Inapplicability of Differential Fault Attacks against Cellular Automata based Lightweight Authenticated Cipher

AMBILI K N[⋆], JIMMY JOSE[†]

Department of Computer Science and Engineering,
National Institute of Technology Calicut, India

Received 31 October 2021; In final form –

Authenticated encryption (AE) schemes are a necessity to secure the physical devices connected to the Internet. Two AE schemes, TinyJambu and Elephant, are finalists of NIST lightweight cryptography competition. Another AE scheme, ACORN v3, a CAESAR competition finalist, has been shown to be particularly vulnerable against Differential Fault Attack (DFA), even more than its previous version ACORN v2. TinyJambu is also susceptible to DFA. An optimized interpolation attack has been proposed against one instance of Elephant, Delirium, recently. We propose methods to strengthen these schemes using the Cellular Automata (CA) and increase their resistance to these attacks. The Programmable Cellular Automata (PCA) 90-150 is effectively deployed to make these ciphers robust against DFA. We also provide mathematical analysis of the invigorated schemes and show that significant improvement is achieved in all the three enhanced schemes.

Key words: ACORN, TinyJambu, Differential Fault Attack, Optimized Interpolation Attack, Cellular Automata, PCA 90-150, Authenticated encryption

[⋆] email: `ambili_p180002cs@nitc.ac.in`
[†] email: `jimmy@nitc.ac.in`

1 INTRODUCTION

The rise in the use of technology in daily lives has made an increase in security a necessity. The usual strategy of ensuring confidentiality and authenticity separately may not be sufficient. In scenarios like the Internet of Things (IoT), the authenticity of information passing through various sensors and servers is very important.

The automatic triggering of sensed data from IoT devices to other network devices happen frequently. The possibilities of vulnerabilities in web applications accessing this data cannot be overlooked. The lack of proper authentication of data may facilitate data tampering. This makes simultaneous execution of confidentiality and authentication advantageous. The concept was first introduced in [16].

AE algorithms are used to achieve confidentiality and authenticity simultaneously. If a passive adversary cannot determine the content of ciphertext, the AE scheme is said to have privacy. If an active adversary cannot successfully forge a ciphertext C, a nonce N and a tag t and mislead the receiver, authenticity is guaranteed. AE schemes are modes of operation. They are algorithms built on top of primitives which are proven to be secure. These algorithms are not theoretically proven secure like the block ciphers. The security of AE modes depend on the primitives which underlie them. We briefly outline the evolution of AE schemes before describing the algorithms under consideration.

The most primitive method of AE involves independent use of algorithms for encryption and authentication. This is called the generic composition. There are three ways to achieve AE based on the order in which encryption and authentication are executed. They are Mac then Encrypt, Encrypt then Mac or Encrypt and Mac. Of these, Encrypt and Mac is proven to be secure.

There are several methods in literature which improved the generic composition. Single pass combined mode is one such method. Integrity Aware Parallelizable Mode (IAPM) discussed in [11] and developed by Jutla at IBM in 2000 is the first such approach. Offset Codebook Mode (OCB) described in Ref [12] was later introduced by Rogaway et. al. It is designed to be fully parallelizable and has a host of other improvements. IAPM and OCB are patented. Hence, two pass combined modes were developed to introduce new patent-free modes. Counter with CBC MAC (CCM) [19], Encrypt then Authenticate then Translate (EAX) [2] and Carter Wegman Counter (CWC) [8] modes belong to this category.

Many robust AE designs based on stream ciphers, block ciphers or sponge functions have been proposed in the last decade. National Institute of Standards and Technology (NIST) also selects algorithms as part of lightweight cryptography project described in [13]. The finalists of NIST lightweight cryptography competition include TinyJambu and Elephant. These have been shown to be susceptible to fault attacks. AE designs were also evaluated in the CAESAR competition [23]. ACORN v3 [22], a CAESAR finalist, is a stream cipher based AE scheme. We consider these three specific cases and propose methods to invigorate the AE schemes using Cellular Automata (CA) against fault attacks.

The fault attack is a form of side channel attack that can work successfully on physical implementations. It is a powerful tool to retrieve the secret key of many cryptographic primitives [6]. A hard fault attack on ACORN v1 and v2 in a nonce respecting scenario was proposed in [7]. The attack was based on assumption that a hard fault is injected at a certain position. The differential fault attack (DFA) under a general fault model is described in [24]. ACORN v3 is found to be vulnerable to DFA more than ACORN v2.

In the current work, the inherent properties of Cellular Automata (CA) [20] are exploited to improve the strength of ACORN, Elephant and TinyJambu. The rest of the paper is organized as follows: Section 2 provides a brief description of algorithms. Section 3 describes the fault attacks mounted on them. Section 4 describes CA and specific features of programmable cellular automata (PCA). Our enhanced AE schemes are described in section 5. Section 6 provides randomness review of PCA 90-150. The security analysis of enhanced algorithm is provided in section 7. Their strength to resist fault attacks is elaborated. Section 8 concludes the paper.

2 PRELIMINARIES

We consider ACORN v3, a CAESAR competition finalist and describe the details of the algorithm. We also consider Elephant and TinyJambu described in [13] which are finalists of NIST lightweight cryptography competition. These are found susceptible to fault attacks recently. We describe them briefly with emphasis on their vulnerabilities.

2.1 ACORN v3

ACORN v3 [22] is a lightweight authenticated cipher. It uses a 128-bit key and a 128-bit initialization vector. The state size of ACORN v3 is 293 bits denoted by S = (s_0, s_1,..., s_{292}). There are six Linear Feedback Shift Registers (LFSRs) being concatenated in ACORN-128 as shown in Fig. 1.

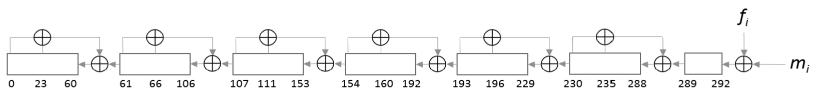

FIGURE 1: The 6 LFSRs are concatenated to represent the cipher state

Here, f_i indicates the feedback bit and m_i refers to the message bit that gets concatenated to the state at the last step of the State Update function which can be divided into 4 steps as outlined below:

1. Update the state:

$$S_{i,289} = S_{i,289} \oplus S_{i,235} \oplus S_{i,230} \tag{1}$$

$$S_{i,230} = S_{i,230} \oplus S_{i,196} \oplus S_{i,193} \tag{2}$$

$$S_{i,193} = S_{i,193} \oplus S_{i,160} \oplus S_{i,154} \tag{3}$$

$$S_{i,154} = S_{i,154} \oplus S_{i,111} \oplus S_{i,107} \tag{4}$$

$$S_{i,107} = S_{i,107} \oplus S_{i,66} \oplus S_{i,61} \tag{5}$$

$$S_{i,61} = S_{i,61} \oplus S_{i,23} \oplus S_{i,0} \tag{6}$$

2. Generate the keystream bit as

$$ks_i = S_{i,12} \oplus S_{i,154} \oplus maj(S_{i,235}, S_{i,61}, S_{i,193}) \oplus ch(S_{i,230}, S_{i,111}, S_{i,66}) \tag{7}$$

   where
   $maj(x,y,z) = (x\&y) \oplus (x\&z) \oplus (y\&z)$
   $ch(x,y,z) = (x\&y) \oplus ((\neg x)\&z)$

3. Generate the nonlinear feedback bit using control bits and feedback function. The control bits ca$_i$ and cb$_i$ are set to either 0 or 1 in different iterations specified in [22]. The feedback function (FBK) computes the nonlinear feedback bit as

$$f_i = S_{i,0} \oplus (\neg S_{i,107}) \oplus maj(S_{i,244}, S_{i,23}, S_{i,160}) \oplus (ca_i \& S_{i,196}) \oplus (cb_i \& ks_i) \tag{8}$$

4. Shift the 293-bit register with the feedback bit f_i as

for j := 0 to 291 do

$$S_{i+1,j} = S_{i,j+1} \tag{9}$$

$$S_{i+1,292} = f_i \oplus m_i \tag{10}$$

The state update is run for 1792 iterations in the initialization step after loading the key and IV into the state. Similarly, the update function is used in a different number of iterations in the associated data processing, encryption and finalization stages [22].

The modified version ACORN v3 is different from ACORN v2 in the feedback function and the filter function. This resulted in a better balance between the feedback function and the output filtering function and larger security margin against guess-and-determine attack. However, these modifications resulted in an increase in the vulnerability against the Differential Fault Attack [24]. The fault is induced randomly into the state and the attack happens in the encryption phase. The procedures of initialization, the processing of associated data and finalization are not directly involved. Hence, we briefly describe the encryption procedure alone.

The encryption happens bitwise. For each bit of the input message p_i, a key bit ks_i is generated, the XOR of these two bits generates the ciphertext bit and the State Update function described above is run. In the next iteration, the keystream bit is generated based on the new state matrix.

2.2 Elephant

Elephant is a nonce based encrypt-then-MAC construction. The mode is permutation based which is evaluated only in the forward direction. Thus, the implementation of multiple primitives or inverse is not needed. It is parallelizable by design.

There are three variants of Elephant algorithm called Dumbo, Jumbo and Delirium which use state matrix of size 160, 170 and 200 respectively. The submission claims security against fault attacks. We consider Delirium which uses a 200 bit state matrix and is found vulnerable in [25]. The authenticated encryption mode of Elephant consists of encryption and decryption.

The encryption and authentication process is shown in FIGURE 2. For the encryption part, message is padded as $M_1..M_{l_M}$ and ciphertext is $C1..C_{l_M}$. For the authentication part, nonce and associated data are padded as $A_1..A_{l_A}$. The ciphertext is padded using 1 at the right end as $C_1..C_{l_C} \longleftarrow C||1$. In encryption and decryption algorithms, key K is used with a mask.

The Keccak-f permutation discussed in [4] permutes the 200 bit state matrix. Eighteen rounds of Keccak permutation is applied on 200-bit state matrix while processing each unit of message.

The number of XOR operations that have to be performed for a state update is low since masking is achieved by adding functions rather than XOR operations exclusively. The mask can be represented as a function as given below:

$$mask_K^{a,b} = mask(K, a, b) = \phi_2^a \circ \phi_1^b \circ P(K||0^{n-k}), \tag{11}$$

where ϕ_1 and ϕ_2 are generated by LFSR, $\circ$ is the composition function and P denotes the permutation being used. The choice of masking a, b, (namely $(a, b) = (i, 1)$ in encryption layer, $(a, b) = (i, 2)$ in ciphertext authentication and $(a, b) = (i, 0)$ for associated data authentication) is such that its contribution is cancelled out while generating authentication tag. LFSR ϕ_1 is defined as F_2-linear map, where x_i's correspond to 8-bit words:

$$(x_0, ..., x_{24}) \rightarrow (x_1, ..., x_{24}, x_0 \lll 1 \oplus x_2 \lll 1 \oplus x_{13} \ll 1). \tag{12}$$

The mask uses simple LFSR defined by the primitive polynomial

$$p(x) = x^8 + x^6 + x^5 + x^4 + 1 \tag{13}$$

. The two LFSR's are chosen such that they satisfy the below condition:

$$\phi_2 = \phi_1 \oplus id \tag{14}$$

where id is the identity function.

The permutation used in Delirium is Keccak-f[200] with state matrix of size 200 bits. The above design has been submitted to the final round and achieves more efficiency and stronger authentication.

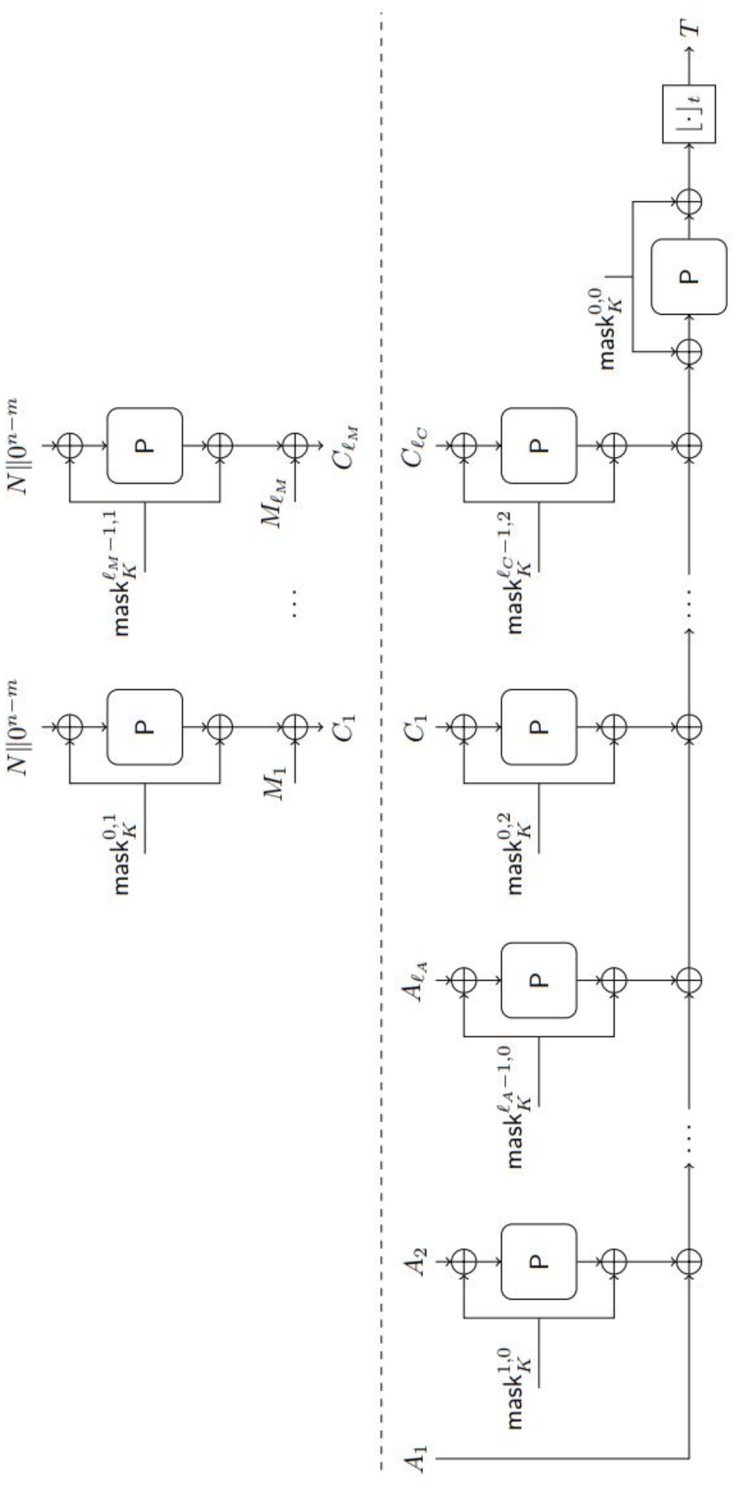

FIGURE 2: Depiction of Elephant Authenticated Encryption [14]

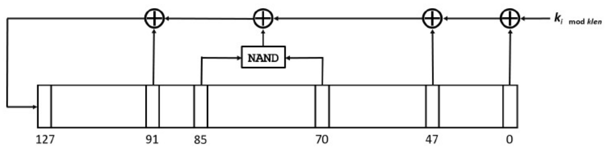

FIGURE 3: 128-bit Non-linear Feedback Shift Register in TinyJambu [15]

2.3 TinyJambu

TinyJambu is a lightweight authenticated encryption mode submitted to final round of NIST lightweight cryptography. It is the smallest block cipher AE with state size of 128 bits. The process is shown in Fig. 4.

The scheme is based on permutation using a nonlinear feedback shift register. A key bit is transferred at the end of the state matrix as shown in FIGURE 3.

The keyed permutation is run n times depending on the position of usage within the algorithm. This helps to achieve state update in nonce setup and processing of associated data. The permutation is described in Algorithm 1.

Algorithm 1 TinyJambu Permutation

```
1: procedure StateUpdate($S, K, i$)
2:     $j = 0$
3:     $feedback = s_0 \oplus s_{47} \oplus (\neg(s_{70} \wedge s_{85})) \oplus s_{91} \oplus k_{i \, mod \, klen}$
4:     while $j \neq 126$ do
5:         $s_j = s_{j+1}$
6:         $j = j + 1$
7:     $s_{127} = feedback$
```

The key setup involves randomizing the state using the keyed permutation P_{1024}. The 128-bit state is initially set to zero and then updated using the permutation. The nonce setup involves three iterations. In each iteration, the constant value 1 is XORed with the state, the keyed permutation P_{640} is used to update the state and 32 bits of nonce are XORed with the state.

The associated data is first processed block wise. Three steps are taken for each block. The bits S_{36}, S_{37}, S_{38} are XORed with $FrameBits_{0,1,2}$ which has a constant value of 3. An update of state using P_{384} is done and then the 32-bit block of associated data is added to the state by XORing with $S_{96..127}$.

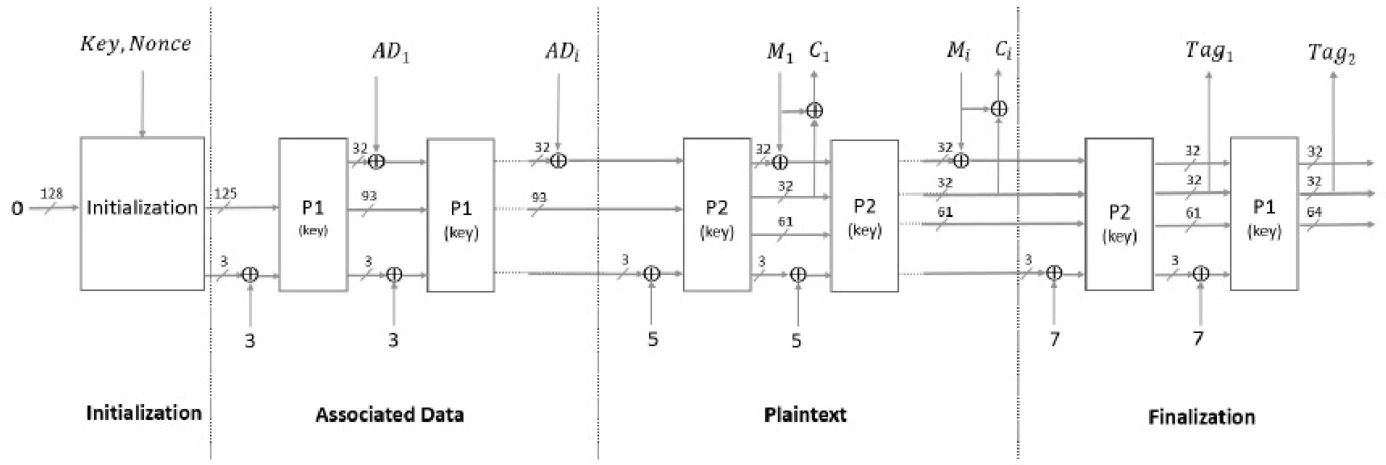

FIGURE 4: Depiction of TinyJambu Authenticated Encryption [15]

The encryption of message happens in blocks of size 32 bits. The bits S_{36}, S_{37}, S_{38} are XORed with $FrameBits_{0,1,2}$ which has a constant value of 5. The keyed permutation P_{1024} is used to update the state. The XOR of message bit and state bits from position 96 to 127 is taken for 32 bits. The bits of the ciphertext can be extracted in this stage.

After encryption, the authentication tag is generated in finalization step. The FrameBits of finalization with constant value 7 is XORed with s_{36}, s_{37}, s_{38} of the state matrix, P_{1024} is run and first 32 bits of tag is generated. Then, the FrameBits of finalization with constant value 7 is XORed with s_{36}, s_{37}, s_{38} of the state matrix, P_{640} is run and first 32 bits of tag is generated.

The decryption process involves processing full blocks of ciphertext by running the ciphertext again and verification of tag. The message is output only if tag verification is successful.

The number of times the keyed permutation is run was increased from 384 to 640 when TinyJAMBU was submitted to final round of NIST lightweight cryptography competition. This was done to increase the security and resist DFA described in [17]. We describe methods to enhance security of TinyJAMBU against DFA using CA without any increase in number of rounds.

3 ATTACKS

Benot et al describes fault attack in [3] as the attack that uses physical methods like electromagnetic radiation, laser, etc. to interfere with normal operation of cryptographic chip, forcing it to perform certain wrong operations. The crux of differential fault attack is to deduce key from the difference between incorrect and correct information.

Fault attack on cryptographic schemes is carried out in two steps. The first step involves locating the fault and the second step is equation solving. Differential fault attack (DFA) was first launched against Data Encryption Standard (DES). There are various modifications to DFA that have been successful against several types of cryptographic algorithms. In this paper, we consider DFA on ACORN v3 and TinyJambu. We also consider a different version of DFA, that is, the optimized interpolation attack on Elephant-Delirium.

3.1 DFA on ACORN v3

The attacker is assumed to have access to the physical device. It is also assumed that the attacker has two privileges. The first is that the attacker has the ability to reset the physical device with the original key. The second is that the attacker can inject a fault into the initial state randomly before the encryption procedure but cannot choose the location. For clarity, the attack [24] is briefly described below.

Let $S = (s_0, s_1, ..., s_{292})$ be the initial state of finite state register of the cipher. Let $P = (p_0, p_1, ..., p_{l-1})$ be the l bit plaintext. Let $z = (z_0, z_1, ..., z_{l-1})$ be the correct keystream and $z^i = (z^i_0, z^i_1, ..., z^i_{l-1})$ be the faulty keystream generated by the faulty initial state at random location i where $i \in [0, 292]$. We define an l-bit differential string Δz^i. The jth element of the l-bit differential string Δz^i is defined as

$$\Delta z^i_j = z_j \oplus z^i_j \tag{15}$$

This gives us a differential set corresponding to a faulty state. All possible differential sets $\Delta z^i, i \in [0, 292]$ are computed. A differential consists of a sequence of positions where their corresponding components are either 1 or non-constant functions with respect to S by omitting zero components. Due to the structure of ACORN, the first 99 positions of state matrix S can be represented as linear or quadratic functions with respect to keystream. These equations can be used to retrieve enough linear equations to recover the initial state. The experiments reveal that the length of the differential string is at most 25 and 32 random initial states are enough to conduct the fault experiments.

The attack consists of two main parts: fault locating and equation solving. If the fault locating step is achieved reliably, then the equation solving can be done by retrieving a system of equations with respect to the initial state of ACORN at which the fault was induced. At this step, fundamental methods to retrieve equations and some improvement strategies to get more linear equations are implemented. After obtaining the linear equations, the guess and determine method is used to obtain the initial state. After that, forgery attacks can be performed on the cipher.

The fault location step identifies the fault location after a fault is injected into the initial set randomly. The first algorithm of the fault locating step is explained briefly.

This algorithm returns two sets MQ_i and AQ_i. MQ_i contains positions where 1 occurs with a probability less than 1. The AQ_i set contains positions where 1 always occurs. Firstly, 32 initial states are chosen randomly. In each state, a random state bit is flipped (from 0 to 1 or from 1 to 0). Now the encryption algorithm proceeds to output the fault-induced keystream (z^i) of size l (size of plaintext and hence the ciphertext). The same state is run without inducing any fault resulting in the correct keystream output (z).

The corresponding bits of z and z^i are logical XORed and put in a set Δz. Based on values of Δz, the positions are input into the sets AQ_i and MQ_i. These sets AQ_i and MQ_i are then used for further computations. The experiments determined 103 unique sets wherein fault could be located reliably. Of the remaining non-unique sets, the key extension strategy was adopted to locate the fault. Once several faults are located, considerable number of equations can be retrieved with respect to the initial state to recover the initial state. Time complexity is bound by the number of fault experiments.

3.2 DFA on TinyJambu

Differential Fault Attack exploits the probability of occurrence of a particular difference in output given a particular input difference. It is a chosen-plaintext attack. The attacker chooses inputs and collects corresponding outputs from the random oracle. A particular pair of inputs is selected to satisfy a particular input differential. The attacker knows that for a particular input differential, a particular output differential occurs with high probability. The differentials are correctly examined to derive the key.

The paper [17] considers how the differences propagate through permutation. As described in Section 2.3, a 128-bit state is updated by the permutation. The key bit is involved in computing 127-th bit and all other bits are obtained by a shift. TinyJambu submission to NIST claims security when differential trail is calculated with minimum number of active AND gates in the simple model. The simple model considers each AND gate to be independent. The search for differential trail under four different constraints about active bit positions was the basis for security claim.

An improved model called Mixed Integer Linear Programming was introduced in [17]. The paper describes how the values of output bits and values of differences of output bits can be calculated. These may be represented as coefficient matrix and augmented coefficient matrix respectively. The number of equations in the matrices is dependent on key length in TinyJambu. These are solved to obtain the secret key bits. The security claimed by original submission was broken using experiments based on improved MILP model. The number of permutations has been increased in the final submission to withstand DFA.

3.3 Interpolation attack on Elephant-Delirium

In interpolation attack on Elephant-Delirium, the intermediate target bit a is considered whose algebraic normal form is a keyed function of ciphertext as shown in the below equations.

$$a = F_K(C) \tag{16}$$

$$F_K(C) = F_K(c_1, .., c_n) = \Sigma_{u=(u_1,..,u_n \in GF(2^n))} \alpha_u M_u \tag{17}$$

where $\alpha_u \in \{0, 1\}$.
The objective is to recover the coefficients α_u which depend on the secret key bits. These coefficients are the variables to be determined and can be recovered by solving a system of linear equations. This is a chosen-plaintext interpolation attack. The attack described in [25] is successful at the 8th round. The authors have constructed the equations of the last two rounds and obtained the algebraic normal form of intermediate target bit a which is the output of 6-round KECCAK-p.

4 CELLULAR AUTOMATA

The 1-dimensional CA structure [20] consists of a lattice of cells in a row fashion, which can take value of 0 or 1. Each cell value evolves in every time step depending on a function of values of itself and its neighbour cells. This is called a two-state three-neighbourhood CA. The next state of a cell can be represented as,

$$x_i(t+1) = f\{x_{i-1}(t), x_i(t), x_{i+1}(t)\} \tag{18}$$

where, $x_i(t)$ denotes the output state of the i^{th} cell at the t^{th} time step and f denotes the transition function of the particular cell realized with a combinational logic and is known as a rule of the CA.

When the rules used in the cells are different, the CA type is called a hybrid CA. Maximal length CA are those CA with specific rules which results in maximum cycle length. These cycle through every possible state (except all 0's) once before repeating the cycle of values. Wolfram's work in [21] proved that the patterns generated by the maximal-length CA are significantly better in randomness properties than other widely used methods like Linear Feedback Shift Registers (LFSRs).

The rules used in the design of CA in this paper are rules 90 and 150.

$$rule\,90: x_i(t+1) = x_{i+1}(t) \oplus\ x_{i-1}(t) \tag{19}$$

$$rule\,150: x_i(t+1) = x_i(t) \oplus\ x_{i+1}(t) \oplus\ x_{i-1}(t) \tag{20}$$

where $x_i(t)$ refers to the state bit of the i^{th} cell at time t. These rules which only involve the logical XORs are called linear or additive rules. CA can also be divided into types based on the neighbors of the extreme cells (the first and last cells). Null boundary refers to the extreme cell's neighbors connected to logic '0'. The CA used in this paper will use null boundary, maximal length CA with rules 90 and 150.

Programmable Cellular Automata (PCA) [10] are structures based on the elementary Cellular Automata but the rule structure is not fixed. The dynamic rule structure works based on control signal, each signifying a particular rule set on which the CA will perform iterations. Numerous hardware implementations have been made. However these control signals can be programmed in software to randomly select from a given set of rules. The PCA configuration used in this paper is based on the 90-150 configuration which signifies that each ruleset defines a hybrid null-boundary maximal length CA with rules 90 and 150.

CA$_s$ and CA$_r$ are used to implement a PCA 90-150 in [10]. The CA$_s$ function uses a 6-cell, null boundary, maximal length [5] hybrid CA. This function returns the value between 1 and 63. The CA$_r$ function is a 6-cell, maximal length, null boundary hybrid CA which selects a rule from a predefined ruleset for CA$_r$ to use. Together this results in a simulation of PCA 90-150. We may also use maximal length, null boundary hybrid CA with higher number of cells.

The PCA configuration in the current design can be represented as a polynomial of degree 6 as described in [10]. When the number of cells n is 6, the equation is

$$x(x+1)(x^4+x+1) \tag{21}$$

For n = 8, we can achieve an even higher degree of 8 with polynomial

$$x(x+1)(x^6+x+1) \tag{22}$$

Similarly for higher values of n, we obtain a higher degree expression.

In Section 7, security analysis based on [9] has been provided to show the aptness of PCA 90-150 for use in ACORN, Elephant and TinyJAMBU.

CA can be used as a good cryptographic primitive against fault attacks and in particular, the DFA. We describe the modified AE schemes in the next section.

5 MODIFIED AE SCHEMES

The improvement in randomization of affine function preserving the parallel relationship is possible with appropriate usage of CA. This section considers incorporation of CA into the algorithms ACORN v3, TinyJambu and Elephant-Delirium. These are redesigned to make use of randomness properties of CA.

5.1 ACORN v3

The modified state update function using PCA 90-150 is shown in Fig. 5. Our novel approach uses two CAs to achieve randomization. This overcomes the uniqueness of state bits used in keystream generation function by randomizing the ks_i bit, described in step 2 of section 2.1, earlier in the paper.

The modified keystream generation function uses PCA 90-150 as shown in Fig. 5. A predefined ruleset is used. CA random-rule (CA$_r$), a maximal length [1] null boundary hybrid CA which selects the rule from the ruleset for CA state (CA$_s$) to use in each call to the keystream bit generation function. CA$_s$ is a 9-cell maximal length null boundary CA using PCA 90-150 configuration. Hence, a basic simulation of programmable CA is used here.

The keystream bit generation function in ACORN v3 is

$$ks_i = S_{i,12} \oplus S_{i,154} \oplus maj(S_{i,235}, S_{i,61}, S_{i,193}) \oplus ch(S_{i,230}, S_{i,111}, S_{i,66}) \tag{23}$$

In our modified cipher the keystream bit generation function is,
CA$_r$();
ks_i=CA$_s$();

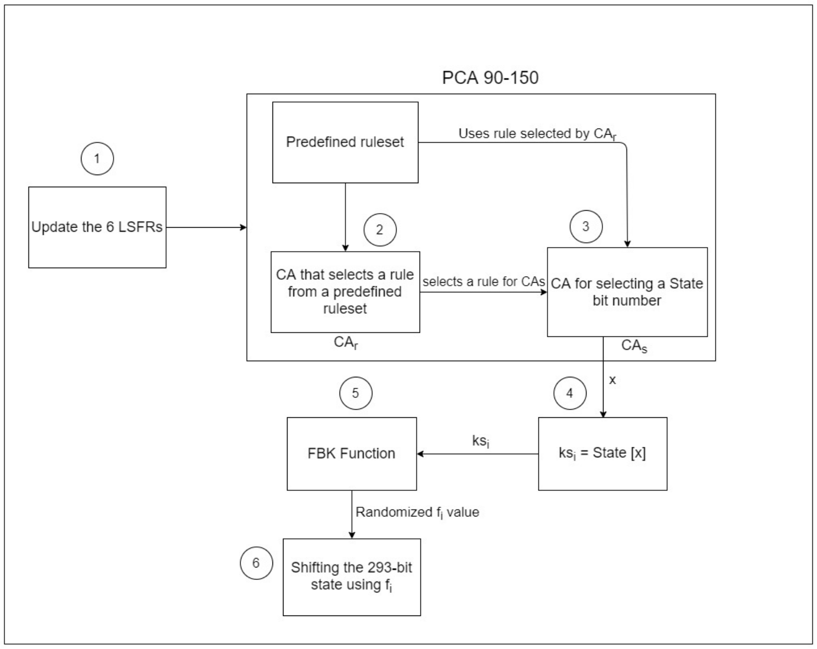
PCA 90-150
1
Update the 6 LSFRs
Predefined ruleset
Uses rule selected by CA_r
2
CA that selects a rule from a predefined ruleset
CA_r
selects a rule for CAs
3
CA for selecting a State bit number
CA_s
x
4
ks_i = State [x]
ks_i
5
FBK Function
Randomized f_i value
6
Shifting the 293-bit state using f_i

FIGURE 5: The modified State Update Function (stepwise as per numbering) in the CA enhanced ACORN

ACORN v3 uses a polynomial function of degree 2. CA configuration in the current design can be represented as a polynomial of degree 6 [10], for n (number of cells) = 6 as

$$x(x+1)(x^4+x+1) \tag{24}$$

For n = 8, we can achieve an even higher degree of 8 with polynomial

$$x(x+1)(x^6+x+1) \tag{25}$$

Similarly for higher values of n, we obtain a higher degree expression.

The fault attacks which are successful on ACORN v3 are ineffective against this modified CA based ACORN cipher. This is due to the parallel transformation of the CA that spreads the fault very quickly into the state, which makes the fault difficult to track. The key reason behind this is the unpredictability of the nonlinear feedback bit XOR with the last bit of the state during every state update.

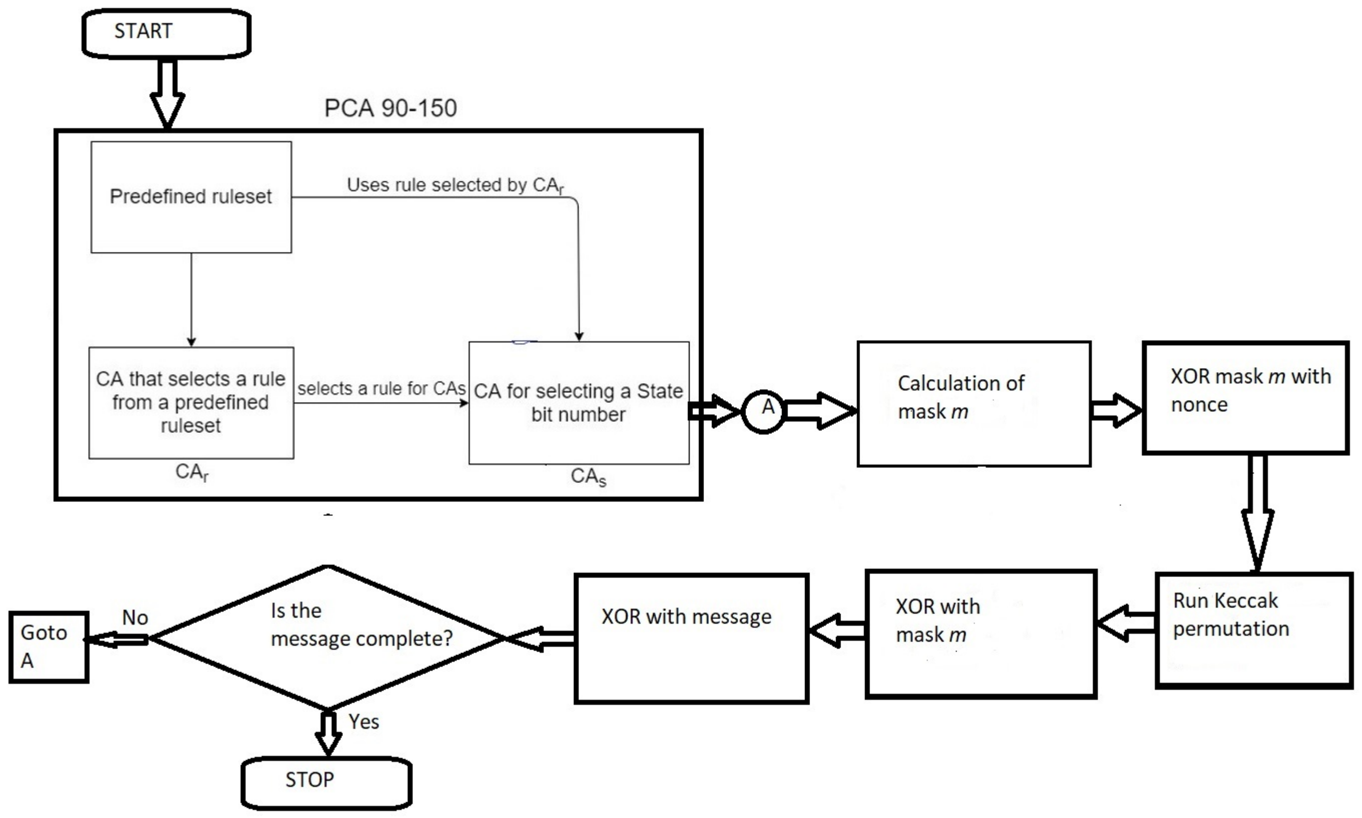

FIGURE 6: Block diagram of encryption in CA-Elephant

5.2 CA-Elephant

We have seen the importance of mask in Elephant-Delirium in section 2.2. CA can be used to generate the key K which is XORed with mask as shown in Fig. 6.

The mask function of equation 11 is redefined as given below:

$$mask_{CA()}^{a,b} = mask(CA(), a, b) = \phi_2^a \circ \phi_1^b \circ P(CA()||0^{n-k}) \quad (26)$$

where $CA()$ is used to generate pseudorandom number used as key, ϕ_1 and ϕ_2 are generated by LFSR, $\circ$ is the composition function.

The encryption in CA-Elephant is done on message broken into stream of 200 bits each. PCA 90-150 is run to generate key once. For each message stream, the mask is evaluated using this key and XOR with nonce. The output is provided to Keccak permutation. Its output is again XOR with the mask evaluated earlier using PCA output. This is followed by XOR with message stream of size 200 bits. The process is repeated for the entire message input. We use the same mask in decryption as well as authentication process.

5.3 CA-TinyJambu

The implementation of TinyJambu at NIST final submission uses $k \ mod \ len$ where len is the length of the key k for the i^{th} round. The enhanced algorithm uses PCA instead of the keystream bit in the StateUpdate function as detailed in Algorithm 2. PCA is used in the calculation of feedback bit.

The update of state is depicted in Fig. 7.

Algorithm 2 TinyJambu Permutation

```
1: procedure STATEUPDATE($S, CA(), i$)
2:     j=0
3:     $feedback = s_0 \oplus s_{47} \oplus (\neg(s_{70} \wedge s_{85})) \oplus s_{91} \oplus CA()$
4:     while $j \neq 126$ do
5:         $s_j = s_{j+1}$
6:         $j = j+1$
7:     $s_{127} = feedback$
```

For each state update process, PCA 90-150 is initially run to generate a bit. The second step is to use this bit in feedback bit generate function. The third step is to shift the 127 bits of state matrix from position 0 to 126. The feedback bit evaluated is then set as the 127-th bit. The encryption and decryption process remain the same as in the original scheme.

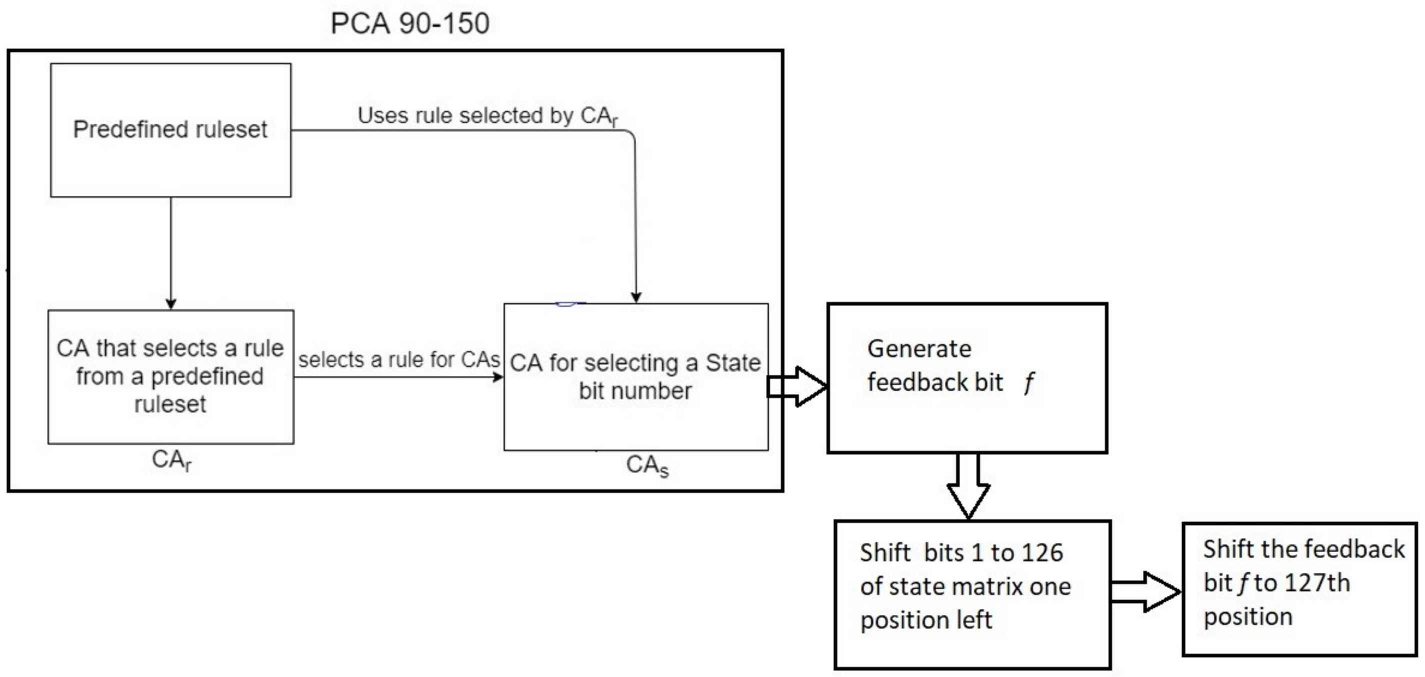
PCA 90-150
Predefined ruleset
Uses rule selected by CA$_r$
CA that selects a rule from a predefined ruleset
CA$_r$
selects a rule for CAs
CA for selecting a State bit number
CA$_s$
Generate feedback bit f
Shift bits 1 to 126 of state matrix one position left
Shift the feedback bit f to 127th position

FIGURE 7: Block diagram of encryption in CA-TinyJambu

We have used PCA 90-150 to obtain robust AE schemes. In the next section, we try to show why the PCA 90-150 is apt for use as a good pseudorandom number generator.

6 RANDOMNESS REVIEW OF PCA 90-150

It has been shown in countless references that fault attacks are easily prevented by randomization. In [9], PCA 90-150 has been compared with Controllable Cellular Automata (CCA). In PCA, there are control bits for each cell to control the rules corresponding to each cell. In CCA, there are more control lines to control the neighbourhood relations between cells and updating of states to further improve the randomness of 1-dimensional CA. CCA0 refers to type of CCA which keeps the state of the cells constant during the CA computation process. CCA1 refers to type of CCA which complements the state of the cells during the CA computation process. More details can be found in [9].

Below are some of the tests that have been performed on PCA 90-150 to prove its quality of randomness [9].

6.1 Entropy (ENT) Test

ENT [18] is a Pseudorandom Number Sequence Test Program, which applies specific tests to bytes of data in a given file and submits the results back. This test program is useful for evaluating pseudorandom number generators for encryption. ENT performs a variety of tests on the input stream and produces output based on various parameters, such as Entropy, Chi-square Test, and Serial Correlation Coefficient (SCC).

	Chi-square (pass rate)	Entropy (average value)	SCC (average value)
PCA 90-150	70%	6.101210	0.121479

TABLE 1: ENT values for PCA 90-150 [9]

As shown in Table 1, the entropy values are very good with acceptable chi-square pass rate compared to Controllable Cellular Automata (CCA) given in [9].

6.2 DIEHARD Test on PCA 90-150

DIEHARD tests [1] are a set of statistical tests used to measure the quality of randomness of a random number generator. DIEHARD is seemingly the best test for general randomness measurement. Usually, a Pseudorandom Number Generator that passes DIEHARD is considered as good.

Test name	PCA 90-150 (p)
1. Overlapping sum	Pass
2. Runs up 1	Pass
Runs Down 1	Pass
Runs up 2	Pass
Runs Down 2	Pass
3. 3D sphere	Pass
4. A parking lot	Fail
5. Birthday Spacing	Pass
6. Count the ones 1	Pass
7. Binary Rank 6*8	Pass
8. Binary Rank 31*31	Pass
9. Binary Rank 32*32	Pass
10. Count the ones 2	Pass
11. Bitstream test	Pass
12. Craps wins	Pass
13. Minimum distance	Fail
14. Overlapping Perm.	Fail
15. Squeeze	Pass
16. OPSO test	Fail
17. OQSO test	Fail
18. DNA test	Pass
Number of tests passed	*13*

TABLE 2: DIEHARD test result on PCA, p is 8-bit integer [9]

The results from Table 2 show that PCA 90-150 is potentially an excellent pseudo-random number generator passing 13 out of 18 tests which is significantly better than single ruleset hybrid cellular automata.

6.3 Randomness value variance

The randomness value variance shown in Fig. 8 is a good indicator of the randomness of the values generated by the CA. Here 15-cell PCA 90-150 is

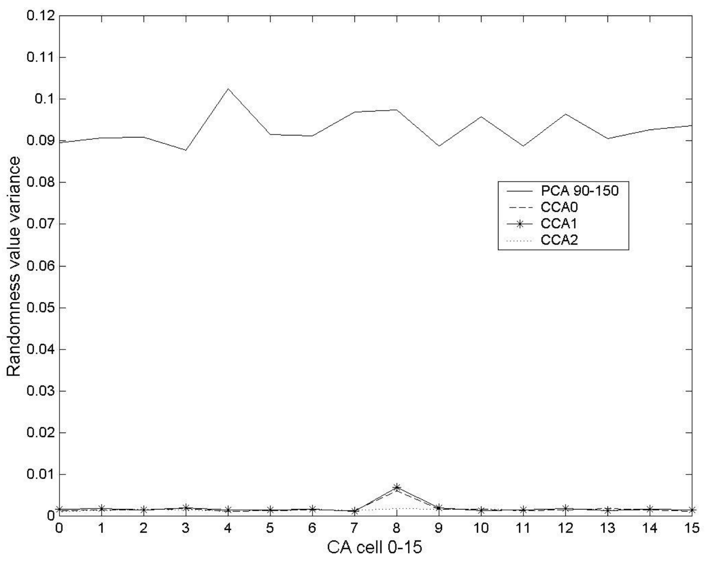

FIGURE 8: Variance of randomness value [9]

used. PCA 90-150 has higher variance values than both CCA0 and CCA1. This has 100% Chi-square pass rate, which is a very significant advantage. In addition, CCAs are much harder to implement, as well.

7 SECURITY ANALYSIS

We now provide a detailed security analysis of the enhanced algorithms and show that the authenticated encryption algorithms considered are invigorated against the considered attacks with the use of PCA.

7.1 DFA on PCA enhanced ACORN

We have described the modified design of ACORN v3 using CA in section 5.1. We have investigated the possibility of differential fault attack on modified ACORN. Locating the fault cannot be executed reliably, since the keystream bit found in every state update function cannot be predicted accurately. This is because, in each state update iteration, a random state bit is assigned as feedback bit by running the CA$_s$, which has a dynamic rules defined by CA$_r$. This is explained in more detail below.

The fault is located using 32 random initial states. These are chosen because a maximum difference of 25 is observed in the differential set by injecting one fault. In the modified design, the rule 90 and 150 used in PCA for generating keystream creates dependency between neighbouring bits of state. Hence, the number of possibilities for the differential set increases. The number of unique and non-unique sets increases exponentially with fault locations thereby making fault location extremely difficult. We now consider the equation solving considered in [24] and analyse the overhead involved from an attacker perspective. We consider 32 random states as the lower bound for analysis.

As per the specification of ACORN v3, the minimum number of steps in encryption is 1279, excluding the 1792 steps in the initialization phase. Assume that on average, the 32 random states selected for the fault locating step of the attack are near 1792/2 steps in the initialization step. Then, the minimum number of steps required is 1279 + (1792/2) = 2175. For each iteration, the attacker has to make on average 2175 accurate predictions about the keystream bit, which is to be XOR to the state and added to the last bit of the updated state. So in total, 32*2175 = 69000 accurate predictions altogether must be made in order for the first algorithm of the fault locating step to give the correct output. We can safely assume that the probability of getting 1 or 0 as the keystream bit in each iteration is 50%. Since the initial seed in the Cellular Automata is random, the attacker will have to choose one correct set of values from a permutation of 2^{69600}. Hence the probability of the attacker finding this correct combination is negligible.

As shown in [24], let n be the number of fault experiments. We can get $11.26n$ equations, including $7.03n$ linear equations. We use the guess and determine method to solve the equations. The time complexity of obtaining the initial state equals to $c * 2^{146.5-3.52n}$ approximately, where c is the time complexity of solving linear equations and 26 $< n <$ 43. Assuming the attacker reaches the correct combination of keystream bits in half the total permutations, The total complexity of obtaining the initial state would equal to

$$c * 2^{146.5-3.52n} * 2^{(69600/2)}$$

which is a huge improvement. This shows that the attacker cannot brute-force the keystream bits feasibly in order to further continue with the attack algorithm.

7.2 CA-Elephant

We have strengthened Elephant-Delirium which has a state size of 200 bits. DFA described in [25] evaluates intermediate bit a. We briefly describe the steps in the attack that determines a below to elucidate the benefits of our method later.

$$F_K(C) = F_K(c_1, .., c_n) = \Sigma_{u=(u_1,..,u_n \in GF(2^n))} \alpha_u M_u \tag{27}$$

where $\alpha_u \in 0, 1$, $M_u = \Pi_{i=1}^{n} c_i^{u_i}$.
Suppose N_{α_u} represent the number of non-zero values of α_u where α_u depend only on the secret key bits. Also, suppose the algebraic degree of $F_K(C)$ is less than d. To reduce the coefficients of $F_K(C)$, the coefficients are regarded as variables and recovered by solving a linear system of equations. Since degree of the equation for $F_K(C)$ is less than $d+1$, the sum of the intermediate bit a over a $d+1$-dimension plaintext subspace S_i is zero. That is, the sum of the values of the polynomial $F_K(C)$ over ciphertexts is zero. The solution can be determined as

$$\Sigma_{t=1}^{2^{(d+1)}} F_K(C)_t = \Sigma \alpha_u \times (\Sigma_{C \in C_{S_i}} M_u) = 0 \tag{28}$$

This is a linear equation with coefficients α_u which are functions of key bits. Each such subspace provides one linear equation. More subspaces are chosen to get N_{α_u} linear equations to recover the bits of the secret key which needs $N_{\alpha_u} \times N_{\alpha_u} \times 2^{(d+1)}$ XOR operations.

Now we describe the advantage of using CA-based key. α_u becomes degree six equations at least and hence N_{α_u} equations will not be enough to recover the bits of secret key. With the use of PCA with 6-bit seed, CA$_r$() and CA$_s$(), each key bit generated is one among $2^6 \times 2^6$ choices. For 128 bits of key, $128 \times 2^6 \times 2^6 = 2^{19}$ accurate predictions are to be made. To solve for the coefficients α_u, we will have $(2^{19} \times \alpha_u) \times (2^{19} \times \alpha_u) \times 2^{d+1}$ XOR operations involving key bits which is a huge improvement.

7.3 CA-TinyJambu

In the differential fault attack on TinyJambu, Mixed Integer Linear Programming (MILP) is used to calculate the differential trail. We briefly describe few calculations related to MILP model from [17].

Let l be the maximum length of permutation. We obtain a system of equations as shown below:

$$z_i = x_{85+i} x_{70+i} \tag{29}$$

$$\Delta z_i = x_{85+i} x_{70+i} \oplus (x_{85+i} \oplus \Delta x_{85+i}) \oplus (x_{70+i} \oplus \Delta x_{70+i}) \quad (30)$$

Here, z_i and Δz_i are the values and the differences of output bits of AND gate respectively, used in the construction of TinyJambu.

As the Δ values are known from the experiment, Equation 30 reduces to a system of linear equations. The equations can be solved by representing them using coefficient matrix and augmented coefficient matrix as discussed in [17]. The number of solutions to this system of equations is $2^{s-\gamma(M)}$ where $\gamma(M)$ is the rank of matrix M and

$$s = |x_{70+i}, x_{85+i} : 0 < i < l| \quad (31)$$

are the number of independent variables in the system of the equations. There are 2^s possible input values to the AND gate. The probability of the differential can be calculated as the number of solutions to the system of equations divided by 2^s which gives $2^{-\gamma(M)}$. The solution of MILP model always satisfies the above system of equations though the generation of optimal differential trail is not guaranteed.

The CA based design of TinyJambu generates key based on 6-cell null boundary maximal length PCA using a 6-bit seed. The variables considered above become interdependent as the key bits are themselves interdependent. The value of s decreases. There are 2^s different input to AND gate. This implies that we will need to compute more differentials. This makes it difficult to make the differential set.

The paper describing the attack [17] also considers refined models to account for correlation between variables.

$$f = \Sigma_{i=0}^{h-1}(x_i y_i + \eta_i x_i + \mu_i y_i) \quad (32)$$

where x_i, y_i are input bits with linear masks η_i, μ_i and h is the number of active AND gates. Here, f is a quadratic boolean function. In CA-based TinyJambu, the degree will be at least 6 making the calculation of differential trail impractical.

Hence, we obtain two major security advantages by invigorating TinyJambu with cellular automata, namely, difficulty in calculating the differential set and in solving differential trails.

8 CONCLUSION

The authenticated encryption schemes based on block ciphers are as secure as the underlying cryptographic primitive. However, recent research shows that use of stream ciphers and combined modes offer enhanced features. By utilizing the pseudorandom and fast-diffusion properties of CA, we have shown that DFA is ineffective against the PCA-enhanced ACORN cipher by randomizing both the state bits and the number of cycles the CA runs for in each iteration. We have shown that the differential set in equation 15 becomes huge and difficult to compute. The number of random states for fault attack increases for DFA in the current scenario. The degree of the keystream bit generation function would be greater than or equal to 8 which is much higher than previous degree of 2, effectively preventing the fault attack. Also, we have shown that it becomes infeasible to obtain the keystream bits by brute-force due to such a large number of permutations during state update. The resistance of 200-bit Elephant against interpolation attack and TinyJAMBU against differential fault attack by inclusion of CA is validated mathematically. Future work include experiments on practical devices.

REFERENCES

[1] Mohammed M Alani. (2010). Testing randomness in ciphertext of block-ciphers using diehard tests. *Int. J. Comput. Sci. Netw. Secur*, 10(4):53–57.

[2] M Bellare, P Rogaway, and D Wagner. (2003). A conventional authenticated-encryption mode. *IACR Eprint archive*.

[3] Olivier Benot. (2011). *Encyclopedia of Cryptography and Security*, pages 218–219. Springer US, Boston, MA.

[4] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer, (2008). Keccak specifications summary, `https://keccak.team/keccak_specs_summary.html`.

[5] Jaydeb Bhaumik. (2015). Synthesis of all maximum length cellular automata of cell size up to 12. *arXiv preprint arXiv:1503.04006*.

[6] Eli Biham and Adi Shamir. (1997). Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 513–525, Berlin, Heidelberg. Springer Berlin Heidelberg.

[7] Prakash Dey, Raghvendra Singh Rohit, and Avishek Adhikari. (2016). Full key recovery of acorn with a single fault. *Journal of Information Security and Applications*, 29:57–64.

[8] Guan, Sheng-Uei, and Shu Zhang. (2004). Cwc: A high-performance conventional authenticated encryption mode. *Fast Software Encryption*.

[9] Sheng-Uei Guan and Shu Zhang. (2004). Pseudorandom number generation based on controllable cellular automata. *Future Generation Computer Systems*, 20(4):627–641.

[10] Dolores de la Guía Martínez and Alberto Peinado Domínguez. (2001). On the sequences generated by 90-150 programmable cellular automata. In *5th World Multiconference on Systemics, Cybernetics and Informatics and 7th International Conference on Information System Analysis and Synthesis (SCI/ISAS 2001, Orlando, Florida)*.

[11] C S Jutla, (2001). A parallellizable authenticated encryption for ipsec. `https://tools.ietf.org/html/draft-jutla-ietf-ipsec-esp-iapm-00`.

[12] T. Krovetz, (2014). Rfc 7253: The ocb authenticated-encryption algorithm
`https://tools.ietf.org/html/rfc7253`.
`https://tools.ietf.org/html/rfc7253`.

[13] NIST, (2020). Lightweight cryptography csrc.
`https://csrc.nist.gov/projects/lightweight-cryptography/round-2-candidates`.

[14] NIST, (2020). Lightweight cryptography csrc tinyjambu v2 specification.
`https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/elephant-spec-final.pdf`.

[15] NIST, (2020). Lightweight cryptography csrc tinyjambu v2 specification.
`https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf`.

[16] Phillip Rogaway. (2002). Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 98–107.

[17] Dhiman Saha, Yu Sasaki, Danping Shi, Ferdinand Sibleyras, Siwei Sun, and Yingjie Zhang. (2020). On the security margin of tinyjambu with refined differential and linear cryptanalysis. *IACR Transactions on Symmetric Cryptology*, 2020:152–174.

[18] John Walker, (Last accessed 04 September 2020). ENT test suite. `http://www.fourmilab.ch/random`.

[19] D. Whiting, Hifn, R. Housley, Vigil Security, N. Ferguson, and MacFergus. (2003, Last accessed 3 March 2021). Counter with cbc-mac (ccm). *Counter with CBC-MAC (CCM).*

[20] Stephen Wolfram. (1984). Cellular automata as models of complexity. *Nature*, 311(5985):419–424.

[21] Stephen Wolfram. (1985). Cryptography with cellular automata. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 429–432. Springer.

[22] Hongjun Wu. (2016, Last accessed 04 September 2020). Acorn: a lightweight authenticated cipher (v3). *Candidate for the CAESAR Competition,* *`https://competitions.cr.yp.to/round3/acornv3.pdf`*.

[23] Fan Zhang, Zi-yuan Liang, Bo-lin Yang, Xin-jie Zhao, Shi-ze Guo, and Kui Ren. (2018). Survey of design and security evaluation of authenticated encryption algorithms in the caesar competition. *Frontiers of Information Technology & Electronic Engineering*, 19(12):1475–1499.

[24] Xiaojuan Zhang, Xiutao Feng, and Dongdai Lin. (2018). Fault attack on acorn v3. *The Computer Journal*, 61(8):1166–1179.

[25] Haibo Zhou, Rui Zong, Xiaoyang Dong, Keting Jia, and Willi Meier. (2021). Interpolation attacks on round-reduced elephant, kravatte and xoofff. *The Computer Journal*, 64(4):628–638.