table 3 and Table 5 that the prediction accuracies are much better for each of the listed PUFs when the GSA, CS, PSO and GWO based modeling attacks are used. Figs. 14 (a), (b), (c), (d), (e), (f) show plots of the prediction accuracies versus the number of iterations for the different PUF designs. It can be concluded from these plots that the prediction accuracy of the GWO algorithm is higher than the other algorithms. Also, the plots show that the GWO converges fast. For the two PUFs that were especially designed to thwart machine learning-based attacks, namely: the Hybrid Delay based PUF and the Modified XOR-Inverter ROPUF, it is found that the prediction accuracies using the Swarm Intelligence algorithms are in the range of 9.8% to

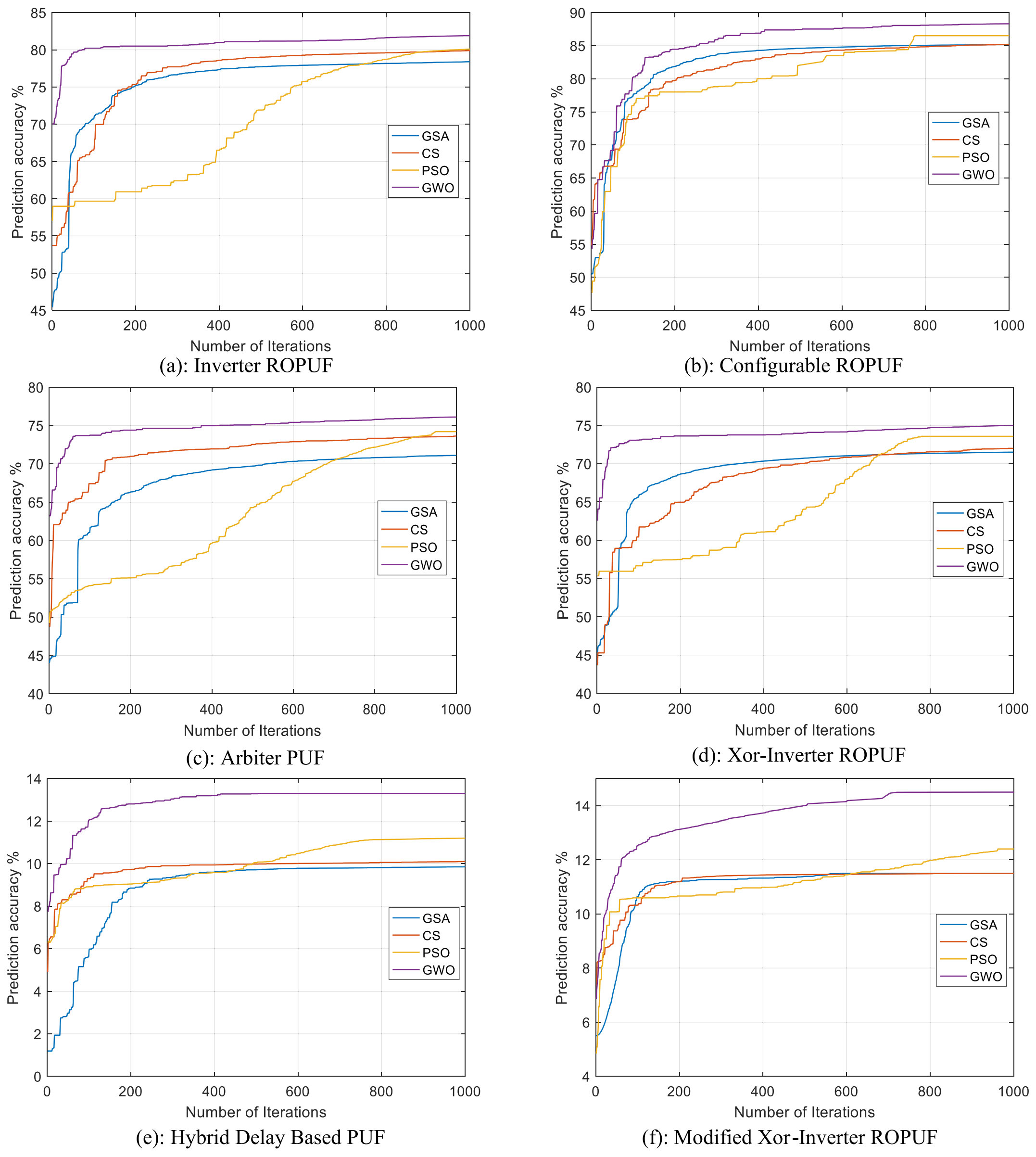

FIGURE 14. Swarm-based prediction accuracy vs. number of iteration for different PUFs.

14.5%, as shown in Table 5. Although in the low range, the prediction accuracies are better than those obtained from Machine Learning ANN-based attacks which range from 7.5% -10.7%. Here, also, it is observed that the performance of the GWO model is better than the others in terms of prediction accuracies. Fig. 15 shows the loss function of the different swarm-based algorithms. It is observed from the figure that the GWO converges faster than the other algorithms.

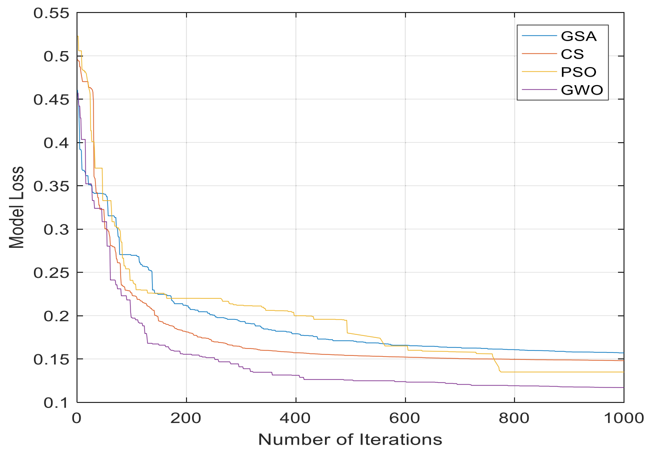

FIGURE 15. Loss function vs. number of Iteration for different swarm algorithms.

C. COMPARATIVE ANALYSIS AMONG DIFFERENT ALGORITHMS

The prediction accuracies are much better for each of the listed PUFs when swarm-based modeling attacks are used. Table 6 summarizes the prediction accuracies for the six different types of PUFs under study. It is observed from

TABLE 5. Swarm-based prediction accuracy, standard deviation, and runtime for PUFs.

Type of PUF	GSA			CS			PSO			GWO		
	Accuracy %	STD	Time (Sec)	Accuracy %	STD	Time (Sec)	Accuracy %	STD	Time (Sec)	Accuracy %	STD	Time (Sec)
Inverter ROPUF	78.6	2.15	292.8	79.9	0.98	340.2	80.1	2.45	354.9	81.9	1.97	243.7
Configurable ROPUF	85.3	3.37	521.8	85.2	2.61	530.1	86.5	3.75	585.3	88.3	4.10	452.2
Xor-Inverter ROPUF	71.5	1.50	489.5	72.0	1.80	497.3	73.5	1.33	545.1	75.0	1.87	446.3
Arbiter PUF	71.1	1.13	470.9	73.3	1.22	517.6	74.2	0.85	532.1	76.1	0.97	429.1
Hybrid Delay PUF	9.8	1.57	513.2	10.1	1.71	550.3	11.2	0.67	567.6	13.3	0.55	490.3
Modified ROPUF	11.3	1.43	501.3	11.5	0.97	530.1	12.4	0.53	545.3	14.5	0.71	440.9

TABLE 6. Prediction accuracy comparison for different algorithms.

Type of PUF	Adadelta %	RMSprop %	Adam %	Nadam %	GSA %	CS %	PSO %	GWO %
Inverter ROPUF	75.8	77.1	78.3	79.4	78.6	79.9	80.1	81.9
Configurable ROPUF	83.8	84.4	85.0	84.1	85.3	85.2	86.5	88.3
Xor-Inverter ROPUF	68.0	69.2	70.3	70.0	71.5	72.0	73.5	75.0
Arbiter PUF	69.3	70.1	71.9	72.1	71.1	73.3	74.2	76.1
Hybrid Based PUF	7.5	8.1	8.9	9.7	9.8	10.1	11.2	13.3
Modified ROPUF	9.1	9.7	10.3	10.7	11.3	11.5	12.4	14.5

this table that the prediction accuracies, when the Swarm Intelligence models (GSA, CS, PSO & GWO) are used, are much better than the other algorithms for each of the listed PUFs. For easy comparison, the results in Table 6 are also shown in the chart of Fig. 16. It is clear from this figure that the GSA, CS, PSO and GWO optimizations in ANN give better prediction accuracy results than Adadelta, RMSprop, Adam, and Nadam optimization algorithms. The Swarm Intelligence-based model attacks have prediction accuracies ranging from 71.1% - 88.3%. In contrast, for the machine learning ANN-based models, the prediction accuracies range from 68.0% to 85.0%. The prediction accuracies for the modified PUFs (Hybrid and Modified Inverter) are less because these PUFs have been especially designed to thwart machine learning attacks. It is found that the prediction accuracies using the Swarm Intelligence algorithms are in the range of 9.8% to 14.5%, while the results obtained from Machine Learning ANN-based attacks with range from 7.5% −10.7%.

D. STATISTICAL ANALYSIS OF THE RESULTS

This subsection explains the statistical analysis of the various algorithm results, where multiple comparison procedures have been employed. In order to apply statistical analysis, a null hypothesis is defined, which implies that all the algorithms have the same performance without a significant difference; therefore, a denial of this hypothesis suggests the existence of differences between these algorithms. If the hypothesis is rejected, a significance value α is applied to decide the rejection level. The p-values are used to describe the significance of the hypothesis test. If the p-value is more significant than α, then there is not enough evidence to reject the null hypothesis. Otherwise, the hypothesis is rejected, which indicates that the algorithms have different performances. The Nonparametric Friedman test is used to compute p-values to define significant differences between the algorithms' prediction accuracy [56]. Then, a significance value $\alpha = 0.05$ is chosen. In computing the Friedman Value $\boldsymbol{F_f}$, the test ranks the algorithms according to the highest

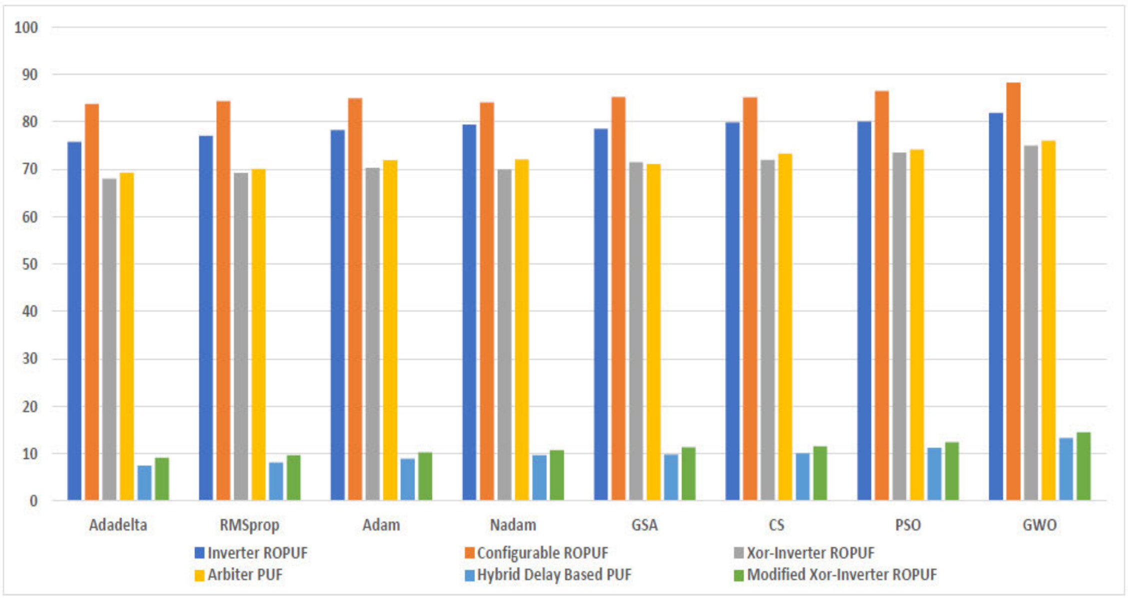

FIGURE 16. Prediction accuracies for different types of PUFs using different optimization models.

TABLE 7. Average rankings of the algorithms by Friedman test.

Algorithm	Ranking
GWO	1
PSO	2
CS	3.25
GSA	4.25
Nadam	4.75
RMSprop	5.75
Adadelta	7

prediction accuracy (Rank 1), the second highest (Rank 2), down to the lowest ranking. The Friedman test computes F_f Value as:

$$F_f = \frac{12n}{k\,(k+1)}\left[\sum R^2 - \frac{k\,(k+1)^2}{4}\right] \tag{27}$$

where, R is the ranks, n is the number of PUF datasets, k is the number of algorithms, and the statistic is distributed according to F_f with $k-1$ degrees of freedom [57], [58].

Table 7 shows the obtained average rankings of the algorithms by the Friedman Test based on prediction accuracy.

GWO has the best performance in prediction accuracy among all algorithms; therefore, it has a rank of 1 and will be used as the control algorithm. The result obtained from the Friedman test, including its corresponding associated p-value, is shown in Table 8. From the table, it is observed that the p-value is lower than the level of significance (0.05); therefore, there are significant performance differences between the algorithms, which implies that the null hypothesis is rejected. Considering the differences between the algorithms, we need a post-hoc procedure to identify these differences and then find out the p-value in order to determine the hypothesis rejection degree. Holm's procedure has been used to determine whether the control algorithm presents statistical differences concerning the remaining algorithms [59].

TABLE 8. Results of the Friedman tests.

Friedman Value	p-value
21.99937	0.00121

TABLE 9. Adjusted p-values. GWO is the control algorithm.

Algorithm	Z	Unadjusted p-value	p-Holm
Adadelta	4.242641	0.000022	0.000132
RMSprop	3.358757	0.000392	0.001960
Nadam	2.651650	0.004006	0.016024
GSA	2.298097	0.010781	0.032343
CS	1.64521	0.049964	0.099928
PSO	0.707107	0.239752	0.239752

Holm's procedure compares the control algorithm, GWO, with the other remaining algorithms, which consider a multiple comparison procedure. The test statistic, z value, is used to find the corresponding probability from the table of the normal distribution:

$$Z = \frac{R_i - R_j}{\sqrt{\frac{k(k+1)}{6N}}} \tag{28}$$

where, R_i and R_j are the average rankings by the Friedman test of the algorithms compared [60]. These unadjusted p values are used to compute p-Holm sequentially and test the hypotheses ordered by their significance level of confidence α. Table 9 shows that when the highest prediction accuracy algorithm (GWO) is used as a control algorithm, it performs better than Adadelta, RMSprop, Nadam and GSA with $\alpha = 0.05$, and GWO outperforms all the algorithms with $\alpha = 0.10$ except PSO.

VII. CONCLUSION

Various Machine Learning based attack models have been used recently to breach the security of PUFs. In this work, we study six different types of PUFs to ascertain their resiliency to such attacks. We especially focus on swarm intelligence-based algorithms to further study the vulnerability of these PUFs to learning attacks. To the best of our knowledge, swarm-based algorithms have not been investigated earlier to test the security of PUFs. In this paper, Artificial Neural Network modeling attacks on different types of PUFs using the Gravitational Search Algorithm (GSA), Cuckoo Search Algorithm (CS), Particle Swarm and Grey Wolf Optimization are presented. From the results, it is observed that the swarm intelligence algorithms produce better response prediction accuracy results (71.1% - 88.3%) when compared to other well-known Machine Learning ANN-based algorithms (68.0% - 85.0%). Amongst the SI algorithms, the GWO algorithm performs better in predicting the CRPs than the rest. It is observed that the Configurable ROPUF is the most vulnerable and its response can be predicted with an accuracy of 88.3% when the GWO is used. For the Modified XOR-Inverter ROPUF, which has been especially designed to thwart machine learning attacks, it is found that the Grey Wolf Optimizer can predict the response with 14.5% accuracy. Although swarm intelligence algorithms used in this paper require considerable computational time, the prediction accuracy of the proposed method is better than ANN-based models. For future work, the proposed method can be used to improve the performance metrics of PUFs and for developing countermeasures against modeling attacks.

REFERENCES

[1] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

[2] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[3] M. Tehranipoor, H. Salmani, and X. Zhang, *Integrated Circuit Authentication*, vol. 10. Cham, Switzerland: Springer, 2014, pp. 973–978.

[4] F. Koushanfar, "Hardware metering: A survey," in *Introduction to Hardware Security and Trust*. New York, NY, USA: Springer, 2012, pp. 103–122.

[5] G. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. DAC*, 2007, pp. 9–14.

[6] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proc. IEEE*, vol. 102, no. 8, pp. 1126–1141, Aug. 2014.

[7] S. Gören, O. Ozkurt, A. Yildiz, H. F. Ugurdag, R. S. Chakraborty, and D. Mukhopadhyay, "Partial bitstream protection for low-cost FPGAs with physical unclonable function, obfuscation, and dynamic partial self reconfiguration," *Comput. Electr. Eng.*, vol. 39, no. 2, pp. 386–397, Feb. 2013.

[8] N. A. Hazari, F. Alsulami, and M. Niamat, "FPGA IP obfuscation using ring oscillator physical unclonable function," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, Dayton, OH, USA, Jul. 2018, pp. 105–108.

[9] F. Amsaad, T. Hoque, and M. Niamat, "Analyzing the performance of a configurable ROPUF design controlled by programmable XOR gates," in *Proc. IEEE 58th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2015, pp. 1–4.

[10] M. Choudhury, N. Pundir, M. Niamat, and M. Mustapa, "Analysis of a novel stage configurable ROPUF design," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Boston, MA, USA, Aug. 2017, pp. 942–945.

[11] J. Sölter, "Cryptanalysis of electrical PUFs via machine learning algorithms," M.S. thesis, Technische Univ. München, München, Germany, 2009.

[12] U. R. Ührmair, F. Sehnke, J. S. Ölter, G. Dror, S. Devadas, and J. Ü. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proc. 17th ACM Conf. Comput. Commun. Secur. (CCS)*, 2010, pp. 237–249.

[13] U. Röhrmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF modeling attacks on simulated and silicon data," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 11, pp. 1876–1891, Nov. 2013.

[14] J. Delvaux, "Machine-learning attacks on polyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF-FSMs," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 8, pp. 2043–2058, Aug. 2019.

[15] F. Ganji, S. Tajik, and J.-P. Seifert, "PAC learning of arbiter PUFs," *J. Cryptograph. Eng.*, vol. 6, no. 3, pp. 249–258, Sep. 2016.

[16] J. Delvaux and I. Verbauwhede, "Fault injection modeling attacks on 65 nm arbiter and RO sum PUFs via environmental changes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 6, pp. 1701–1713, Jun. 2014.

[17] J. Shi, Y. Lu, and J. Zhang, "Approximation attacks on strong PUFs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2138–2151, Oct. 2020.

[18] J. Delvaux and I. Verbauwhede, "Side channel modeling attacks on 65 nm arbiter PUFs exploiting CMOS device noise," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)*, Austin, TX, USA, Jun. 2013, pp. 137–142.

[19] X. Xu and W. Burleson, "Hybrid side-channel/machine-learning attacks on PUFs: A new threat?" in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, 2014, pp. 1–6.

[20] M. Khalafalla and C. Gebotys, "PUFs deep attacks: Enhanced modeling attacks using deep learning techniques to break the security of double arbiter PUFs," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Florence, Italy, Mar. 2019, pp. 204–209.

[21] Q. Guo, J. Ye, Y. Gong, Y. Hu, and X. Li, "Efficient attack on non-linear current mirror PUF with genetic algorithm," in *Proc. IEEE 25th Asian Test Symp. (ATS)*, Hiroshima, Japan, Nov. 2016, pp. 49–54.

[22] I. Saha, R. R. Jeldi, and R. S. Chakraborty, "Model building attacks on physically unclonable functions using genetic programming," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)*, Austin, TX, USA, Jun. 2013, pp. 41–44.

[23] Y. Xu, Y. Lao, W. Liu, Z. Zhang, X. You, and C. Zhang, "Mathematical modeling analysis of strong physical unclonable functions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4426–4438, Dec. 2020.

[24] N. A. Hazari, A. Oun, and M. Niamat, "Analysis and machine learning vulnerability assessment of XOR-inverter based ring oscillator PUF design," in *Proc. IEEE 62nd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Dallas, TX, USA, Aug. 2019, pp. 590–593.

[25] J. H. Holland, "Genetic algorithms," *Sci. Amer.*, vol. 267, no. 1, pp. 66–73, 1992.

[26] J. R. Koza and J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT Press, 1992.

[27] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.*, vol. 11, no. 1, pp. 1–18, Mar. 2003.

[28] R. Storn and K. Price, ‘‘Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces,’’ *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.
[29] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, ‘‘GSA: A gravitational search algorithm,’’ *J. Inf. Sci.*, vol. 179, no. 13, pp. 2232–2248, 2009.
[30] O. K. Erol and I. Eksin, ‘‘A new optimization method: Big bang-big crunch,’’ *Adv. Eng. Softw.*, vol. 37, no. 2, pp. 106–111, 2006.
[31] R. A. Formato, ‘‘Central force optimization: A new metaheuristic with applications in applied electromagnetics,’’ *Prog. Electromagn. Res.*, vol. 77, pp. 425–491, Jan. 2007.
[32] H. S. Hosseini, ‘‘Principal components analysis by the galaxy-based search algorithm: A novel metaheuristic for continuous optimisation,’’ *Int. J. Comput. Sci. Eng.*, vol. 6, nos. 1–2, pp. 132–140, 2011.
[33] B. Webster and P. J. Bernhard, ‘‘A local search optimization algorithm based on natural principles of gravitation,’’ Florida Inst. Technol., Melbourne, FL, USA, Tech. Rep. #CS-2003-10, 2003.
[34] A. Kaveh and S. Talatahari, ‘‘A novel heuristic optimization method: Charged system search,’’ *Acta Mechanica*, vol. 213, nos. 3–4, pp. 267–289, 2010.
[35] S. Selvaraj and E. Choi, ‘‘Survey of swarm intelligence algorithms,’’ in *Proc. 3rd Int. Conf. Softw. Eng. Inf. Manage. (ICSIM)*. New York, NY, USA: Association for Computing Machinery, Jan. 2020, pp. 69–73.
[36] J. Kennedy and R. Eberhart, ‘‘Particle swarm optimization,’’ in *Proc. Int. Conf. Neural Netw. (ICNN)*, vol. 4, 1995, pp. 1942–1948.
[37] M. Dorigo, M. Birattari, and T. Stutzle, ‘‘Ant colony optimization,’’ *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.
[38] X.-S. Yangc and S. Deb, ‘‘Cuckoo search via Lévy flights,’’ in *Proc. World Congr. Nature Biologically Inspired Comput. (NaBIC)*, 2009, pp. 210–214.
[39] S. Mirjalili, S. M. Mirjalili, and A. Lewis, ‘‘Grey wolf optimizer,’’ *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.
[40] A. Kumar and S. Chakarverty, ‘‘Design optimization for reliable embedded system using cuckoo search,’’ in *Proc. 3rd Int. Conf. Electron. Comput. Technol. (ICECT)*, Apr. 2011, pp. 264–268.
[41] M. Abdulgader, S. Lakshminarayanan, and D. Kaur, ‘‘Efficient energy management for smart homes with grey wolf optimizer,’’ in *Proc. IEEE Int. Conf. Electro Inf. Technol. (EIT)*, May 2017, pp. 388–393.
[42] H. Liu, G. Hua, H. Yin, and Y. Xu, ‘‘An intelligent grey wolf optimizer algorithm for distributed compressed sensing,’’ *Comput. Intell. Neurosci.*, vol. 2018, pp. 1–10, Jan. 2018.
[43] A. Oun and M. Niamat, ‘‘Defense mechanism vulnerability analysis of ring oscillator PUFs against neural network modeling attacks using the dragonfly algorithm,’’ in *Proc. IEEE Int. Conf. Electro Inf. Technol. (EIT)*, Jul. 2020, pp. 378–382.
[44] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, ‘‘Silicon physical random functions,’’ in *Proc. 9th ACM Conf. Comput. Commun. Secur. (CCS)*, 2002, pp. 148–160.
[45] A. Maiti and P. Schaumont, ‘‘Improving the quality of a physical unclonable function using configurable ring oscillators,’’ in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 703–707.
[46] M. T. Rahman, D. Forte, J. Fahrny, and M. Tehranipoor, ‘‘ARO-PUF: An aging-resistant ring oscillator PUF design,’’ in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.
[47] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, ‘‘Extracting secret keys from integrated circuits,’’ *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 10, pp. 1200–1205, Oct. 2005.
[48] N. Pundir, F. Amsaad, M. Choudhury, and M. Niamat, ‘‘Novel technique to improve strength of weak arbiter PUF,’’ in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2017, pp. 1532–1535.
[49] M. Matsumoto and T. Nishimura, ‘‘Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator,’’ *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, p. 330, Jan. 1998.
[50] M. I. Velazco and C. Lyra, ‘‘Optimization with neural networks trained by evolutionary algorithms,’’ in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 2, 2002, pp. 1516–1521.
[51] A. West and D. Saad, ‘‘Adaptive back-propagation in on-line learning of multilayer networks,’’ in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 8, 1996, pp. 323–329.
[52] X.-S. Yang and M. Karamanoglu, ‘‘Swarm intelligence and bio-inspired computation: An overview,’’ in *Swarm Intelligence and Bio-Inspired Computation*. Amsterdam, The Netherlands: Elsevier, 2013, pp. 3–23.
[53] L. Brezočnik, I. Fister, and V. Podgorelec, ‘‘Swarm intelligence algorithms for feature selection: A review,’’ *Appl. Sci.*, vol. 8, no. 9, p. 1521, Sep. 2018.
[54] Y. Shi, ‘‘Particle swarm optimization: Developments, applications and resources,’’ in *Proc. Congr. Evol. Comput.*, vol. 1, 2001, pp. 81–86.
[55] R. Kohavi, ‘‘A study of cross-validation and bootstrap for accuracy estimation and model selection,’’ in *Proc. IJCAI*, 1995, vol. 14, no. 2, pp. 1–7.
[56] M. Friedman, ‘‘The use of ranks to avoid the assumption of normality implicit in the analysis of variance,’’ *J. Amer. Statist. Assoc.*, vol. 32, no. 200, pp. 675–701, Dec. 1937.
[57] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th ed. London, U.K.: Chapman & Hall, 2006.
[58] J. Derrac, S. García, D. Molina, and F. Herrera, ‘‘A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms,’’ *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, Mar. 2011.
[59] S. Holm, ‘‘A simple sequentially rejective multiple test procedure,’’ *Scand. J. Statist.*, vol. 6, pp. 65–70, Jan. 1979.
[60] W. Daniel, *Applied Nonparametric Statistics*, 2nd ed. Boston, MA, USA: Duxbury Thomson Learning, 2000.

AHMED OUN (Graduate Student Member, IEEE) received the M.S. degree in electrical engineering from the University of Bridgeport, Bridgeport, CT, USA, in December 2012. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computer Science, The University of Toledo, Toledo, OH, USA. From January 2014 to May 2017, he served as a Project Manager for General Electric International Inc. His research interests include hardware oriented security and trust, testing of digital VLSI circuits, field-programmable gate arrays, swarm and machine learning algorithms, optimization techniques, neural networks, and the IoT devices.

NOOR AHMAD HAZARI received the B.Sc. degree in electrical and electronics engineering from Khulna University of Engineering and Technology (KUET), Khulna, Bangladesh, and the Ph.D. degree in electrical engineering from The University of Toledo, Toledo, OH, USA. He is currently working as a Postdoctoral Fellow at The University of Toledo on the ‘‘Assured and trusted digital microelectronics’’ project funded by the U.S. Air Force. His research interests include hardware security, FPGA design security, PUFs, machine learning, and blockchain technology for hardware security.

MOHAMMED Y. NIAMAT (Life Member, IEEE) received the bachelor's degree in electrical engineering from Aligarh Muslim University, Aligarh, India, the master's degree in electrical engineering from the University of Saskatchewan, Saskatchewan, Canada, and the Ph.D. degree from The University of Toledo, OH, USA, in 1989. From 1996 to 1997, he was a Visiting Associate Professor with the Center for Reliable Computing, Stanford University. He is currently the focus Group Leader for the High-Performance Computing Research Group, Department of Electrical Engineering and Computer Science, The University of Toledo. He has supervised more than 50 graduate students, including Noor Ahmad Hazari and Ahmed Oun.