Breaking ACORN with a Single Fault

Elena Dubrova
Royal Institute of Technology (KTH), Stockholm, Sweden
dubrova@kth.se

***Abstract*—Assuring security of the Internet of Things (IoT) is much more challenging than assuring security of centralized environments, like the cloud. A reason for this is that IoT devices are often deployed in domains that are remotely managed and monitored. Thus, their physical security cannot be guaranteed as reliably as physical security of data centers. Some believe that physical security becomes less important if all data processed and stored within a device is encrypted. However, an attacker with a physical access to a device implementing an encryption algorithm may be able to extract the encryption key and decrypt data. As a demonstration, in this paper we attack ACORN stream cipher, a finalist of CESAR competition for authenticated encryption. By injecting a single stuck-at-0 fault into ACORN's implementation, we reduce its non-linear feedback function to a linear one. Since this obviously makes ACORN weaker, many known attacks can be applied to break it. We apply an algebraic attack which recovers the key from $2^{15.34}$ keystream bits using $2^{35.46}$ operations.**

***Index Terms*—Internet of Things (IoT) security, physical security, fault attack, algebraic attack, ACORN, stream cipher.**

I. Introduction

A rapid growth of the Internet of Things (IoT) applications is expected in the coming years [1]. Household appliances, meters, sensors, and vehicles will be accessible and controlled via local networks or the Internet to provide new services appealing to users.

However, assuring IoT security is much more challenging than assuring security of centralized environments, like the cloud. While IoT inherits old problems such as weak zero-day vulnerabilities and lack of updates, it creates new problems.

First, the attack surface of future IoT with billions of connected devices will be enormous. Second, IoT relies on many different types of devices, including resource-constrained sensors and actuators. Such devices may not have enough storage, computing and energy resources for implementing a strong cryptographic protection. The security of a network is only as strong as its weakest link. A compromised sensor can potentially be used as an entry point for cyberattacks on other devices connected to the network, or the network itself [2]. Finally, IoT devices are often deployed in domains that are remotely managed and monitored. Thus, their physical security cannot be guaranteed as reliably as physical security of data centers.

Some believe that physical security becomes less important if all data processed and stored within a device is encrypted and secure access is assured [3]. However, an attacker with a physical access to an device implementing the encryption algorithm may be able to extract the encryption key and decrypt data. In this paper, we demonstrate such an attack on the example of ACORN stream cipher, a finalist of CESAR cryptographic competition for authenticated encryption (2014-2018) [4].

Previous Work. Due to its importance, ACORN has been actively cryptanalyzed before. In [5], leakage of ACORN is evaluated using t-test, and in [6] using cube testers and d-monomial test. In [7], a differential power analysis attack on ACORN is described. In [8], an EM-based side-channel attack on ACORN is presented. A SAT-based cryptanalysis of ACORN v1 and v2 is made in [9]. A state recovery attack on ACORN v1 and ACORN v2 with 2^{120} complexity is described in [10]. In [11], the key is recovered from ACORN by re-using the nonce several times to encrypt the same chosen plaintext. In [12], a cube attack on the reduced versions of ACORN v1 and v2 is presented. In [13], a state recovery attack on ACORN v2 with 2^{40} complexity is presented under a chosen plaintext attack model which assumes that, for a given key, one can find a message m whose corresponding polynomial is equal to the non-linear feedback function f of ACORN. As a result, $m \oplus f = 0$ and f is canceled from the state update. However, it is not clear how such a message can be found if the key is unknown.

Several faults attack on ACORN has also been presented. A differential fault attack on ACORN v3 which requires 9 bit flips to be injected into the initial state to recover the state with the complexity $2^{25.40}$. Zhang et al. [14] present another differential fault attack on ACORN v3 in which a bit of the initial state is flipped at random. Then, the fault is located and the resulting equations are solved. With k faults, the initial state can be recovered with time complexity $c \cdot 2^{146.5-3.52k}$, where c is the complexity of solving linear equations and $26 < k < 43$. In [15], an attack on ACORN v1 and v2 is described in which a random bit of the fifth LFSR of ACORN is stuck to the constant-1 value during the encryption. For certain faulty bit positions, the attack complexity is $2^{55.85}$.

Our Contribution. In this paper, we present a new fault attack on ACORN v3 in which a stuck-at-0 fault is injected to reduce the non-linear feedback function of ACORN to a linear one. As a result, it becomes possible to express the state update function of ACORN in terms of the linear equations depending on the key and unfold the function to the initial encryption state without its size blowing-up exponentially, as in the case of the fault-free ACORN. It also becomes possible to recover the initial encryption state from $2^{15.34}$ keystream bits with $2^{35.46}$ operations by applying known linearization methods [16], [17], [18], [19]. Once the state is recovered, the key is obtained by solving the system of linear equations

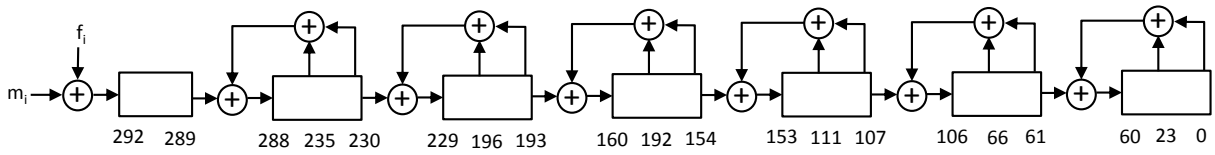

Fig. 1. Block diagram of ACORN v3.

representing the unfolded update function, which takes $2^{16.63}$ operations.

In the second part of the paper, we describe several ways of injecting stuck-at faults in practice and discuss potential countermeasures.

Paper Outline. The paper is organized as follows. Section II describes our assumptions and the attack model. Section III reviews the ACORN v3 design. Section IV presents the fault attack. Section V discuss potential countermeasures. Section VI concludes the paper.

II. Assumptions and Attack Model

We consider two possible attack scenarios. In both cases, we assume that the key is stored on chip/board. The goal of the attack is to extract the key.

In the first scenario, the attacker has a physical access to the chip implementing the stream cipher and has means of injecting faults of the following type:

- Granularity: Single bit
- Fault type: Stuck-at fault
- Control on the fault location and timing: Precise

The attacker then uses the fault-injected chip to encrypt some plaintext and uses the ciphertext for cryptanalysis.

If the attacker's goal is to decrypt the future traffic from/to the legitimate user, he/she injects the fault using some non-invasive method which does not leave the evidence of tampering. After the key is extracted, the attacker de-activates the fault and returns the compromised device to its legitimate user. Since the distribution stage of today's global supply chain of electronic products involves multiple parties, including third-party logistics providers, distributors, and retailers, any of these parties can potentially physically access and manipulate the device during its distribution. The device can also be manipulated when it is returned for a repair or maintenance.

In the second scenario, the attacker is a design house or an untrusted foundry which has means to add to the implementation of the stream cipher a hardware Trojan having the effect of a stuck-at fault in its active state. Other characteristics of the Trojan are (following to Trust-Hub [20] taxonomy):

- Insertion Phase: Design or Fabrication
- Abstraction Level: Gate
- Activation Mechanism: Triggered Internally or Externally
- Effect: Change functionality
- Location: Depends on the implementation of the algorithm
- Physical Characteristics
  - Type: Functional
  - Structure: Layout change

Once the Trojan-infected chip is manufactured, it can be physically accessed by the attacker, or attacker's collaborates, to activate the Trojan for one encryption and extract the key.

III. Design description

ACORN v3 is a bit-oriented authenticated stream cipher [21]. It is constructed from six LFSR which are composed into a 293-bit register, as shown in Fig. 1. It uses several functions: a function to generate the keystream bit, a function to compute the feedback bit, and 6 functions to update the state.

The keystream generation function, ks, is defined as follows:

$$ks(s) = s_{12} \oplus s_{154} + maj(s_{235}, s_{61}, s_{193}) \oplus ch(s_{230}, s_{111}, s_{66}) \quad (1)$$

where s_j is the jth bit of the state $s = (s_0, \ldots, s_{292})$ and maj and ch are the *majority* and the *choice* operations, respectively, defined by

$$maj(x, y, z) = xy \oplus xz \oplus yz$$
$$ch(x, y, z) = xy \oplus \overline{x}z,$$

where "$\oplus$" is the Boolean XOR and $\overline{x}$ is for the Boolean complement of x, $\overline{x} = 1 \oplus x$.

The feedback function, f, is given by:

$$f(s, ca, cb) = s_0 \oplus \overline{s}_{107} \oplus maj(s_{244}, s_{23}, s_{160}) \oplus ca \cdot s_{196} \oplus cb \cdot ks, \quad (2)$$

where "$\cdot$" is the Boolean AND, and ca and cb are the control bits. The bit ca is used to separate the processing of associated data, the processing of plaintext, and the generation of authentication tag. The bit ca is used to let the keystream bit to affect a feedback bit during initialization, processing of associated data, and the tag generation.

The state is updated in four steps. First, six LFSRs are updated as follows:

$$\begin{aligned} s_{289} &= s_{289} \oplus s_{235} \oplus s_{230} \\ s_{230} &= s_{230} \oplus s_{196} \oplus s_{193} \\ s_{193} &= s_{193} \oplus s_{160} \oplus s_{154} \\ s_{154} &= s_{154} \oplus s_{111} \oplus s_{107} \\ s_{107} &= s_{107} \oplus s_{66} \oplus s_{61} \\ s_{61} &= s_{61} \oplus s_{23} \oplus s_0 \end{aligned} \quad (3)$$

Second, the keystream bit is computed as (1). Third, the feedback bit is generated as (2). Finally, all but the input register bits are updated by a shift as $s_j^+ = s_{j+1}$, for $j \in \{0, \ldots, 291\}$, and the input bit is updated as $s_{292}^+ = f(s) \oplus m_i$,

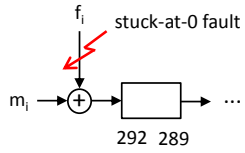

Fig. 2. Fault injection point.

where m is the data bit at the step i and s_j^+ denotes the value of s_j at the next step.

The initialization of ACORN is done as follows. First, all bits of the state are set to zero. Then, the cipher is run for 1792 steps with a bit of a 128-bit key K or a 128-bit IV being loaded into the register at the step i through the data bit m_i as follows:

$$\begin{aligned} &m_i = K_i \text{ for } i = 0 \text{ to } 127;\\ &m_i = IV_i \bmod 128, \text{ for } i = 128 \text{ to } 255;\\ &m_i = \overline{K}_i \bmod 128 \text{ for } i = 256;\\ &m_i = K_i \bmod 128 \text{ for } i = 257 \text{ to } 1535. \end{aligned} \tag{4}$$

After the initialization, the associated data is used to update the state. Even when there is no associated data, the cipher is run for 256 steps with the data bit $m_i = 0$. Otherwise, the cipher is run for $256 + adlen$ steps, where $adlen$ is the length of associated date. Then, the encryption starts.

At each step of the encryption, one plaintext bit p_i is used as the data bit $m_i = p_i$ to update the state. The ciphertext bit c_i is computed by XORing the plaintext bit p_i with the keystream bit computed according to (1).

After processing all the plaintext, the authentication tag is generated. We omit the description since it is not related to the presented work.

IV. Fault attack

The attack is performed in 3 steps:

1) Inject a stuck-at-0 fault at f_i as shown in Fig. 2.
2) Encrypt some r-bit plaintext while the fault is active.
3) Analyze the resulting r-bit ciphertext as described in Section IV-A and IV-B to recover the key.
4) Deactivate the fault after $2048 + adlen + r$ steps.

As we can see from the previous section, the initialization takes 1792 steps and the associated data processing takes $256+adlen$ steps. Assuming the associated data is empty, ACORN starts the encryption after 2048 steps.

The analysis consists of two parts: (1) deriving the value of the initial encryption state from the keystream bits, and (2) deriving the key from the initial encryption state. We start from describing the latter.

A. Deriving the key form the initial state

If $f_i = 0$ for $i = \{0, 1, \ldots, 2047\}$, its value does not affect the next state of ACORN during the initialization and the associated data processing. The state is updated by the linear function L defined by the six LFSRs and the data bit m_i, namely

$$\begin{aligned} s^+_{292} &= m_i\\ s^+_{288}(s) &= s_{289} \oplus s_{235} \oplus s_{230}\\ s^+_{229}(s) &= s_{230} \oplus s_{196} \oplus s_{193}\\ s^+_{192}(s) &= s_{193} \oplus s_{160} \oplus s_{154}\\ s^+_{153}(s) &= s_{154} \oplus s_{111} \oplus s_{107}\\ s^+_{106}(s) &= s_{107} \oplus s_{66} \oplus s_{61}\\ s^+_{60}(s) &= s_{61} \oplus s_{23} \oplus s_0 \end{aligned} \tag{5}$$

and $s_j^+ = s_{j+1}$, for all other $j \in \{0, \ldots, 291\}$. Since m_i is a function of the key K and IV, see (4), the initial encryption state is defined by $L^{2048}(K, IV)$. It can be expressed by a system of 293 linear equations depending of 128 unknown key bits and 128 known IV bits.

It is known that a linear system with k variables can be solved by the Gaussian elimination in time k^ω, where $\omega \leq 2.376$ is the exponent of the Gaussian reduction [22]. So, in our case, finding the solution takes at most $128^{2.376} = 2^{16.63}$ operations.

B. Deriving the initial state from the keystream

Given the initial encryption state $s = (s_0, \ldots, s_{292})$, ACORN with the fault $f_i = 0$ for $i = \{2048, \ldots, 2047 + r\}$, generates the keystream bits for the encryption of the plaintext $p_0, \ldots, p_{r-1}$ as follows

$$\begin{aligned} b_0 &= ks(s, p_0)\\ b_1 &= ks(L(s, p_0, p_1))\\ b_2 &= ks(L^2(s, p_0, p_1, p_2))\\ &\ldots\\ b_{r-1} &= ks(L^{r-1}(s, p_0, p_1, p_2, \ldots, p_{r-1})) \end{aligned} \tag{6}$$

where L is the linear state updating function defined by (5) where $m_j = p_j$, for $j = \{0, 1, \ldots, r-1\}$.

In the equation (1), the keystream generation function $ks(s)$ is expressed in terms of 8 variables. However, since each keystream bit is computed *after* the state bits $s_{289}, s_{230}, s_{193}, s_{154}, s_{107}, s_{61}$ are updated as shown in (3), the function $ks(s)$ actually depends on 13 bits of the state. After substituting s_{61}, s_{154}, s_{193} and s_{230} by their corresponding expressions from (3) and expanding, we get the following algebraic normal form for $ks(s)$:

$$\begin{aligned} ks(s) = {} & s_{12} + s_{66} + s_{107} + s_{111} + s_{154} + s_{235}s_{61}\\ & + s_{235}s_{23} + s_{235}s_0 + s_{61}s_{193} + s_{61}s_{160} + s_{61}s_{154}\\ & + s_{23}s_{193} + s_{23}s_{160} + s_{23}s_{154} + s_0s_{193} + s_0s_{160}\\ & + s_{235}s_{154} + s_{230}s_{111} + s_{196}s_{111} + s_{193}s_{111} + s_{230}s_{66}\\ & + s_0s_{154} + s_{235}s_{193} + s_{235}s_{160} + s_{196}s_{66} + s_{193}s_{66} \end{aligned}$$

The function $ks(s)$ is non-linear, however, there are known linearization methods [16], [19] which can find the solution to the system of the non-linear equations (6) given $r \geq \binom{n}{d}$ keystream bits and within r^ω computations, where n is the state size and d is the algebraic degree of the output function. The linearization is done by introducing a new variable for each monomial and solving the resulting system of linear equations by the Gaussian elimination. Since the number of

```
:100140008B4DE84881C1A00000000FB60921F131B4
:10015000C831C2488B45E84805C40000000FB6000E
:10016000 2245DC31C2488B45E00FB6002245D8312C
:10017000D083F0018845FF0FB645FFC9C3554889B4
:10018000E54883EC3848897DE889F0488955D848A6
:10019000894DD04489C14489CA8845E4884DE088A6

:100140008B4DE84881C1A00000000FB60921F131B4
:10015000C831C2488B45E84805C40000000FB6000E
:10016000 2245DC31D083F0018845FFC645FF00C63B
:10017000 45FF00C645FF000FB645FFC9C355488976
:10018000E54883EC3848897DE889F0488955D848A6
:10019000894DD04489C14489CA8845E4884DE088A6
```

Fig. 3. The original (upper part) and the modified (lower part) hex code.

monomials of degree d is $\begin{pmatrix} n \\ d \end{pmatrix}$ such a method is efficient if the degree d is small.

In our case, the algebraic degree of $ks(s)$ is only two. The four state bits $s_{289}, \ldots, s_{293}$ can be ignored since they just delay the input plaintext bits by 4 clock cycles. The remaining 289 bits of the state s can be recovered using $r = \begin{pmatrix} 289 \\ 2 \end{pmatrix} = 2^{15.34}$ keystream bits with $2^{15.34 \times 2.376} = 2^{35.46}$ operations.

It is worth mentioning that there are other methods, such as XL algorithm or Gröbner bases algorithms [16], [17], [18], which can find a solution of an overdefined system using less than $\begin{pmatrix} n \\ d \end{pmatrix}$ keystream bits. However, the complexity of the attack increases substantially.

C. Injecting stuck-at faults

Various fault injection methods have been proposed over the last two decades, see [23] for a overview. In this section, we first describe a fault injection technique by hex code modification which we implemented. Then, we discuss other possible options.

1) Microcontroller hex code modification: If the device under attack is implemented in a microcontroller and lock bits responsible for protecting from readback are not set, the attacker can download the original hex code programmed into the microcontroller, inject a fault, and re-program the modified code.

In general, it is difficult to do modify a hex code if the original C code is not available. However, a local modification is easier to perform. Many tools for disassembling the hex code exist, e.g. *radare2* [24] or *PICHexDisassembler* [25]. These tools can be used to partially reverse engineer the hex code and locate the attack point. In our case, the attack point is the ACORN's function $f(s, ca, cb)$.

Through trial and error, we found that it is possible to inject a fault having the effect $f(s, ca, cb) = 0$ by a local modification illustrated in Fig. 3. In the upper part of the figure, a snapshot of the original hex code of ACORN complied for x86 instruction set architecture is shown. The x86 architecture is used, for example in Intel Quark microcontrollers. The lower part of the figure shows the modified hex code, with the fault $f(s, ca, cb) = 0$ injected. The segment C2 48 8B 45 E0 0F B6 00 22 45 D8 31 (marked in blue, 24 hex symbols) is removed from the original code. After the segment D0 83 F0 01 88 45 FF (marked in pink), the segment C6 45 FF 00 (marked in yellow, 8 hex symbols) is added three times. Three repetitions are required in order to equalize the size of removed and added segments.

The segment C6 45 FF 00 (mov BYTE PTR [ebp-0x1],0x0 in assembly) implements the C statement $f = 0$, so repeating it 3 times causes no problems. The removal of the segment C2 48 8B 45 E0 0F B6 00 22 45 D8 31 has the effect of reducing the term $cb \cdot ks$ in the expression (2) to $0 \cdot ks$.

The original and modified hex codes differ only in the two lines (10016000 and 10017000). In order for the modified code to be accepted by a microcontroller, the 8-bit CRC checksums for these two lines (the last two hex symbols) have to be re-computed and replaced. The CRC makes the sum of all of the bytes in the line, including the checksum, zero.

2) FPGA bitstream modification: If the device under attack is implemented in SRAM-based FPGAs, it may be possible to inject a stuck-at fault by bitstream modification. In several works [26], [27], [28], [29] it has been shown that direct bitstream modification is feasible in practice. The techniques presented in [27], [29] are particularly relevant for our case since they target stuck-at faults. In [27] a reverse-engineering technique that finds Look-Up Tables (LUTs) in a bitstream is presented. On the examples of DES and AES, it is shown how to find positions of all LUTs implementing SBoxes and replace their content with "0"s. The CRC checksums which verify integrity of bitstream frames are re-computed and replaced by the new ones. As long as the attacker knows which Boolean function to search for, he/she may be able to find LUTs implementing this function in the bitstream. In our case, the LUTs implementing the function $f(s, ca, cb)$ have to be identified in the bitstream and their content modified to $f(s, ca, cb) = 0$.

3) EM fault injection: Theoretically, electromagnetic (EM) pulses can be used for fault injection. Such attacks can be carried out without depackaging the chip, they do not leave any evidence of tampering, and require much cheaper equipment compared to the invasive attacks [30]. It has been demonstrated that electromagnetic pulses can induce faults with the precision up to the level of a single bit [31]. However, in advanced technologies, achieving the bit-level precision might be difficult.

4) Optical fault injection: If evidence of tampering is not a problem as, for example, in the case of the attacker is the user, an optical fault injection can potentially be used. Optical attacks expose a decapsulated chip to a strong light, or use a laser beam. Since CMOS transistors are sensitive to ionizing radiation, in this way it is possible to cause a transistor to conduct. The fact that it is possible to set a selected bit to either 0 or 1 value with a precise timing has been demonstrated by Skorobogatov and Anderson already in 2002 [32]. They used a focused flash light to set individual SRAM cells in

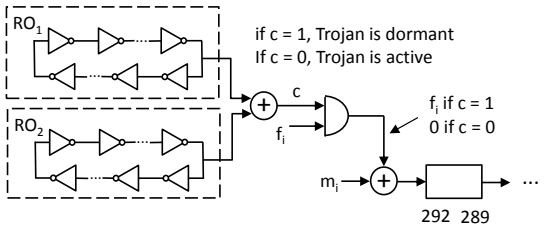

Fig. 4. Injecting a stuck-at-0 at f_i using a Trojan.

the PIC16F84 microcontroller to a fixed value. Since then the state-of-the-art in optical attacks advanced considerably. It was demonstrated that it is possible to simultaneously flip multiple bits at different locations with multiple lasers (two different bits in a Xilinx Spartan-6 45nm FPGA were simultaneously flipped using two lasers [33]) as well as to flip the same bit multiple times (the same bit was flipped more than once in the CRT-RSA running on a 32-bit ARM Cortex-M3 core [34]). However, optical methods might destroy the transistor under attack if the fault is active for many consecutive clock cycles, as in our case.

5) Hardware Trojans: A stuck-at fault at f_i can also be injected through a hardware Trojan. In today's globalized world where the manufacturing is typically outsourced and the use of third-party IPs from small and relatively new vendors is widespread, such an possibility cannot be neglected [35], [36]. Recent controversial news about the Tiny Chip hack [37] are likely to bring more attention to the problem of assuring that electronic products are Trojan-free.

A Trojan having the effect of a stuck-at-0 fault on a line can be implemented, for example, using two 90-degree out-of-phase Ring Oscillators (ROs) feeding an XOR gate, as shown in Fig. 4. While ROs are out of phase, the XOR's output is 1 and the Trojan is dormant. By injecting a signal with a specific frequency into the power supply, an attacker can lock ROs in phase, producing 0 at the XOR's output and activating the Trojan. Such a technique, called *injection locking*, was proposed in [38] with the purpose of reducing entropy of true random number generators. We can turn injection locking into a Trojan by feeding the output of the XOR into an AND gate which has takes F as its second input, as shown in Fig. 4 (right). When the output of the XOR gate is 1 (Trojan is dormant), the output of the AND is F. When the output of the XOR gate is 0 (Trojan is active), the output of the AND is 0.

V. Countermeasures

The microcontroller hex code modification attack described in Section IV-C1 can be prevented by setting lock bits responsible for protecting the program memory from readback. If lock bits are set, the program memory will be typically read back as all zeros.

Countermeasures against fault attacks usually rely on redundancy (hardware or time [39]) to detect the computational errors caused by injected faults, partly because fault injections themselves are difficult to detect [40], [30], [41], [42]. For example, the duplication with comparison [43] duplicates the cryptographic algorithm and compares the two results to detect disagreement. Such an approach can protect against single fault injections. However, attacks based on simultaneously injecting faults into both modules remain a threat [33]. Furthermore, duplication may reduce the number traces required for a successful power analysis [44], even if the duplicated module is implemented in the complementary form [45]. Countermeasures based on error-detecting codes may also simplify power analysis [46].

Hardware Trojans are typically detected either by side-channel analysis, or by testing, or by visual inspection. In side-channel analysis, signals leaked by a chip are measured and compared to the ones of a "golden" chip [47]. Testing applies test stimuli to a chip under test and monitors chip's output to detect disagreement with the specification [48], [49]. In visual inspection, layers of a chip are removed one-by-one and the exposed circuitry is scanned using various imaging methods [50].

It might be difficult to detect the Trojan shown in Fig. 4 by side-channel analysis, since it adds only a few gates to the original design. So, the change in the side-channel information is likely to be too small to be detected. Typically side-channel analysis can only detect sufficiently large Trojans that are at most three to four orders of magnitude smaller than the original design [51]. If the Trojan is dormant during testing, testing might not detect it as well, since a dormant Trojan does not change the functionality. However, visual inspection methods are likely to detect the Trojan since the Trojan changes the layout of the original design.

VI. Conclusion

We presented an attack which can recover the key of ACORN v3 from $2^{15.34}$ keystream bits using $2^{35.46}$ operations. The presented attack is general and can potentially be applied to other stream ciphers whose feedback functions can be made linear with a single fault.

Our results show that encryption does not diminish the importance of physical security. Future work involves developing techniques for assuring physical security of resource-constrained IoT devices.

VII. Acknowledgement

This work was supported in part by the research grants 2017-05232 and 2018-03964 from VINNOVA.

References

[1] Ericsson, "More that 50 billions connected devices," 2012. www.ericsson.com/ res/docs/whitepapers/wp-50-billions.pdf.

[2] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar, "Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study," in *Proc. of the 19th USENIX Conf. on Security*, (Berkeley, CA, USA), pp. 21–21, 2010.

[3] J. Edwards, "Edge computing security dos and don'ts." Network Computing, November 2018. https://www.networkcomputing.com/network-security/edge-computing-security-dos-and-donts.

[4] CESAR, "Cryptographic competition for authenticated encryption: Security, applicability, and robustness," 2018. https://competitions.cr.yp.to.

[5] W. Diehl, A. Abdulgadir, F. Farahmand, J. Kaps, and K. Gaj, "Comparison of cost of protection against differential power analysis of selected authenticated ciphers," in *2018 IEEE Int. Symp. on Hardware Oriented Security and Trust*, pp. 147–152, April 2018.

[6] A. Ghafari and H. Hu, "A new chosen iv statistical distinguishing framework to attack symmetric ciphers, and its application to ACORN-v3 and Grain-128a," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–6, 2018.

[7] A. Adomnicai, J. Fournier, and L. Masson, "Masking the lightweight authenticated ciphers acorn and ascon in software," 2018.

[8] A. Adomnicai, L. Masson, and J. Fournier, "Practical algebraic side-channel attacks against acorn," in *Proc. of Int. Conf. on Information Security and Cryptology (ICISC'2018)*, (Seoul, Korea), 2018.

[9] F. Lafitte, L. Lerman, O. Markowitch, and D. V. Heule, "Sat-based cryptanalysis of acorn," 2016.

[10] D. Dalai and D. Roy, "A state recovery attack on acorn-v1 and acorn-v2," in *Proc. of Int. Conf. on Network and System ...*, Springer, 2003.

[11] C. Chaigneau, T. Fuhr, , and H. Gilbert, "Full key-recovery on acorn in nonce-reuse and decryption-misuse settings." Posted on the Crypto-competition mailing list, 2015.

[12] Md Iftekhar Salam et al., "Investigating cube attacks on the authenticated encryption stream cipher acorn." Cryptology ePrint Archive, Report 2016/743, 2016. http://https://eprint.iacr.org/2016/743.pdf.

[13] D. Roy and S. Mukhopadhyay, "Some results on acorn." Cryptology ePrint Archive, Report 2016/1132, 2016. http://https://eprint.iacr.org/2016/1132.pdf.

[14] X. Zhang, X. Feng, and D. Lin, "Fault attack on ACORN v3." Cryptology ePrint Archive, Report 2017/855, 2017. http://eprint.iacr.org/2017/855.pdf.

[15] P. Dey, R. S. Rohit, and A. Adhikarian, "Full key recovery of ACORN with a single fault," *Journal of Information Security and Applications*, vol. 29, pp. 57–64, 2016.

[16] A. Shamir, J. Patarin, N. Courtois, and A. Klimov, "Efficient algorithms for solving overdefined systems of multivariate polynomial equations," in *Proc. of Eurocrypt'2000, LNCS 2612*, pp. 141–157, Springer, 2000.

[17] N. Courtois and J. Patarin, "About the XL algorithm over GF(2)," in *Proc. of Cryptographers' Track RSA'2003, LNCS 2612*, pp. 141–157, Springer, 2003.

[18] N. Courtois, "Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt," in *Proc. of ICISC'2002, LNCS 1807*, pp. 392–407, Springer, 2000.

[19] N. Courtois and W. Meier, "Algebraic attacks on stream ciphers with linear feedback," in *Proc. of Eurocrypt'2003, LNCS*, pp. 345–359, Springer, 2003.

[20] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," pp. 471–474, 10 2013.

[21] H. Wu, "Acorn: A lightweight authenticated cipher," 2016. https://competitions.cr.yp.to/round3/acornv3.pdf.

[22] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progression," *J. Symboic Computation*, vol. 9, pp. 251–280, 1990.

[23] I. Verbauwhede, D. Karaklajic, and J.-M. Schmidt, "The fault attack jungle - a classification model to guide you," in *Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, FDTC '11, (Washington, DC, USA), pp. 3–8, IEEE Computer Society, 2011.

[24] Radare2, "Reverse engineering framework and commandline tools." https://github.com/radare/radare2.

[25] PICHexDisassembler, "Hex disassembler for PIC 16fxxx microcontrollers." https://github.com/fredimachado/PICHexDisassembler.

[26] M. Alderighi, S. D'Angelo, M. Mancini, and G. R. Sechi, "A fault injection tool for SRAM-based FPGAs," in *9th IEEE On-Line Testing Symposium, 2003. IOLTS 2003.*, pp. 129–133, July 2003.

[27] P. Swierczynski, M. Fyrbiak, P. Koppe, and C. Paar, "FPGA trojans through detecting and weakening of cryptographic primitives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, pp. 1236–1249, Aug 2015.

[28] P. Swierczynski, M. Fyrbiak, P. Koppe, A. Moradi, and C. Paar, "Interdiction in practice—hardware trojan against a high-security USB flash drive," *Journal of Cryptographic Engineering*, vol. 7, pp. 199–211, Sep 2017.

[29] P. Swierczynski, G. Becker, A. Moradi, and C. Paar, "Bitstream fault injections (BiFI) automated fault attacks against SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 76, pp. 1–1, 2018.

[30] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proc. of the IEEE*, vol. 100, pp. 3056–3076, Nov 2012.

[31] J. Quisquater and D. Samyde, "Eddy current for magnetic analysis with active sensor," in *Esmart*, 2002.

[32] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Revised Papers from the 4th Int. Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, (London, UK), pp. 2–12, Springer-Verlag, 2003.

[33] B. Selmke, J. Heyszl, and G. Sigl, "Attack on a DFA protected AES by simultaneous laser fault injections," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 36–46, Aug 2016.

[34] E. Trichina and R. Korkikyan, "Multi fault laser attacks on protected CRT-RSA," in *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 75–86, Aug 2010.

[35] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Springer Publishing Company, Incorporated, 2011.

[36] G. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware Trojans," *Proc. of Cryptographic Hardware and Embedded Systems (CHES'2013), LNCS 8086*, pp. 197–214, 2013.

[37] Bloomberg, "The big hack: How china used a tiny chip to infiltrate U.S. companies," 2018. https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies.

[38] A. T. Markettos and S. W. Moore, "The frequency injection attack on ring-oscillator-based true random number generators," in *Proc. of CHES'09*, (Berlin, Heidelberg), pp. 317–331, Springer-Verlag, 2009.

[39] E. Dubrova, *Fault-Tolerant Design*. Springer, 2013.

[40] F. Regazzoni, T. Eisenbarth, J. Grobschadl, L. Breveglieri, P. Ienne, I. Koren, and C. Paar, "Power attacks resistance of cryptographic s-boxes with added error detection circuits," in *22nd IEEE Int. Symp. on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, pp. 508–516, Sept 2007.

[41] S. Subramanian, M. Mozaffari-Kermani, R. Azarderakhsh, and M. Nojoumian, "Reliable hardware architectures for cryptographic block ciphers LED and HIGHT," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, pp. 1750–1758, Oct 2017.

[42] G. Di Natale, M. Doulcier, M.-L. Flottes, and B. Rouzeyre, "A reliable architecture for parallel implementations of the advanced encryption standard," *Journal of Electronic Testing*, vol. 25, no. 4-5, pp. 269–278, 2009.

[43] L. Papay, *Use of SAT Solvers in Cryptanalysis*. MSc. Thesis, Comenius University, Bratislava, Slovakia, 2016.

[44] H. Pahlevanzadeh, J. Dofe, and Q. Yu, "Assessing CPA resistance of AES with different fault tolerance mechanisms," in *2016 21st Asia and South Pacific Design Automation Conf.*, pp. 661–666, Jan 2016.

[45] Y. Yu, F. Marranghello, V. D. Teijeira, and E. Dubrova, "One-sided countermeasures for side-channel attacks can backfire," in *Proc. of the 11th ACM Conf. on Security & Privacy in Wireless and Mobile Networks*, WiSec '18, (New York, NY, USA), pp. 299–301, ACM, 2018.

[46] F. Regazzoni, T. Eisenbarth, L. Breveglieri, P. Ienne, and I. Koren, "Can knowledge regarding the presence of countermeasures against fault attacks simplify power attacks on cryptographic devices?," in *2008 IEEE Int. Symp. on Defect and Fault Tolerance of VLSI Systems*, pp. 202–210, Oct 2008.

[47] R. Rad, J. Plusquellic, and M. Tehranipoor, "A sensitivity analysis of power signal methods for detecting hardware Trojans under real process and environmental conditions," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 12, pp. 1735–1744, 2010.

[48] S. Dupuis, P.-S. Ba, M.-L. Flottes, G. Di Natale, and B. Rouzeyre, "New testing procedure for finding insertion sites of stealthy hardware trojans," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*, pp. 776–781, 2015.

[49] E. Dubrova, M. Näslund, G. Carlsson, J. Fornehed, and B. Smeets, "Two countermeasures against hardware Trojans exploiting non-zero aliasing probability of BIST," *Journal of Signal Processing Systems*, pp. 1–11, 2016.

[50] F. Courbon, P. Loubet-Moundi, J. J. A. Fournier, and A. Tria, "A high efficiency hardware Trojan detection technique based on fast SEM imaging," in *Proc. of the 2015 Design, Automation & Test in Europe Conf. & Exhibition (DATE'15)*, pp. 788–793, 2015.

[51] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using IC fingerprinting," in *IEEE Symposium on Security and Privacy (SP'07)*, pp. 296–310, May 2007.