COMPUTERS & SECURITY XXX (XXXX) XXX

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Random active shield generation based on modified artificial fish-swarm algorithm

Ruishan Xin[a,b], Yidong Yuan[a,b], Jiaji He[a,b], Shuai Zhen[a,b], Yiqiang Zhao[a,b,*]

[a] School of Microelectronics, Tianjin University, Tianjin 300072, China
[b] Tianjin Key Laboratory of Imaging and Sensing Microelectronic Technology, Tianjin 300072, China

ARTICLE INFO

Article history:
Received 13 November 2018
Revised 8 April 2019
Accepted 8 June 2019
Available online xxx

Keywords:
Integrated circuits
Invasive attacks
Active shield
Random Hamiltonian path
Artificial fish-swarm algorithm

ABSTRACT

Active shield has already been a primitive sensor of security critical integrated circuits for detecting invasive attacks. Because of the complex topology structure, the active shield based on random Hamiltonian path has a high security level. However, the available generation algorithms of this random path have poor efficiency when shield area is large, restricting its application in integrated circuits. In this paper, a novel generation algorithm of random active shield is proposed using a modified artificial fish-swarm algorithm. By changing the random selection strategy of the generation process, the proposed algorithm makes each selection turn into a successful combination, thus improving the efficiency greatly. Simulations prove that this algorithm is seventeen times faster than the classical Cycle Merging algorithm, while keeping good randomness. Meanwhile, the proposed algorithm is capable of large shield generation. In a 0.18 μm CMOS process with the minimum top-metal width and space of 1.5 μm, the active shield with the area of 3×3 mm^2 only needs approximately 2 h for generation.

1. Introduction

Invasive attacks have already been employed to extracting sensitive information from integrated circuits (ICs) (Anderson and Kuhn, 1996; Handschuh et al., 1999; Quadir et al., 2016; Van Tilborg and Jajodia, 2011). With the help of modern test equipment (Boit et al., 2013; Helfmeier et al., 2013), such as microprobing and focused ion beam (FIB) workstations, invasive attacks can be implemented directly on chips. Attackers can change the connections of internal wires (Ray, 2009), draw artificial pads conducting into the inner circuits and monitor the signals on data buses (Kömmerling and Kuhn, 1999; Weingart, 2000). Therefore, sensitive information like cryptographic keys can be obtained easily. Experiments have proved that these attacks can extract critical information from commercial chips (Tarnovsky, 2008). As a result, the devices and systems, such as computers, embedded devices and intelligent control systems, take integrated circuits as the cores. They have to face the fact that they are unsafe under the threats of invasive attacks.

Various countermeasures (Beit-Grogger and Riegebauer, 2005; Briais et al., 2012a; 2012b; Helfmeier et al., 2012; Manich et al., 2012; Mishra et al., 2017; Ngo et al., 2017; Shi et al., 2016; Xin et al., 2019) are put forward to detect invasive attacks for protecting chips. Some special sensors, such as the capacitance sensor based on ring oscillator (Manich et al., 2012) and the charge sensor with an antenna (Helfmeier et al., 2012), cannot provide protection against microprobing and FIB attacks simultaneously. Particularly, active shield is an effective countermeasure which can resist both of the attacks. Though some backside invasive attacks (Boit et al., 2013; Helfmeier et al., 2013) are given attention because of the circumvention

* Corresponding author at: School of Microelectronics, Tianjin University, Tianjin 300072, China.
E-mail address: yq_zhao@tju.edu.cn (Y. Zhao).
https://doi.org/10.1016/j.cose.2019.06.006
0167-4048/

of the active shield, such attacks are costly and difficult to be performed in some ICs. Reverse engineering must be implemented before the backside attacks to obtain the suitable attack location (Helfmeier et al., 2013). Only after polishing the silicon substrate down to the scale of dozens of micrometers can the subsequent attack steps be performed. All of the extra steps make the implementations of the backside attacks commonly more expensive than those of the frontside attacks. In addition, without the active shield, the frontside attacks will be a better choice rather than the backside attacks because of the easy implementation. Therefore, the active shield is still of great use in thwarting invasive attacks.

Active shield utilizes a complex metal mesh on the topmost metal layer to cover the whole chip. The mesh based on random Hamiltonian path is preferred because of its complex topological structure. However, the random Hamiltonian path is generated in a special way because this path must meet the process requirements. The efficiencies of the available generation algorithms are agonizingly low, which makes them only fit for module-level shield generation. Except an optimization algorithm (Xin et al., 2019), there are few available unoptimized algorithms that mention the capability of large shield generation with the area over 1×10^5 vertices. In this paper, a novel generation algorithm of random Hamiltonian path is proposed using the artificial fish-swarm random-Hamiltonian algorithm (AFSRHA). This algorithm is based on modified artificial fish-swarm algorithm. The AFSRHA has rapid convergence rate and good stability. High execution speed indicates that the AFSRHA is suitable for large shield generation.

The rest of this paper is structured as follows. Section 2 briefly gives the background. Section 3 presents the proposed algorithm in detail. Simulation results are given in Section 4. Conclusions are drawn in Section 5.

2. Background

2.1. Attack methods

Though invasive attacks are sophisticated and expensive requiring advanced technical skills and equipment (Mishra et al., 2017), such attacks are still prevalent and present great threats because they can unearth critical information directly from chips, including the layouts and the stored data in memories, etc. Reverse engineering, microprobing and FIB are three main methods of invasive attacks. Reverse engineering reconstructs the netlist of a circuit by means of layer-by-layer analyses. Attackers are clear about the layout and function of the circuit, so they can do whatever they want to the circuit. Therefore, there are few effective countermeasures to resist reverse engineering at present. Microprobing attack can be utilized to read data on buses or inject faults into internal nets (Mishra et al., 2017). Through the FIB workstation, attackers can cut original tracks and deposit new tracks in a metal layer of a chip (Anderson and Kuhn, 1996). They can also build through-holes to connect nets in the lower layers of the chip. Microprobing and FIB attacks are often combined to form a more effective attack method. Attackers employ the FIB workstation to draw artificial pads connecting with the inner tracks, and then extract information through these pads by microprobing.

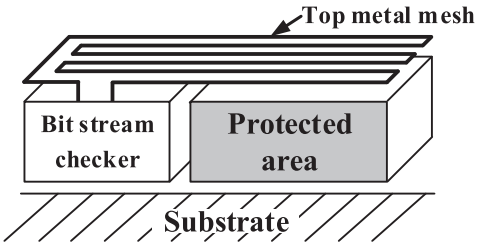


Fig. 1 – Structure of an active shield.

2.2. Detection countermeasures

Several countermeasures against microprobing and FIB attacks have been proposed, which can be divided into two categories. One is the detection of the physical changes during the attack process, such as resistance, capacitance and quantity of electric charge. Salvador Manich presents a design to detect the attack of microprobing (Manich et al., 2012). This detector is based on ring oscillators which monitor the changes of capacitance on data buses. However, this detector needs special time to perform a check, which may influence the data transmission on buses. Clemens Helfmeier proposes a FIB attack detector which checks the variation of charges by a special antenna during the FIB circuit modification (Helfmeier et al., 2012). Nevertheless, this detector can be easily disturbed when the electromagnetic field of environment is strong and unstable.

The other category is barriers. A barrier covers the whole chip and stops access to the inner circuits. Attackers cannot obtain layout information by optical methods. Invasive attacks can be detected by checking if the barrier is modified. Barriers fall into two categories, the passive shield and the active shield. Passive shield utilizes an analogue detector. For instance, the parasitic resistance of a passive shield can serve as a signature. In general, passive shields have simple geometry structures and must tolerate variations in process. There are few studies on passive shield. By contrast, active shield is a research hotspot of the barriers. As is shown in Fig. 1, an active shield is made up of a metal mesh and a bit stream checker. The metal mesh is a mesh of dense metal wires on the top metal layer of a chip (Ngo et al., 2017). The bit stream checker is utilized to check whether the mesh is modified. Compared with passive shields, active shields have more complicated geometry structures. In addition, the detectors of active shields have strong robustness and good tolerance of process deviation. Therefore, active shields are used more wildly.

2.3. Geometry structures

Rerouting attack is the main attack aiming at active shield. Attackers cut off the original metal mesh of an active shield and make new connections through FIB workstation. The new connections result in the invalidation of protection provided by the active shield. The process of rerouting attack is illustrated in Fig. 2.

Please cite this article as: Ruishan Xin, Yidong Yuan and Jiaji He et al., Random active shield generation based on modified artificial fish-swarm algorithm, Computers & Security, https://doi.org/10.1016/j.cose.2019.06.006

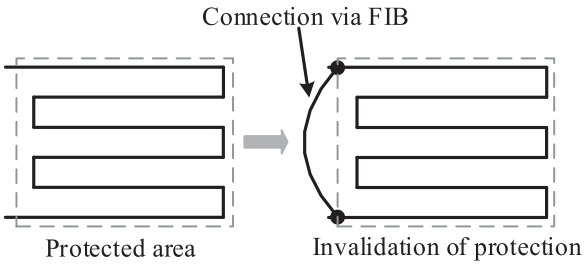


Fig. 2 – Process of rerouting attack.

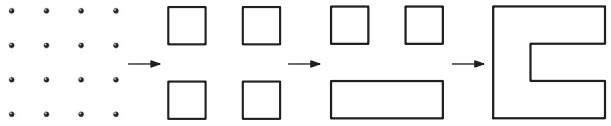


Fig. 3 – Process of the CMA.

Before implementing such an attack, attackers need to recognize the metal mesh as a precondition (Briais et al., 2012b). Therefore, the geometry of the mesh is the key part of an active shield. Complexity is regarded as a measurement of difficulty in recognizing a geometry. There are a variety of geometries available for the metal mesh. Regular geometries, such as shake and spiral, have low complexity. Thus, the rerouting attacks on these shields are implemented nearly without any hindrance. Some commercial chips, such as Infineon SLE66 and STMicroelectronics ST16 (Ray, 2009), have already employed regular shields. Unfortunately, their geometries are so simple that they have been already conquered by the researcher in Tarnovsky (2008). Therefore, a complicated geometry is necessary for an active shield. Among the available geometries, random Hamiltonian path (Briais et al., 2012a; 2012b) is outstanding. This geometry is disordered and has high complexity because its generation process is based on random choices, which makes the geometry difficult to recognize. The active shield based on random Hamiltonian path greatly raises the cost of rerouting attack and improves the security level. However, if reverse engineering is implemented on a chip with such an active shield, the geometry and connections can be extracted easily. Thus, the active shield based on random Hamiltonian path does not protect against reverse-engineering attack.

2.4. *Common problem of available algorithms*

Among all the geometry structures of the active shield, random Hamiltonian path is the most sophisticated geometry admittedly (Briais et al., 2012a; 2012b). The generation process of the random Hamiltonian path is so complex that only a few generation algorithms have been proposed. However, these algorithms have a common problem that their efficiencies are low. Cycle Merging algorithm (CMA) is a representative among these algorithms (Briais et al., 2012a). Though this algorithm can generate Hamiltonian paths with good randomness, its execution speed drops a lot when the shield area is large.

A simple example of the CMA process is shown in Fig. 3. The CMA starts with an array of vertices which have m rows and n columns. The parameters m and n are both even. Then, all vertices are sorted into an array of squares, each of which is formed by four neighboring vertices. Each square is regarded as a small independent cycle. The CMA has two main steps, the choosing step and the merging step. In the choosing step, two cycles are randomly selected from among the squares or the already-merged cycles. If they have a couple of parallel sides, they will be merged together to form a big cycle in the merging step. Otherwise, the choosing step will be executed afresh. The algorithm ends when there is only one cycle left.

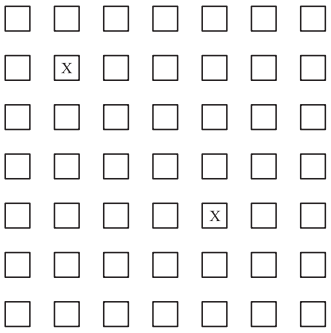


Fig. 4 – Abortive attempt at merging.

The efficiency of the CMA drops during the choosing step. When the number of total cycles is small, the two selected cycles may have parallel sides to a great extent. With the increasing number of total cycles, the probability that the two selected cycles have parallel sides decreases. During large shield generation, most of the execution time is occupied by the choosing step to find such two cycles, resulting in the low efficiency. As shown in Fig. 4, the two squares with *X* are the cycles selected randomly in the choosing step. They have no parallel sides, resulting in an abortive attempt at merging. When the number of optional cycles is large, the abortive merging attempts will occur frequently.

The complexity of the random Hamiltonian path is guaranteed by the choosing step. To improve the efficiency, a new method of the choosing step should be proposed. In this paper, we propose the following generation strategy. Before choosing, all optional cycles are divided into two categories: the well-merged cycles and the badly-merged cycles. The well-merged cycles have parallel sides with the main cycle for certain. Only the well-merged cycles can be randomly selected, thus making each selection effective. Based on the generation strategy, the AFSRHA is proposed.

3. AFSRHA design

3.1. *Formulation of AFSRHA*

First proposed in 2002 (Li et al., 2002), artificial fish-swarm algorithm (AFSA) is one of the best swarm intelligence algorithms. This algorithm is inspired by the collective action of fish and their diverse social behaviors (Cheng et al., 2016; Guan and Yin, 2012; Neshat et al., 2014; Wang et al., 2005). This algorithm is based on neighborhood search. Each individual artificial fish executes special behaviors according to its current

Please cite this article as: Ruishan Xin, Yidong Yuan and Jiaji He et al., Random active shield generation based on modified artificial fish-swarm algorithm, Computers & Security, https://doi.org/10.1016/j.cose.2019.06.006

state and its surrounding environment. This process is similar to the generation process of Hamiltonian path. Each cycle searches its surroundings to find whether there are cycles which can be merged. Thus, the rationale of the artificial fish-swarm algorithm is conformable with the generation of Hamiltonian path. Besides, the solving process of the artificial fish-swarm algorithm keeps good randomness, which is significant for the random generation process of Hamiltonian path. Therefore, the artificial fish-swarm algorithm is especially suitable for optimizing the generation of Hamiltonian path.

3.1.1. Basic AFSA

Artificial fish (AF), which is a fictitious entity of a true fish, carries special data and implements a series of fixed behaviors. The state vector of individual AF is expressed as $X = (x_1, x_2, \ldots, x_n)$, in which $x_i (i = 1, 2, \ldots, n)$ are the optimization variables by desire (Cheng et al., 2016). The food concentration of AF at current position is $Y = f(X)$. The distance between two AF is defined as $d_{i,j} = \|X_i - X_j\|$. *Visual* is the visual distance, which stands for the distance perception of AF. *Step* is the moving step of AF. δ is the crowding factor, which represents the crowding level of fish swarm. The functions of standard AFSA are preying behavior, swarming behavior and following behavior.

Preying behavior. The fish tend to the food. The current position of AF is X_i. The AF randomly selects the position X_j in its visual distance. If $Y_i < Y_j$, the AF moves a step in the direction. Otherwise, another position will be selected randomly. If the forward condition is not satisfied after multiple attempts, the AF will go forward a step randomly.

Swarming behavior. The fish tend to gather in groups while moving. The current position of AF is X_i. By exploring the surroundings ($d_{i,j} < Visual$), the number of neighborhood partners n_f and the center position X_c are obtained. N is the total fish number. If $Y_i < Y_c$ and $\frac{n_f}{N} < \delta$, which means the center has more food and is not crowded, the AF moves a step to the center. Otherwise, it performs preying behavior.

Following behavior. When an AF finds food, the adjacent fish will trail and reach the food. X_i is the current position of AF and X_j is the position of the partner which has the greatest Y_j in the neighborhood ($d_{i,j} < Visual$). If $Y_i < Y_j$ and $\frac{n_f}{N} < \delta$, which means the partner has more food and its surroundings are not crowded, the AF moves a step to the partner. Otherwise, it performs preying behavior.

3.1.2. AFSRHA

The AFSRHA draws on the experience of the rationale of the standard artificial fish-swarm algorithm. To fit the generation process of the Hamiltonian path, all concepts and behaviors of the standard artificial fish-swarm algorithm are modified properly. Here, some basic concepts of the modified artificial fish-swarm algorithm are briefly described.

In the AFSRHA, $AF_{i,j}$ stands for an artificial fish whose position is at the coordinate point (i, j). Artificial fish are distributed evenly throughout the whole region. An artificial fish occupies 1×1 in area. The distance between two adjacent artificial fish is defined as 2. *Visual* and *Step* keep 3 and 2 respectively, making an artificial fish only explore small neighborhood and obtain information from adjacent artificial fish. We assume that all positions have enough space to accommodate the fish swarms so that the crowding factor δ is useless.

Vitality. Vitality stands for the activity level of an artificial fish. It is a built-in attribute of the artificial fish and represents the food concentration in the current position of the artificial fish. The vitality has three values: 0, 1 and 2. Its value is related to the distance between an artificial fish and food. As an artificial fish approaches the food, its vitality rises gradually. $Vit(AF_{i,j})$ stands for the vitality of $AF_{i,j}$. The vitality of an artificial fish is initialized to 0 and can be improved by implementing fixed behaviors.

To improve the execution efficiency of behaviors, artificial fish fall into three swarms, free-fish swarm (FFS), active-fish swarm (AFS) and central-fish swarm (CFS), according to the vitality.

Free-fish swarm. Free-fish swarm is a swarm of artificial fish whose vitalities are all 0. This swarm is the initial swarm. The artificial fish in this swarm swim in their own areas because of no food in their territories and neighborhoods. Only one artificial fish in the FFS can perform the preying behavior when food is put. The others can only perform the following behavior. The FFS is expressed as

$$FFS \Leftarrow \{AF | Vit(AF) = 0\}. \tag{1}$$

Active-fish swarm. The vitalities of artificial fish in the AFS are all 1 because food is near to this swarm. The artificial fish in the FFS will join the AFS after implementing the following behavior. The AFS is the set of the well-merged cycles. The artificial fish in the AFS can only perform the swarming behavior. The AFS is expressed as

$$AFS \Leftarrow \{AF | Vit(AF) = 1\}. \tag{2}$$

Central-fish swarm. The artificial fish in the CFS take food, so their vitalities are up to 2. The artificial fish in the AFS will join the CFS after executing the swarming behavior. The cycle formed by the CFS is the main cycle which other cycles are merged with. The artificial fish in the CFS are in the best positions, so they dispense with any behavior. The CFS is expressed as

$$CFS \Leftarrow \{AF | Vit(AF) = 2\}. \tag{3}$$

Bulletin board. The number of the artificial fish in the CFS is recorded in the bulletin board. The value of the bulletin board shows the schedule of the generation process. The algorithm finishes when the bulletin board reaches a certain value.

In standard artificial fish-swarm algorithm, the artificial fish have three main behaviors based on the characteristic that fish usually seek and stay in the place with plenty of food. In the AFSRHA, the three behaviors are appropriately modified to fit the special generation process.

Preying behavior. At the initial time, only one position is randomly selected to put food. The artificial fish at this position finds the food and takes food. Thus, its vitality increases. This artificial fish is the only one which can execute the preying behavior and becomes the center of the whole swarms as well as the origin of the CFS. The preying behavior can be described

Please cite this article as: Ruishan Xin, Yidong Yuan and Jiaji He et al., Random active shield generation based on modified artificial fish-swarm algorithm, Computers & Security, https://doi.org/10.1016/j.cose.2019.06.006

as the following procedures

$$\begin{cases} Vit(AF_{i,j}) & \Leftarrow 2 \\ CFS & \Leftarrow \left\{AF_{i,j}\right\}. \end{cases} \tag{4}$$

Swarming behavior. The center is the position of the CFS. It has the greatest food concentration. Thus, the artificial fish near the CFS instinctively tend to join the CFS to obtain food. All of these artificial fish, which surround the CFS, form the swarm AFS. The artificial fish in the AFS have equal opportunities to join the CFS, but during a cycle of the swarming behavior, only one artificial fish in the AFS can join successfully. The others keep in current status. This lucky artificial fish is selected randomly from the AFS. Its vitality rises and then it joins the CFS. This behavior can be described as the following procedures

$$\begin{cases} AF_{i,j} & \Leftarrow rand(AFS) \\ Vit(AF_{i,j}) & \Leftarrow 2 \\ CFS & \Leftarrow CFS \cup \left\{AF_{i,j}\right\}, \end{cases} \tag{5}$$

where $rand(AFS)$ stands for the artificial fish randomly selected from the AFS.

Following behavior. This behavior is exclusive to the artificial fish in the FFS. If an artificial fish $AF_{i,j}$ joins the CFS, the adjacent artificial fish $AF_{i-2,j}$, $AF_{i,j-2}$, $AF_{i+2,j}$ and $AF_{i,j+2}$ in the FFS will be enlivened by the $AF_{i,j}$, making their vitalities rise. If these artificial fish belong to the AFS or the CFS, their vitalities will be unaffected. For the $AF_{i-2,j}$, if it belongs to the FFS, this behavior can be described as the following procedures

$$\begin{cases} Vit(AF_{i-2,j}) & \Leftarrow 1 \\ AFS & \Leftarrow AFS \cup \left\{AF_{i-2,j}\right\}. \end{cases} \tag{6}$$

3.2. Procedures of AFSRHA

The goal of the AFSRHA is to find a path to visit all the vertices which are generated at the beginning of the generation process. Each vertex is permitted to be visited only once, which ensures that the path forms a closed loop without overlaps. This closed loop is a Hamiltonian path. If the visit sequence is random, the geometry of the Hamiltonian path is also random. The complete algorithm is detailed in Algorithm 1. Here, $AF^{v}_{i,j}$ stands for the artificial fish at the position (i, j) with the vitality v. The function $rand(A)$ stands for the element randomly selected from the set A. $Path(A)$ is the path formed by the set A. $Board$ is the value of the bulletin board. $Card(A)$ stands for the number of elements in the set A. Num is a parameter related to $Card(FFS)$. Symbol "$\rightsquigarrow$" stands for the merging procedure. "$Path(A) \rightsquigarrow Path(B)$" stands for the path formed by merging the path A and B.

The flow chart of the AFSRHA is shown in Fig. 5. At the beginning, the initialization is implemented, which is realized from Step 1 to Step 5. This procedure includes the initialization of the FFS, the CFS and the bulletin board. Afterwards, the main procedures are implemented. The following behavior is realized from Step 7 to Step 22. This procedure updates the elements of the AFS and makes sure that all the cycles in the AFS can be merged with the main cycle formed by the CFS,

Algorithm 1 AFSRHA.

Input: width W and height H ($W, H \in 2\mathbb{N}$);
Output: final Hamiltonian path $Path(F)$;
1: $FFS \Leftarrow \left\{ AF^{0}_{i,j} \middle| 1 \le i \le W, 1 \le j \le H, \tilde{i}, j \in 2\mathbb{N}+1 \right\}$
2: $AF^{0}_{m,n} \Leftarrow rand(FFS)$
3: set $Vit(AF_{m,n}) \Leftarrow 2$, so $CFS \Leftarrow \{AF^{2}_{m,n}\}$
4: set $Path(CFS) \Leftarrow Path(AF^{2}_{m,n})$
5: set $Board \Leftarrow 1$ and $Num \Leftarrow Card(FFS)$
6: **while** $Board \le Num$ **do**
7: **if** $AF_{m-2,n} \in FFS$ **then**
8: $Vit(AF_{m-2,n}) \Leftarrow 1$
9: $AFS \Leftarrow AFS \cup \left\{AF^{1}_{m-2,n}\right\}$
10: **end if**
11: **if** $AF_{m,n-2} \in FFS$ **then**
12: $Vit(AF_{m,n-2}) \Leftarrow 1$
13: $AFS \Leftarrow AFS \cup \left\{AF^{1}_{m,n-2}\right\}$
14: **end if**
15: **if** $AF_{m+2,n} \in FFS$ **then**
16: $Vit(AF_{m+2,n}) \Leftarrow 1$
17: $AFS \Leftarrow AFS \cup \left\{AF^{1}_{m+2,n}\right\}$
18: **end if**
19: **if** $AF_{m,n+2} \in FFS$ **then**
20: $Vit(AF_{m,n+2}) \Leftarrow 1$
21: $AFS \Leftarrow AFS \cup \left\{AF^{1}_{m,n+2}\right\}$
22: **end if**
23: $AF^{1}_{m,n} \Leftarrow rand(AFS)$
24: $Vit(AF_{m,n}) \Leftarrow 2$
25: $CFS \Leftarrow CFS \cup \{AF^{2}_{m,n}\}$
26: $Path(CFS) \Leftarrow Path(CFS) \rightsquigarrow Path(AF^{2}_{m,n})$
27: $Board \Leftarrow Card(CFS)$
28: **end while**
29: $Path(F) \Leftarrow Path(CFS)$

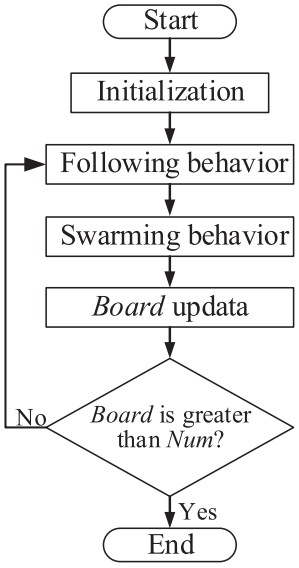


Fig. 5 – Flow chart of the AFSRHA.

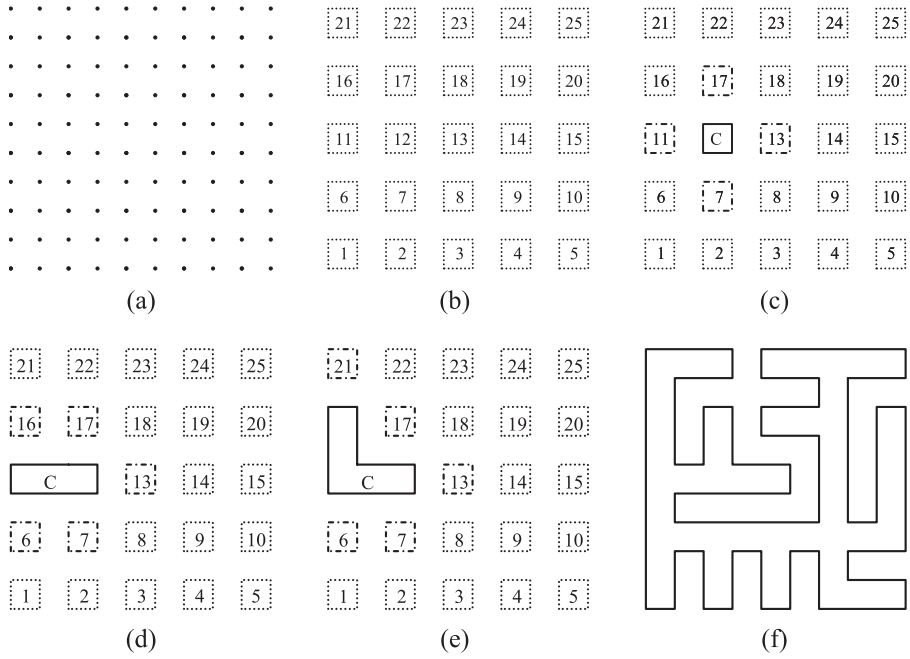


Fig. 6 – Example of the AFSRHA procedures: (a) an initialized array of vertices; (b) FFS initialization; (c) the following behaviors of the $AF_{1,5}$, $AF_{3,3}$, $AF_{5,5}$ and $AF_{3,7}$; (d) the swarming behavior of the $AF_{1,5}$; (e) the swarming behavior of the $AF_{1,7}$; (f) the final Hamilton path.

which improves the efficiency greatly. The next procedure is the swarming behavior, which is realized from Step 23 to Step 26. In this procedure, a cycle is randomly selected from the AFS and then joins the CFS. Then, *Board*, the value of the bulletin board, is updated in Step 27. If *Board* is equal or less than *Num*, the three main procedures are implemented again. Otherwise, the generation ends. An example of the AFSRHA procedures is illustrated in Fig. 6, where the artificial fish in the FFS, AFS and CFS are shown as the squares with the dot line, the squares with the dot-dash line, and the irregular path with the solid line, respectively.

The detailed process of the AFSRHA is described here. The inputs of the AFSRHA are the width *W* and height *H* of the needed Hamiltonian path. The *W* and *H* are both even. The output is the final Hamiltonian path. Step 1 to Step 5 are the initialization procedures. In Step 1, the FFS is initialized. An array of the vertices with *W* columns and *H* rows are generated, as shown in Fig. 6 (a). Then, every four adjacent vertices form a square, as shown in Fig. 6 (b). Each square stands for an artificial fish whose vitality is 0. The coordinates of the lower-left vertex of a square stand for the position of this artificial fish. The whole artificial fish form the swarm FFS jointly. There are 25 artificial fish in Fig. 6 (b), all of which are numbered according to their positions. The reference point is the lower-left vertex of the No. 1 artificial fish, which is at the position (1,1). In Step 2, a special artificial fish is randomly selected from the FFS. Then, food is put at the position of this artificial fish. We assume that its position is (m, n). The $AF^0_{m,n}$ finds food and executes the preying behavior. Therefore, in Step 3, the vitality of the $AF_{m,n}$ rises to 2 and this artificial fish becomes the initial element of the CFS. In Step 4, the path formed by the CFS is initialized to the path of $AF^2_{m,n}$. Shown as a solid-line square in Fig. 6 (c), the $AF_{3,5}$ is the special artificial fish which is marked *C*. The path *C* is the initial path formed by the CFS. In Step 5, the value of the bulletin board is initialized to 1 because there is only one element in the CFS. The parameter *Num* is initialized to the number of elements in the FFS. There are 24 artificial fish in the FFS, so *Num* is initialized to 24. *Num* determines the upper limit of the bulletin board. At this point, the initialization finishes.

Step 6 to Step 28 are the main procedures which perform the three functions, including the following behavior, the swarming behavior and the bulletin board update. If *Board* is equal or less than *Num*, the three functions are executed circularly in order. Step 7 to Step 22 implement the following behavior function. The artificial fish around the $AF_{m,n}$ are checked in proper order to estimate whether they are influenced. If these artificial fish belong to the FFS, they will be enlivened, making their vitalities rise to 1. Then, these artificial fish join the AFS. In Fig. 6 (c), the $AF_{3,5}$ joins the CFS newly. Therefore, the $AF_{1,5}$, $AF_{3,3}$, $AF_{5,5}$ and $AF_{3,7}$ are checked. Because they all belong to the FFS, they are influenced and then join the AFS. Step 23 to Step 26 perform the swarming behavior function. The artificial fish in the AFS are very close to the CFS. Thus, each artificial fish in the AFS can join the CFS easily. A new artificial fish $AF^1_{m,n}$ is randomly selected from the AFS in Step 23. Due to the swarming behavior, the vitality of this artificial fish rises to 2 in Step 24. Then, this artificial fish joins the CFS in Step 25. Because of the element change of the CFS, the path will also be updated. Therefore, in Step 26, the path formed by the CFS is merged with the path $AF^2_{m,n}$ to get a new path. In Fig. 6 (d), the $AF_{1,5}$ is randomly selected to join the CFS. The path $AF^2_{1,5}$ and the path *C* are merged. The new path is also marked *C*. In Step 27, the bulletin board is updated. The value of *Board*

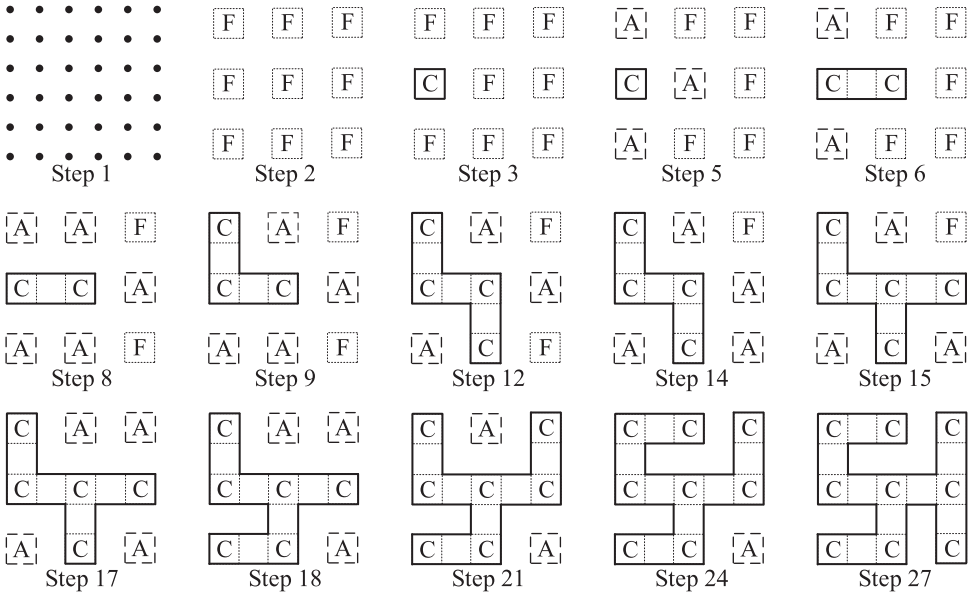


Fig. 7 – Example of a complete generation process.

is changed to the number of elements in the CFS. Then, the process jumps to Step 6. If *Board* is equal or smaller than *Num*, there are still some artificial fish which have not yet joined the CFS. The process continues and the three functions are executed in order again. Otherwise, the process jumps to Step 29. The path formed by the CFS is the final Hamiltonian path and the process finishes. In Fig. 6 (d), *Board* is updated to 2. Because this value is smaller than 24, the three functions are implemented again. In the following behavior, the $AF_{1,3}$ and $AF_{1,7}$ are influenced and join the AFS. In Fig. 6 (e), $AF_{1,7}$ is selected randomly and executes the swarming behavior. The path $AF_{1,7}$ is merged with the path *C*. Then, *Board* is updated to 3. The three functions are executed repeatedly until *Board* reaches 25. Then, all the artificial fish join the CFS. There is only one closed path in Fig. 6 (f), which is the final path we need. In the AFSRHA, thanks to the following behavior, each random selection leads to a successful merging procedure, thus improving the efficiency of this algorithm.

An example of a complete generation process is illustrated in Fig. 7. In this figure, the artificial fish marked *F*, *A* and *C* belong to the FFS, AFS and CFS, respectively. Here, $\divideontimes PB(AF_{i,j})$ stands for that $AF_{i,j}$ performs the preying behavior and becomes the origin of the CFS. $\divideontimes SB(AF_{i,j})$ stands for that $AF_{i,j}$ is randomly selected to perform the swarming behavior and joins the CFS. $\divideontimes FB(AF_{i,j})$ stands for that $AF_{i,j}$ performs the following behavior and joins the AFS. $\divideontimes FB(none)$ stands for that none of artificial fish can perform the following behavior. The detailed implementation steps are as follows.

Step1: Initialization of vertices.
Step2: Initialization of the FFS.
Step3: The position (1, 3) is randomly selected to put food. $\divideontimes PB(AF_{1,3})$.
Step4: *Board*⇐1.
Step5: $\divideontimes FB(AF_{1,1}, AF_{1,5}, AF_{3,3})$.
Step6: $\divideontimes SB(AF_{3,3})$.
Step7: *Board*⇐2. *Board* < 8, so the process continues.
Step8: $\divideontimes FB(AF_{3,1}, AF_{3,5}, AF_{5,3})$.
Step9: $\divideontimes SB(AF_{1,5})$.
Step10: *Board*⇐3. *Board* < 8, so the process continues.
Step11: None of AF in the FFS are enlivened by $AF_{1,5}$. $\divideontimes FB(none)$.
Step12: $\divideontimes SB(AF_{3,1})$.
Step13: *Board*⇐4. *Board* < 8, so the process continues.
Step14: $\divideontimes FB(AF_{5,1})$.
Step15: $\divideontimes SB(AF_{5,3})$.
Step16: *Board*⇐5. *Board* < 8, so the process continues.
Step17: $\divideontimes FB(AF_{5,5})$.
Step18: $\divideontimes SB(AF_{1,1})$.
Step19: *Board*⇐6. *Board* < 8, so the process continues.
Step20: None of AF in the FFS are enlivened by $AF_{1,1}$. $\divideontimes FB(none)$.
Step21: $\divideontimes SB(AF_{5,5})$.
Step22: *Board*⇐7. *Board* < 8, so the process continues.
Step23: None of AF in the FFS are enlivened by $AF_{5,5}$. $\divideontimes FB(none)$.
Step24: $\divideontimes SB(AF_{3,5})$.
Step25: *Board*⇐8. *Board* = 8, so the process continues.
Step26: None of AF in the FFS are enlivened by $AF_{3,5}$. $\divideontimes FB(none)$.
Step27: $\divideontimes SB(AF_{5,1})$.
Step28: *Board*⇐9. *Board* > 8, so the process finishes.

3.3. *Probability analysis*

The efficiency of a random Hamiltonian path generation algorithm is greatly limited by the choosing procedure. The two selected cycles can be merged only when they have parallel sides. Therefore, the probability that the two selected cycles have parallel sides mainly determines the efficiency of the algorithm. This probability is also called the parallel probability (PP).

Please cite this article as: Ruishan Xin, Yidong Yuan and Jiaji He et al., Random active shield generation based on modified artificial fish-swarm algorithm, Computers & Security, https://doi.org/10.1016/j.cose.2019.06.006

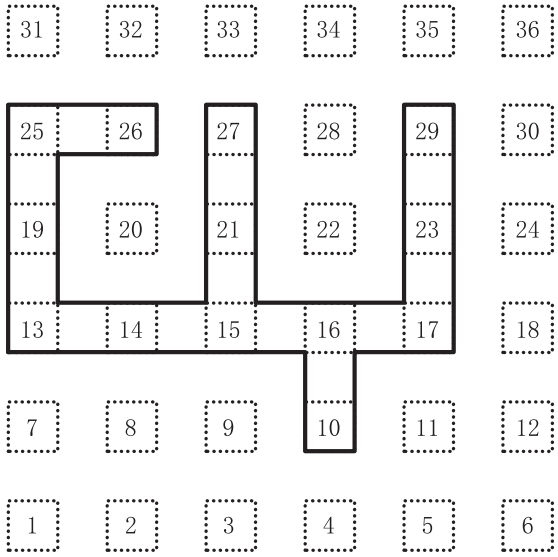


Fig. 8 – Example of a certain state during generation.

The parallel probabilities of the CMA and the AFSRHA generations are analyzed separately in this section. Fig. 8 is an example of a certain state during generation. Here, $AF_{i,j}$ still stands for the square cycle at the position (i, j). The independent cycles shown as the dot squares form the set of the independent-fish swarm (IFS). CFS still stands for the set of the already-merged cycles. The path formed by the CFS is shown as the solid-line cycle. The already-merged cycles are also shown as the dot squares inside of the solid-line cycle for convenient analysis.

The PP in a certain state of the CMA generation is analyzed first. The width and height of the target path are W and H, respectively, which are both even. Thus, the total number of the initial squares is

$$n = \frac{W \times H}{4}. \tag{7}$$

Here we assume that in this state m squares have been merged. Thus, the number of the elements in the CFS is

$$Card(CFS) = m. \tag{8}$$

The couples of parallel sides can be divided into two categories. In category 1, the two sides belong to an element in the CFS and an element in the IFS, respectively. In category 2, the two sides both belong to the elements in the IFS.

In category 1, the number of the couples of the sides which can be utilized to merge is calculated as

$$Num(C) = 4 \times Card(CFS) - 2 \times [Card(CFS) - 1] - g - k. \tag{9}$$

The first item of the Eq. (9) is the total number of the sides of the already-merged squares, each of which has four sides. If two cycles are merged, the number of the sides will decrease by 2. The second item stands for the reduced number of the sides because of merging. The parameter g stands for the number of the invalid sides which are at the edge of the whole pattern and belong to the elements in the CFS. In Fig. 8, the left sides of the already-merged cycles $AF_{0,4}$, $AF_{0,6}$ and $AF_{0,8}$ are at the edge. Thus, the g is 3. The parameter k stands for the number of the invalid sides which form couples of parallel sides but all belong to the elements in the CFS. In Fig. 8, the already-merged cycles $AF_{2,8}$ and $AF_{4,8}$ have a couple of parallel sides. Therefore, the k is 2. Because some couples of the sides result in the merging of the same cycles, the number of the valid couples of the sides for merging is calculated as

$$Num(C1) = Num(C) - l. \tag{10}$$

The parameter l is the repetitive number of the couples of the sides resulting in the merging of the same cycles. For example, the $AF_{2,6}$ has four couples of parallel sides with the path formed by the CFS in Fig. 8, but three couples are repetitive. Therefore, the l is 8.

In category 2, the element number of the IFS is

$$Card(IFS) = n - Card(CFS). \tag{11}$$

The number of the couples of the sides for merging is calculated as

$$Num(C2) = \frac{4 \times Card(IFS) - Num(C) - p}{2}. \tag{12}$$

The parameter p stands for the number of the invalid sides which are at the edge of the whole pattern and belong to the elements in the IFS. For example, in Fig. 8, the left side and the bottom side of the $AF_{0,0}$ are both at the edge. According to the state in Fig. 8, the p is 21.

The total number of the cycles is

$$Num(total) = Card(IFS) + 1. \tag{13}$$

Therefore, the PP in a certain state of the CMA generation can be calculated as

$$PP(CMA) = \frac{C^1_{Num(C1)} + C^1_{Num(C2)}}{C^2_{Num(total)}}. \tag{14}$$

According to the state in Fig. 8, n is 36 and m is 13. From the Eq. (14), the $PP(CMA)$ of the state in Fig. 8 is approximately 0.141. If the width W increases by 2 and others remain unchanged, the $PP(CMA)$ will reduce to approximately 0.115. Therefore, if the width and height are large, the $PP(CMA)$ will be rather small.

Then, we analyze the PP in the AFSRHA generation. The FFS and AFS form the IFS jointly. From the Algorithm 1, the path formed by the CFS is one of the two selected cycles for merging. The other one is randomly selected from among the AFS. The artificial fish in the AFS are adjacent with the path formed by the CFS. Thus, the number of the elements in the AFS is equal to the number of the valid couples of the sides in category 1 in the above analysis. Therefore, the PP in a certain state of the AFSRHA generation can be calculated as

$$PP(AFSRHA) = \frac{C^1_{Num(C1)}}{C^1_{Num(C1)}} = 1. \tag{15}$$

Please cite this article as: Ruishan Xin, Yidong Yuan and Jiaji He et al., Random active shield generation based on modified artificial fish-swarm algorithm, Computers & Security, https://doi.org/10.1016/j.cose.2019.06.006

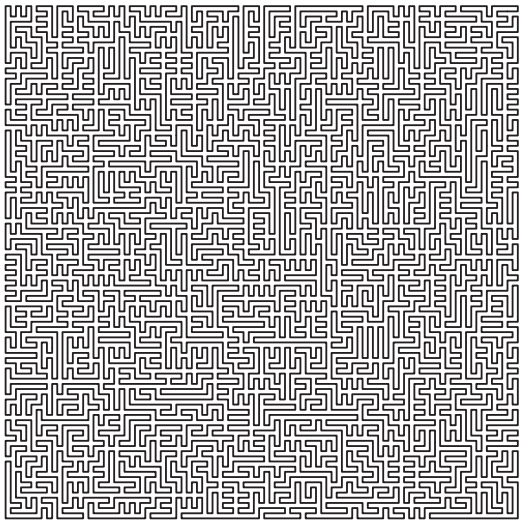


Fig. 9 – Hamiltonian path with 1 x 10^4 vertices.

No matter how large the width and height are, the *PP*(*AFSRHA*) keeps 1 unchangeably. In contrast, the *PP*(*CMA*) is less than 1 in most cases and reduces as the size of the target path increases. Through the above analysis, we prove that the AFSRHA commonly has higher PP values than the CMA does in the same state of generation. The differences of PP are accumulated during the whole generation, leading to the higher efficiency of the AFSRHA in comparison with the CMA.

Table 1 – Results of the CMA and the AFSRHA.

Size	CMA		AFSRHA		SUR
	ET/s	Entropy/bit	ET/s	Entropy/bit	
500	13.3	0.989	0.78	0.98	17.05
1000	105.4	0.992	0.81	0.995	130.12
1500	345.3	0.994	0.84	0.986	411.07
2000	738.9	0.994	0.95	0.994	777.79
2500	1431.3	0.997	1.2	0.996	1192.75
3000	2687.2	0.995	1.3	0.997	2067.08
3500	4166.4	0.996	1.4	0.998	2976
4000	5983.1	0.997	1.5	0.996	3988.73
4500	8663.5	0.996	1.74	0.995	4979.02

Table 2 – Comparison of the AFSRHA and the DCDPOA for large area.

Size	AFSRHA		DCDPOA Xin et al. (2019)	
	ET/s	Entropy/bit	ET/s	Entropy/bit
1×10^4	3.52	0.995	10.84	0.996
4×10^4	22.42	0.996	40.37	0.995
9×10^4	84.59	0.997	79.55	0.995
1×10^5	98.34	0.997	N/A	N/A
4×10^5	1081.9	0.997	N/A	N/A
9×10^5	6572.5	0.997	N/A	N/A
1×10^6	7799.7	0.997	1050.07	0.996

4. Simulation results

To testify the high efficiency of the AFSRHA, this algorithm and the CMA are both implemented on MATLAB2016a platform on Linux on a computer with an Intel Xeon X5690 CPU cadenced at 3.47 GHz. The merging procedures of the two generation algorithms are realized by the same MATLAB code. Therefore, the efficiency is mainly determined by the choosing procedure of each algorithm.

The generation time and the quality of paths are two main evaluation criteria of a generation algorithm. The generation time is related to the arithmetic speed, which is the major concern in this paper. This indicator is measured by the execution time (ET) acquired from MATLAB2016a. The quality indicates the complexity of a path. Here, we employ the entropy as the evaluation indicator of the quality Briais et al. (2012b). For a 2-dimensional shield, the entropy is calculated as below

$$E(d) = \sum_{d \in \{x,y\}} -P(d) \cdot \log_2 P(d), \tag{16}$$

where $P(d)$ is the probability for the path in the direction of d. The upper limit of the entropy is 1.000 bit for a 2-dimensional shield.

Several Hamiltonian paths with different sizes are generated by the two algorithms, respectively. A Hamiltonian path with 1×10^4 vertices is shown in Fig. 9. The simulation results are given in Table 1. The size is the number of vertices. The speedup ratio (SUR) is employed to evaluate the growth of the efficiency from the CMA to the AFSRHA. It is the ratio of the execution time of the CMA to that of the AFSRHA in the shield generation processes with the same size. The SUR is calculated as

$$SUR = ET_{CMA}/ET_{AFSRHA}, \tag{17}$$

where ET_{CMA} is the execution time of the CMA and ET_{AFSRHA} is that of the AFSRHA.

In Table 1, the paths generated by the AFSRHA have similar entropy with the paths generated by the CMA, proving that the AFSRHA also has good randomness. Therefore, the paths generated by the AFSRHA are complex enough to apply to the active shield. With the increasing number of vertices, the execution time of the CMA rises quickly, while that of the AFSRHA keeps small stable values approximately. According to the SUR, the speed of the AFSRHA is seventeen times faster than that of the CMA. Therefore, compared with the CMA, the AFSRHA has great improvements in the operational performance.

The path with 4500 vertices is only suitable for small modules. However, it takes more than 2 h for the CMA to generate such a path. Therefore, the CMA is only suitable for module-level shield generation. To verify the ability of large shield generation of the AFSRHA, some further simulations have been implemented. The results are shown in Table 2. All paths have similar high entropy, proving that large paths are also complex enough. For a standard 0.18 μm CMOS process with a thick top metal layer, the minimum width and space of the top metal

Please cite this article as: Ruishan Xin, Yidong Yuan and Jiaji He et al., Random active shield generation based on modified artificial fish-swarm algorithm, Computers & Security, https://doi.org/10.1016/j.cose.2019.06.006

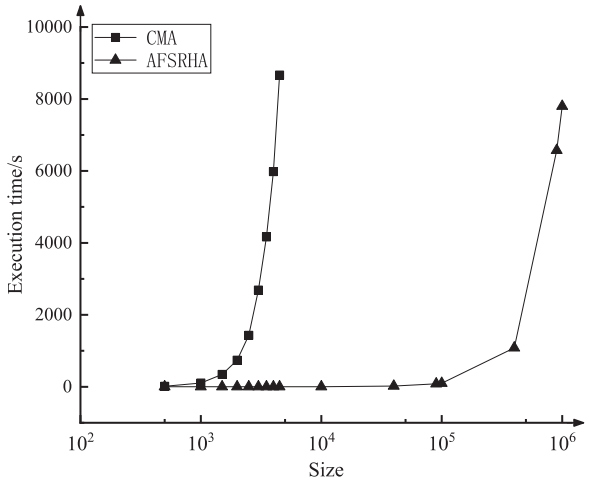


Fig. 10 – Execution time of the CMA and the AFSRHA.

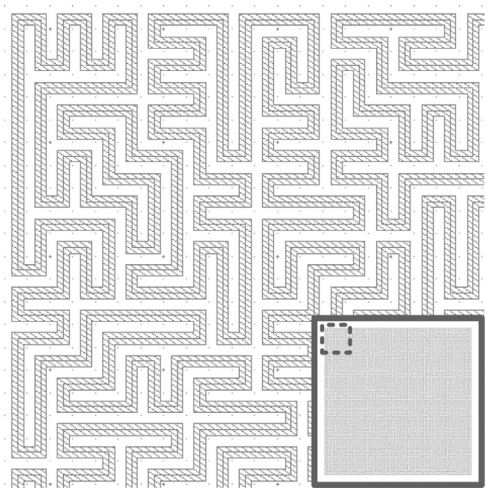


Fig. 11 – Layout of a Hamiltonian path.

are both 1.5 μm. Thus, the shield with the size of 1×10^6 vertices is enough to protect a chip with the area of $3 \times 3\,mm^2$. Therefore, the AFSRHA is capable of large shield generation.

The execution time of the AFSRHA and the CMA is shown in Fig. 10. With the growth of the size, both curves increase exponentially. The curve of the CMA shows phenomenal growth when the size exceeds 1000. However, a similar growth appears on the AFSRHA curve when the size exceeds 1×10^5. Obviously, the execution speed of the AFSRHA is faster than that of the CMA in the shield generation with the same size. In addition, the AFSRHA is capable of large shield generation for the entire chip.

The generation time of the divide-and-conquer and dynamic programming optimized algorithm (DCDPOA) in Xin et al. (2019) is also listed in Table 2 for comparison. DCDPOA is a Hamiltonian path generation algorithm based on the CMA. This algorithm applies the rationales of divide-and-conquer optimization algorithm and dynamic programming optimization algorithm to the CMA. Thus, its efficiency is higher than that of the CMA. According to Xin et al. (2019), a Hamiltonian path with 3600 vertices needs 6.31 sec to generate via the DCDPOA, while a path with 4500 vertices only needs 1.74 sec for generation by the DCDPOA. According to Table 2, the execution time of the AFSRHA is still less than the DCDPOA when the size of shield is less than 4×10^4 vertices. Thus, compared with the DCDPOA, the AFSRHA has distinct advantage in efficiency in small-to-medium shield generation. However, with the growth of size, the generation time of the AFSRHA rises faster than that of the DCDPOA. In large shield generation, the choosing step of the AFSRHA still keeps high efficiency. But the CFS variable consists of numerous elements and occupies a lot of memory. It consumes a great deal of time to visit and change this variable, resulting in the rapid growth of generation time. This problem can be solved by applying some optimization algorithms. Note that the AFSRHA is unvarnished without any optimization. Besides, in shield generation with the size less than 4×10^4 vertices, the AFSRHA is the fastest algorithm among the three algorithms, even though the DCDPOA combines with two classic optimization algorithms. Therefore, if the AFSRHA is optimized by some optimization algorithms, for example the divide-and-conquer optimization algorithm, its efficiency will be further improved.

5. Layout

The Hamiltonian path generation process of the AFSRHA is implemented on the MATLAB2016a platform. The output of the AFSRHA is a file containing the path coordinates. The values of the coordinates are all integers. Because the coordinates are normalized and merely show the relative positions of the vertices in the path. Therefore, the coordinates are independent of manufacturing process. Moreover, the generated path can be applied to any process.

The generated path can be transformed into layout through a script based on Cadence SKILL language. The parameters related to the manufacturing process are defined in this script. The script runs on the Virtuoso tool and reads the Hamiltonian path coordinates first. Here, we assume that the width and space of the top metal wires in the target process is w and s, respectively. During the transformation, if the coordinates of a vertex in the Hamiltonian path is (x, y), the transformed coordinates of the vertex in layout is $(x \times (w + s), y \times (w + s))$. Then, the script creates metal path in layout according to the transformed coordinates. By this means, the path generated by the AFSRHA is transformed into layout. Fig. 11 is a part of the layout transformed from the Hamiltonian path shown in Fig. 9.

6. Conclusion

This paper proposes a novel generation algorithm for random active shield. The AFSRHA is inspired by the rationale of the artificial fish-swarm algorithm. The AFSRHA is highly efficient

Please cite this article as: Ruishan Xin, Yidong Yuan and Jiaji He et al., Random active shield generation based on modified artificial fish-swarm algorithm, Computers & Security, https://doi.org/10.1016/j.cose.2019.06.006

because each random selection leads to a successful combination. According to the simulation results, the AFSRHA is excellent at the execution speed which is seventeen times faster than that of the CMA. Furthermore, it has the capability of large shield generation. The AFSRHA makes the active shield based on random Hamiltonian path apply to the full-chip protection, enhancing the security of integrated circuits.

Declarations of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

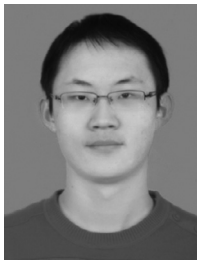
This work was supported by National Natural Science Foundation of China (Grant No. 61832018)

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cose.2019.06.006

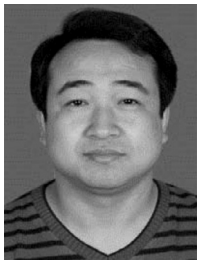
REFERENCES

Anderson R, Kuhn M. Tamper resistance-a cautionary note, 2; 1996. p. 1–11.
Beit-Grogger A., Riegebauer J.. Integrated circuit having an active shield. 2005. US Patent 6,962,294.
Boit C, Helfmeier C, Kerst U. Security risks posed by modern IC debug and diagnosis tools. In: Proceedings of the fault diagnosis and tolerance in cryptography (FDTC). IEEE; 2013. p. 3–11. doi:10.1109/FDTC.2013.13.
Briais S, Caron S, Cioranesco JM, et al. 3D hardware canaries. In: Proceedings of the international workshop on cryptographic hardware and embedded systems. Springer; 2012a. p. 1–22. doi:10.1007/978-3-642-33027-8_1.
Briais S, Cioranesco JM, Danger JL, et al. Random active shield. In: Proceedings of the fault diagnosis and tolerance in cryptography; 2012b. p. 103–13. doi:10.1109/FDTC.2012.11.
Cheng C, Li H, Bao C. Hybrid artificial fish algorithm to solve TSP problem. In: Proceedings of the sixth international Asia conference on industrial engineering and management innovation; 2016. p. 275–85. doi:10.2991/978-94-6239-145-1_27.
Guan X, Yin Y. An improved artificial fish swarm algorithm and its application, 433. Trans Tech Publications; 2012. p. 4434–8. doi:10.4028/www.scientific.net/AMR.433-440.4434.
Handschuh H, Paillier P, Stern J. Probing attacks on tamper-resistant devices. In: Proceedings of the international workshop on cryptographic hardware and embedded systems. Springer; 1999. p. 303–15. doi:10.1007/3-540-48059-5_26.
Helfmeier C, Boit C, Kerst U. On charge sensors for FIB attack detection. In: Proceedings of the hardware-oriented security and trust (HOST). IEEE; 2012. p. 128–33. doi:10.1109/HST.2012.6224332.
Helfmeier C, Nedospasov D, Tarnovsky C, et al. Breaking and entering through the silicon. In: Proceedings of the ACM SIGSAC conference on computer & communications security. ACM; 2013. p. 733–44. doi:10.1145/2508859.2516717.
Kömmerling O, Kuhn MG. Design principles for tamper-resistant smartcard processors. Smartcard 1999;99:9–20.
Li X, Shao Z, Qian J. An optimizing method based on autonomous animats: fish-swarm algorithm. Syst Eng Theory Pract 2002;22(11):32–8.
Manich S, Wamser MS, Sigl G. Detection of probing attempts in secure ICs. In: Proceedings of the hardware-oriented security and trust (HOST). IEEE; 2012. p. 134–9. doi:10.1109/HST.2012.6224333.
Mishra P, Bhunia S, Tehranipoor M. Hardware IP security and trust. Springer; 2017.
Neshat M, Sepidnam G, Sargolzaei M, et al. Artificial fish swarm algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications. Artif Intell Rev 2014;42(4):965–97. doi:10.1007/s10462-012-9342-2.
Ngo XT, Danger JL, Guilley S, et al. Cryptographically secure shield for security IPs protection. IEEE Trans Comput 2017;66(2):354–60. doi:10.1109/TC.2016.2584041.
Quadir SE, Chen J, Forte D, et al. A survey on chip to system reverse engineering. ACM J Emerg Technol Comput Syst 2016;13(1):1–34. doi:10.1145/2755563.
Ray V. Freud applications of FIB: invasive FIB attacks and countermeasures in hardware security devices. In: Proceedings of the east-coast focused ion beam user group meeting; 2009. p. 1–28.
Shi Q, Asadizanjani N, Forte D, et al. A layout-driven framework to assess vulnerability of ICs to microprobing attacks. In: Proceedings of the hardware oriented security and trust (HOST). IEEE; 2016. p. 155–60. doi:10.1109/HST.2016.7495575.
Tarnovsky C. Security failures in secure devices. Black Hat DC Presentation; 2008.
Van Tilborg HC, Jajodia S. Encyclopedia of cryptography and security. second ed. New York: Springer Science & Business Media; 2011.
Wang C, Zhou C, Ma J. An improved artificial fish-swarm algorithm and its application in feed-forward neural networks, 5; 2005. p. 2890–4. doi:10.1109/ICMLC.2005.1527436.
Weingart SH. Physical security devices for computer subsystems: a survey of attacks and defenses. In: Proceedings of the international workshop on cryptographic hardware and embedded systems. Springer; 2000. p. 302–17.
Xin R, Yuan Y, He J, Li Y, Zhao Y. High-efficient generation algorithm for large random active shield. Sci. China Inf. Sci. 2019;62(3):39108. doi:10.1007/s11432-018-9474-3.



Ruishan Xin received the B.E. degree in electronic science and technology from Tianjin University, Tianjin, China, in 2014. He is currently working toward the Ph.D. degree in the Application Specific Integrated Circuit Design Center at Tianjin University, Tianjin, China.

His research interests include mixed signal integrated circuit design and anti-attack security chip design.

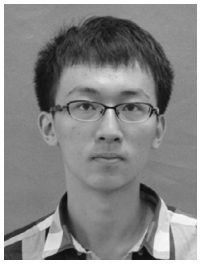


Yidong Yuan received the the B.E degree in electronic science and technology from Tianjin University in 2005, and the M.S degree in microelectronics and solid-state electronics from Tianjin University, Tianjin, China, in 2007. He is currently studying for a Ph. D. in Microelectronics.

His current research interests include IC design and security technology.

Jiaji He received the B.S. degree in electronic science and technology from Tianjin University, Tianjin, China, in 2013, and he received the M.S. degree in microelectronics from Tianjin University, Tianjin, China, in 2015.

He is currently a Ph.D. student in the School of Microelectronics, Tianjin University, China. He visited University of Florida as a visiting scholar under the guidance of Yier Jin, from 2016 to 2018. His research interests are digital circuit design and validation, and hardware security.



Shuai Zhen graduated from the school of microelectronics at Tianjin University, Tianjin, China, in 2017. He is studying for a masters degree in school of microelectronics at Tianjin University. His mainly research interests include anti-attack security chip and digital integrated circuits.

Yiqiang Zhao received the B.E. in semiconductor physics and device, the M.S. in microelectronics, and Ph.D. degrees in microelectronics and solid-state electronics from Tianjin University, Tianjin, China, in 1988, 1991 and 2006, respectively.

In 1991, he joined Jinhang Technical Physics Institute, Tianjin, China, where he is responsible for analog and mixed signal circuit design. Since 2001, he has been with School of Electronic Information Engineering, Tianjin University, where he is currently a Professor. His research interests include mixed-signal integrated circuits, security chips, photoelectric imaging systems and sensor systems.

Prof. Zhao is a senior member of Chinese Society of Astronautics.

Please cite this article as: Ruishan Xin, Yidong Yuan and Jiaji He et al., Random active shield generation based on modified artificial fish-swarm algorithm, Computers & Security, https://doi.org/10.1016/j.cose.2019.06.006