Finding State Collisions in the Authenticated Encryption Stream Cipher ACORN

Md Iftekhar Salam[1], Kenneth Koon-Ho Wong[1], Harry Bartlett[1], Leonie Simpson[1], Ed Dawson[1], and Josef Pieprzyk[1,2]

[1] Queensland University of Technology, Brisbane, QLD 4000, Australia
[2] Institute of Computer Science Polish Academy of Sciences Warsaw, Poland
{m.salam,kk.wong,h.bartlett,lr.simpson, e.dawson,josef.pieprzyk}@qut.edu.au

Abstract. This paper analyzes the authenticated encryption algorithm ACORN, a candidate in the CAESAR cryptographic competition. We identify weaknesses in the state update function of ACORN which result in collisions in the internal state of ACORN. This paper shows that for a given set of key and initialization vector values we can construct two distinct input messages which result in a collision in the ACORN internal state. Using a standard PC the collision can be found almost instantly when the secret key is known.

Keywords: CAESAR, Authenticated encryption, AEAD, ACORN, collision, stream cipher, integrity, symmetric encryption, message authentication code

1 Introduction

ACORNv1 [1] is a simple binary feedback shift register based authenticated encryption stream cipher which was recently submitted to the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [2]. Including ACORNv1, there are 57 cipher proposals submitted to the first round of the CAESAR competition. Recently, successful candidates for the second round of the CAESAR competition were announced. ACORNv1 is one of the 30 selected cipher proposals for the second round of the CAESAR competition. In September 2015, a tweaked version of ACORNv1 named ACORNv2 [3] was submitted to the second-round submission of the CAESAR competition.

Both version of ACORN use a 128-bit secret key and a 128-bit initialization vector. This proposed cipher provides authentication and encryption functionality for the input message. Encryption is performed by XOR-ing the plaintext message bits with the keystream bits output by the keystream generation function. Message authentication is provided by including a tag computed from the

message. The cipher also provides authentication but not encryption of the associated data (AD).

To the best of our knowledge there are two published analyses of ACORNv1 [4,5]. Liu and Lin [4] analyze the existence of slid pairs in ACORNv1 that generate the same state, up to a clock difference. They also explore possible state recovery attacks using guess-and-determine and differential algebraic techniques, although the attacks described in their paper are worse than exhaustive key search. Chaigneau et al. [5] show that a key recovery attack can be applied to ACORNv1 under the nonce-reuse and decryption-misuse settings. However, as the designer specifically prohibits the nonce-reuse and decryption-misuse settings, this should not be considered a feasible attack.

In this paper, we demonstrate the existence of state collisions in ACORN which can be exploited in a forgery attack. The analysis provided in this paper is based on the description of ACORNv1; however, the differences between the two versions of ACORN do not affect the validity of our analysis, so our results and conclusions are applicable to ACORNv2 also. For the rest of the paper, we used ACORN to refer to the first round submission of ACORNv1.

2 Description of ACORN

ACORN-128 uses a 128-bit key, $K \in \{0,1\}^{128}$ and a 128-bit initialization vector, $V \in \{0,1\}^{128}$. The cipher takes a plaintext message $P \in \{0,1\}^*$ of arbitrary length within the range $0 \leq |P| \leq 2^{64}$. The input to the cipher may also include associated data $D \in \{0,1\}^*$, again of arbitrary length within the range $0 \leq |D| \leq 2^{64}$. The associated data does not require confidentiality and so is not encrypted, but an integrity mechanism is applied. The output of ACORN consists of ciphertext $C \in \{0,1\}^{|P|}$ and an authentication tag $T \in \{0,1\}^{64 \leq |T| \leq 128}$. The designer of the cipher strongly recommends the use of a 128-bit tag.

The structure of ACORN is based on six binary linear feedback shift registers (LFSRs) of lengths 61, 46, 47, 39, 37 and 59, respectively, and an additional 4 bit register. This gives the cipher a total internal state size of 293 bits. Let S^t denote the internal state of ACORN at time t. Let s_i^t denote the contents of register stage i of the state at time t, where $0 \leq i < 293$.

Operations performed in the ACORN stream cipher can be divided into four phases: Initialization, Encryption, Tag Generation and Decryption and Tag Verification. The differences between ACORNv1 and ACORNv2 occur only in the initialization phase. The operations performed by the cipher during these phases are based on several component functions, as described below.

2.1 ACORN Component Functions

ACORN uses three functions: an output function which generates a single bit from the internal state of the cipher, a nonlinear feedback function used to update the register stage s_{292} and a state update function used to update the register stages of the LFSRs. In the paper the author describes the state update function as follows:

Algorithm 1 Wu's Pseudo code for ACORN state update function

$s^t_{289} = s^t_{289} \oplus s^t_{235} \oplus s^t_{230}$
$s^t_{230} = s^t_{230} \oplus s^t_{196} \oplus s^t_{193}$
$s^t_{193} = s^t_{193} \oplus s^t_{160} \oplus s^t_{154}$
$s^t_{154} = s^t_{154} \oplus s^t_{111} \oplus s^t_{107}$
$s^t_{107} = s^t_{107} \oplus s^t_{66} \oplus s^t_{61}$
$s^t_{61} = s^t_{61} \oplus s^t_{23} \oplus s^t_0$
Compute Output Function, Y^t
Compute Feedback Function, f'
for $i := 0$ **to** 291 **do**
  $s^{t+1}_i = s^t_{i+1}$
end for
$s^{t+1}_{292} = f^t \oplus M^t$

Output Function. At time instant t, the output function of ACORN described by Wu [1], takes input from five of the stages: s_{12}, s_{61}, s_{154}, s_{193} and s_{235}. This version of the output bit Y^t at time t is defined as:

$$\begin{aligned} Y^t &= f_y(s^t_{12}, s^t_{61}, s^t_{154}, s^t_{193}, s^t_{235}) \\ &= s^t_{12} \oplus s^t_{154} \oplus maj(s^t_{61}, s^t_{193}, s^t_{235}) \end{aligned} \tag{1}$$

where the majority function is defined as:

$$maj(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_1x_3 \tag{2}$$

Note that in Equation 1, the contents of register stages s_{61}, s_{154} and s_{193} have undergone an intermediate update before they are used as inputs. This is not highlighted in Wu's version of the output function but is implicit in the state update function given in Algorithm 1.

We wish to express the output function directly in terms of the contents of the register prior to update. To do this, we substitute the updated values of stages s_{61}, s_{154} and s_{193} into Equation 1 and also use the definition of the majority function to obtain the following function:

$$\begin{aligned} Z^t &= f_z(s^t_{12}, s^t_{61}, s^t_{23}, s^t_0, s^t_{154}, s^t_{111}, s^t_{107}, s^t_{193}, s^t_{160}, s^t_{235}) \\ &= s^t_{12} \oplus s^t_{154} \oplus s^t_{111} \oplus s^t_{107} \oplus s^t_{61}s^t_{193} \oplus s^t_{61}s^t_{160} \oplus s^t_{61}s^t_{154} \oplus s^t_{23}s^t_{193} \\ &\oplus s^t_{23}s^t_{160} \oplus s^t_{23}s^t_{154} \oplus s^t_0s^t_{193} \oplus s^t_0s^t_{160} \oplus s^t_0s^t_{154} \oplus s^t_{193}s^t_{235} \oplus s^t_{160}s^t_{235} \\ &\oplus s^t_{154}s^t_{235} \oplus s^t_{61}s^t_{235} \oplus s^t_{23}s^t_{235} \oplus s^t_0s^t_{235} \end{aligned} \tag{3}$$

Note that including the intermediate updates in the output function increases the number of input register stages by five, as shown in Equation 3. However, it does not affect the degree of the output equation, which remains quadratic. In total, the output function will take input from ten register stages: s^t_0, s^t_{12}, s^t_{23}, s^t_{61}, s^t_{107}, s^t_{111}, s^t_{154}, s^t_{160}, s^t_{193} and s^t_{235}.

Feedback Function. The generic form of the feedback function of ACORN, as described by Wu, uses the contents of the fourteen register stages s_0, s_{12}, s_{23}, s_{61}, s_{66}, s_{107}, s_{111}, s_{154}, s_{160}, s_{193}, s_{196}, s_{230}, s_{235}, s_{244} and two control bits a, b. The values for a and b vary depending on the phase: initialization, encryption or tag generation. Wu describes the feedback function as:

$$f' = s_0^t \oplus \overline{s_{107}^t} \oplus maj(s_{244}^t, s_{23}^t, s_{160}^t) \oplus ch(s_{230}^t, s_{111}^t, s_{66}^t) \oplus a^t s_{196}^t \oplus b^t Z^t \quad (4)$$

where $\overline{s_{107}^t}$ denotes the complement value of the content of register stage s_{107}^t. In Equation 4, the choice function is defined as:

$$ch(x_1, x_2, x_3) = x_1 x_2 \oplus \overline{x_1} x_3 \quad (5)$$

where $\overline{x_1}$ denotes the complement of x_1. Similar to the output function, Wu's version of the feedback function given in Equation 4 incorporates implicitly the intermediate updates of the register stages s_{61}, s_{107}, s_{154}, s_{193} and s_{230}. After substituting the updated values of these register stages and using algebraic expressions for the majority function, choice function and output function, we can express the generic form of the feedback function as follows:

$$\begin{aligned} f(S^t, a^t, b^t) = 1 \oplus s_0^t \oplus s_{107}^t \oplus s_{61}^t \oplus s_{244}^t s_{23}^t \oplus s_{23}^t s_{160}^t \oplus s_{160}^t s_{244}^t \oplus s_{230}^t s_{111}^t \\ \oplus s_{196}^t s_{111}^t \oplus s_{193}^t s_{111}^t \oplus s_{230}^t s_{66}^t \oplus s_{196}^t s_{66}^t \oplus s_{193}^t s_{66}^t \oplus a^t s_{196}^t \oplus b^t \left(s_{12}^t \right. \\ \oplus s_{154}^t \oplus s_{111}^t \oplus s_{107} \oplus s_{61}^t s_{193}^t \oplus s_{61} s_{160}^t \oplus s_{61}^t s_{154}^t \oplus s_{23}^t s_{193}^t \oplus s_{23}^t s_{160}^t \\ \oplus s_{23}^t s_{154}^t \oplus s_0^t s_{193}^t \oplus s_0^t s_{160}^t \oplus s_0^t s_{154}^t \oplus s_{193}^t s_{235}^t \oplus s_{160}^t s_{235}^t \oplus s_{154}^t s_{235}^t \\ \left. \oplus s_{61}^t s_{235}^t \oplus s_{23}^t s_{235}^t \oplus s_0^t s_{235}^t \right) \end{aligned} \quad (6)$$

Equation 6 is quadratic for given constant values of a and b. We fix the values of a and b to specifiy four possible feedback functions. We use the notation f_T, f_A, f_E and f_I to denote the specific feedback functions for $(a, b) = (0,0), (0,1), (1,0)$ and $(1,1)$ respectively. These four feedback functions are given in Table 1.

State Update Function. Let the input to the state at time t be denoted by M^t. In each iteration of the state update function the register stages are updated by shifting, except for seven stages s_{60}, s_{106}, s_{153}, s_{192}, s_{229}, s_{288} and s_{292}. The last register stage, s_{292}, is updated by combining the output of the nonlinear feedback function with the input message. Register stages s_{60}, s_{106}, s_{153}, s_{192}, s_{229} and s_{288} are updated as linear combinations of the contents of selected register stages. The state of ACORN at time $t+1$ is defined as:

$$s_i^{t+1} = \begin{cases} M^{t+1} \oplus f\left(S^t, a^t, b^t\right) & \text{for } i = 292 \\ s_{289}^t \oplus s_{235}^t \oplus s_{230}^t & \text{for } i = 288 \\ s_{230}^t \oplus s_{196}^t \oplus s_{193}^t & \text{for } i = 229 \\ s_{193}^t \oplus s_{160}^t \oplus s_{154}^t & \text{for } i = 192 \\ s_{154}^t \oplus s_{111}^t \oplus s_{107}^t & \text{for } i = 153 \\ s_{107}^t \oplus s_{66}^t \oplus s_{61}^t & \text{for } i = 106 \\ s_{61}^t \oplus s_{23}^t \oplus s_0^t & \text{for } i = 60 \\ s_{i+1}^t & \text{otherwise} \end{cases} \quad (7)$$

Table 1. Feedback functions for different operation phase of the cipher

a b	f	Feedback Function
0 0	f_T	$1 \oplus s_0^t \oplus s_{61}^t \oplus s_{244}^t s_{23}^t \oplus s_{160}^t \left(s_{23}^t \oplus s_{244}^t\right) \oplus \left(s_{230}^t \oplus s_{196}^t \oplus s_{193}^t\right) \left(s_{66}^t \oplus s_{111}^t\right) \oplus s_{107}^t$
0 1	f_A	$1 \oplus s_0^t \oplus s_{61}^t \oplus s_{244}^t s_{23}^t \oplus s_{160}^t \left(s_{23}^t \oplus s_{244}^t\right) \oplus \left(s_{230}^t \oplus s_{196}^t \oplus s_{193}^t\right) \left(s_{66}^t \oplus s_{111}^t\right) \oplus s_{12}^t \oplus s_{154}^t \oplus s_{111}^t \oplus s_{61}^t s_{193}^t \oplus s_{61}^t s_{160}^t \oplus s_{61}^t s_{154}^t \oplus s_{23}^t s_{193}^t \oplus s_{23}^t s_{160}^t \oplus s_{23}^t s_{154}^t \oplus s_0^t s_{193}^t \oplus s_0^t s_{160}^t \oplus s_0^t s_{154}^t \oplus s_{193}^t s_{235}^t \oplus s_{160}^t s_{235}^t \oplus s_{154}^t s_{235}^t \oplus s_{61}^t s_{235}^t \oplus s_{23}^t s_{235}^t \oplus s_0^t s_{235}^t$
1 0	f_E	$1 \oplus s_0^t \oplus s_{61}^t \oplus s_{244}^t s_{23}^t \oplus s_{160}^t \left(s_{23}^t \oplus s_{244}^t\right) \oplus \left(s_{230}^t \oplus s_{196}^t \oplus s_{193}^t\right) \left(s_{66}^t \oplus s_{111}^t\right) \oplus s_{107}^t \oplus s_{196}^t$
1 1	f_I	$1 \oplus s_0^t \oplus s_{61}^t \oplus s_{244}^t s_{23}^t \oplus s_{160}^t \left(s_{23}^t \oplus s_{244}^t\right) \oplus \left(s_{230}^t \oplus s_{196}^t \oplus s_{193}^t\right) \left(s_{66}^t \oplus s_{111}^t\right) \oplus s_{196}^t \oplus s_{12}^t \oplus s_{154}^t \oplus s_{111}^t \oplus s_{61}^t s_{193}^t \oplus s_{61}^t s_{160}^t \oplus s_{61}^t s_{154}^t \oplus s_{23}^t s_{193}^t \oplus s_{23}^t s_{160}^t \oplus s_{23}^t s_{154}^t \oplus s_0^t s_{193}^t \oplus s_0^t s_{160}^t \oplus s_0^t s_{154}^t \oplus s_{193}^t s_{235}^t \oplus s_{160}^t s_{235}^t \oplus s_{154}^t s_{235}^t \oplus s_{61}^t s_{235}^t \oplus s_{23}^t s_{235}^t \oplus s_0^t s_{235}^t$

where $0 \leq i \leq 292$. Figure 1 shows the state update process for ACORN in diagrammatic form. Depending on the phase the cipher is in, the input M can denote either the key, initialization vector, associated data, plaintext or fixed length padding.

2.2 Initialization

The initialization procedure uses the key, initialization vector and associated data as inputs to construct the initial internal state of ACORN. The initialization process is performed in two phases: first the key and initialization vector are loaded and then the associated data is loaded. The description below applies to ACORNv1. As noted above, the changes in ACORNv2 do not affect the validity of our analysis below.

Key and Initialization Vector Loading. To begin, the internal state of the cipher is loaded with all zero bits, i.e, $s_i = 0$ for $i = 0, \cdots, 292$. The cipher takes an input, I, of 1536 bits which consists of the concatenation of the 128-bit key, K, the 128-bit initialization vector, V, and 1280 bits of padding, π_I. The padding consists of one bit with value 1, followed by 1279 bits of 0.

$$\begin{aligned} I &= (k_1, \cdots, k_{128} || v_1, \cdots, v_{128} || \pi_{I1}, \pi_{I2}, \cdots, \pi_{I1280}) \\ &= (k_1, \cdots, k_{128} || v_1, \cdots, v_{128} || 1, 0, 0, 0, \cdots, 0) \end{aligned} \tag{8}$$

During the key and initialization vector loading, both of the control bits are set to 1, i.e, $a = b = 1$. Therefore, during the key and initialization vector loading phase the nonlinear update function is the function f_I as per Table 1. The cipher is run for 1536 clocks. At each clock, the register stages are updated using the state update function given in Equation 7, with the input I used as the message M. Figure 2 shows the initialization procedure of ACORNv1.

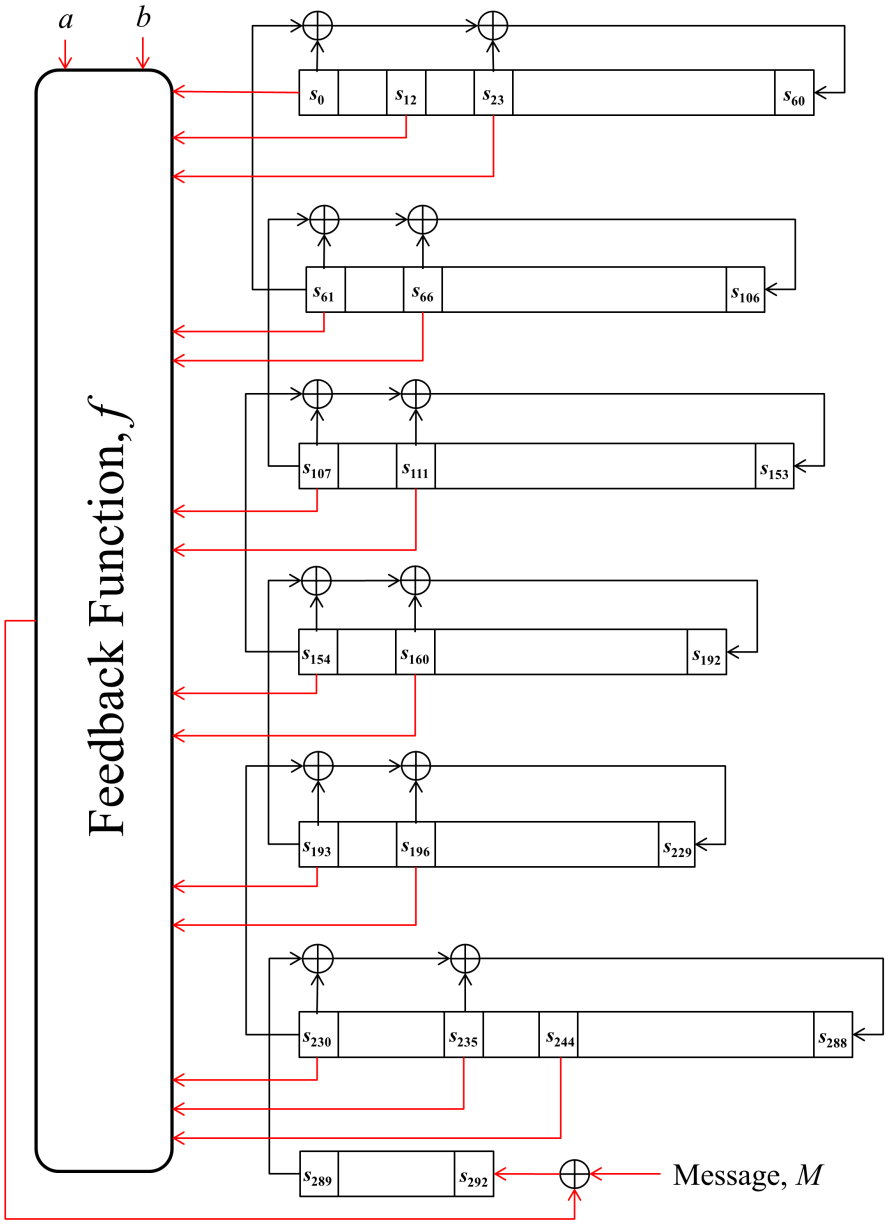

Fig. 1. ACORN State update

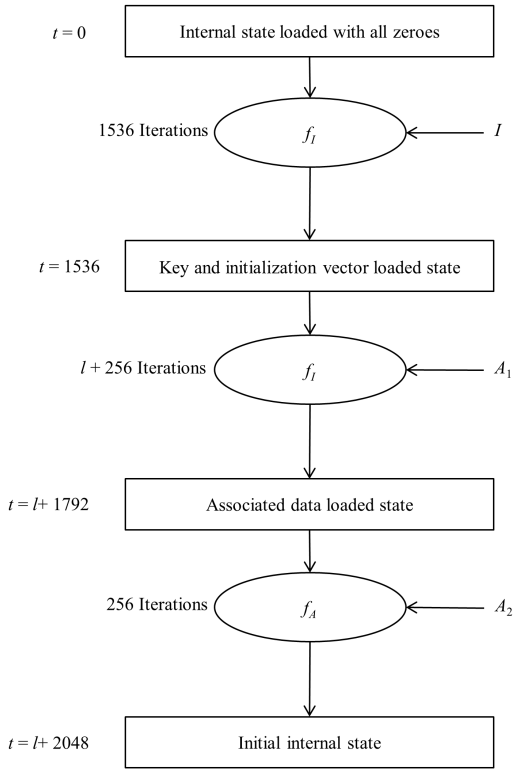

Fig. 2. Initialization

Associated Data Loading. After loading the key, initialization vector and subsequent padding, the cipher next processes the associated data. In this phase, the cipher takes an input vector A consisting of l bits of associated data, D, with 512 bits of padding, π_A. The first padding bit is set to one and the rest of the padding bits are set to zeroes, i.e, $\pi_A = 1, 0, 0, 0, \cdots, 0$.

$$\begin{aligned} A &= (d_1, \cdots, d_l || \pi_{A1}, \pi_{A2}, \cdots, \pi_{A512}) \\ &= (d_1, \cdots, d_l || 1, 0, 0, 0, \cdots, 0) \end{aligned} \tag{9}$$

The cipher is run for $l+512$ clocks to update the state using the state update function. During associated data loading, the control bits a and b are set to 1, i.e, $a = b = 1$ for the first $l + 256$ clocks. Therefore, feedback function f_I given in Table 1 will be used for the first $l + 256$ steps of the associated data loading process with the input vector $A_1 = (d_1, \cdots, d_l || \pi_{A1}, \pi_{A2}, \cdots, \pi_{A256})$. For the remaining 256 iterations, control bit a is set to zero and b is set to one. That is, the feedback function f_A given in Table 1 is used for the last 256 clocks of the associated data loading process with the input vector $A_2 = (\pi_{A257}, \cdots, \pi_{A512})$.

Note that even if there is no associated data the cipher will still need to run for 512 clocks to incorporate the padding bits.

2.3 Encryption

The encryption procedure takes the plaintext, P, as input and computes the output ciphertext, C. During the encryption phase, the control bits a and b are set to one and zero, respectively. Therefore, the nonlinear update function f_E given in Table 1 is used during the encryption phase.

The n-bit plaintext message $P = (p_1, \cdots, p_n)$ is loaded into the register stages using the state update function. In the state update function, the plaintext message P will be used as the input message M during the encryption phase. The output bits Z are used as keystream bits and the ciphertext, C is computed by XOR-ing these bits with the input message bits, P. The encryption procedure of ACORNv1 is shown in Figure 3.

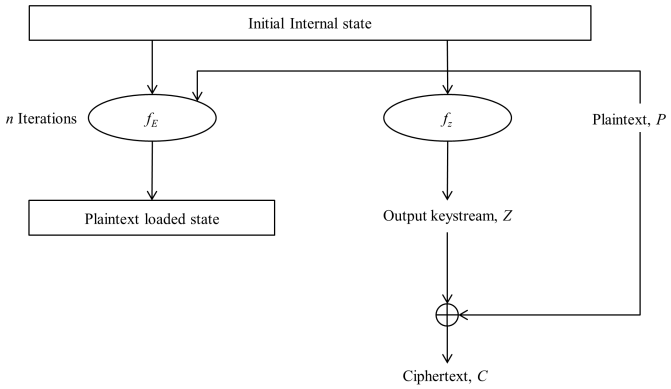

Fig. 3. Encryption

2.4 Tag Generation

After all the plaintext has been encrypted, the cipher goes through some finalization steps to generate the authentication tag. During this procedure, it takes a 1024 bit input vector τ, consisting of one bit with value 1, followed by 1023 bits of value 0.

$$\begin{aligned} \tau &= (\tau_1, \tau_2, \cdots, \tau_{1024}) \\ &= (1, 0, 0, 0, \cdots, 0) \end{aligned} \tag{10}$$

This input vector is XOR-ed with the feedback bits to update the cipher. The cipher uses different feedback functions at different steps of this phase. For the first 256 clocks, the feedback function f_E given in Table 1 is used in the state update function. For the next 256 clocks, both of the control bits are set to zero and the function f_T given in Table 1 is used. For the next 512 clocks, the cipher is run with the feedback function f_I and for the last l_{tag} number of iterations the keystream bits Z are computed and used as the tag, where $64 \leq l_{tag} \leq 128$ is the length of the tag. Figure 4 shows the tag generation phase of ACORNv1.

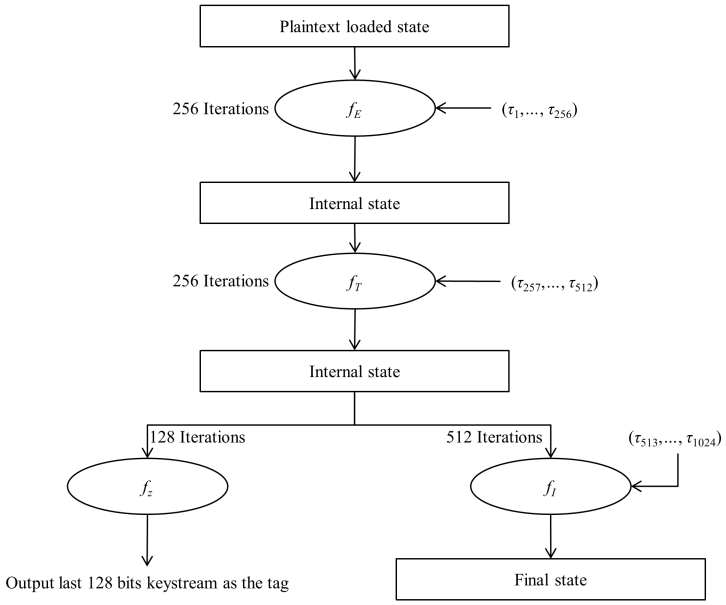

Fig. 4. Tag Generation

2.5 Decryption and Tag Verification

To carry out the decryption, the cipher first performs the initialization process to obtain the initial internal state. During the actual decryption phase, the keystream generator generates a keystream bit Z^t at each clock, t. This bit is XOR-ed with the ciphertext bit C^t at that time instant to obtain the decrypted plaintext bit. The decrypted plaintext bit is then fed into the keystream generator as the input message M. This process is iterated till all the ciphertext bits are processed. After processing all the ciphertext bits the tag is generated at the receiver, following the same procedure as mentioned above. Finally for the

verification, the tag generated at the receiver side is matched with the received tag from the sender. Clearly, the verification succeeds if the plaintext used at the sender is the same as the decrypted plaintext at the receiver.

3 Analysis of ACORN

In this section we make observations on the ACORN design, related to the possibility of state convergence and state collision in the internal state. Our motivation for analyzing the convergence or collision in ACORN is to find different input messages which generate the same tag, and hence could be used to facilitate forgery attacks.

We consider state convergence for the situation in which the keystream generator has no external input (or in which the external input is fixed and an attacker therefore has no opportunity to influence the values of M). Convergence occurs when two or more distinct states at a given time are mapped into the same state after α iterations, for some $\alpha > 0$. In the operation of a general cipher, state convergence can occur during any phase, i.e., initialization, encryption or the tag generation, if the state update function is not one-to-one.

State collisions occur when two different sets of inputs produce identical internal states at some point of operation of the cipher. Here, the input combination implies different possibilities for key, initialization vector and external input message combination. State collisions may be exploited in a forgery attack or used in secret key recovery attacks.

Both state convergence and state collision result in identical internal states at some point in the operation of a cipher. The difference is that state convergence occurs when the cipher runs autonomously (or with fixed input) whereas state collisions can be engineered by manipulating the external inputs to the cipher. ACORN does not have any external variable input messages during the tag generation phase, so state collisions can not be forced in this phase of the cipher operation. However, state collision might be obtained in either initialization or encryption phase for different inputs of key, initialization vector and input message. This will then lead to the formation of identical message authentication tags. That is, if two sets of inputs K, V, M and K', V', M' result in colliding states, then the generated tag after the final phase will be the same, given that the rest of the input bits are the same once the colliding states are obtained.

A forgery attack can be applied to the ACORN message authentication algorithm if two input combinations which result in the same tag value can be identified. This is particularly useful for the scenario where $K = K'$, $V = V'$ but $M \neq M'$. That is, for a given key and initialization vector, an attacker can find two distinct input messages which will result in the same tag. A malicious sender can manipulate either the associated data or the plaintext messages to obtain the collision of the tag and therefore can frame a forged message accepted as legitimate. An attacker can easily manipulate the associated data, because this is a publicly available information.

3.1 Convergence of Two Different States

In this section, we explore the possibility of state convergence in ACORN. From the state update function given in Equation 7, we see that register stages s_{60}, s_{106}, s_{153}, s_{192}, s_{229} and s_{288} are updated with linear combinations of the contents of several other stage bits. If two ACORN states differ only in the register stages s_0, s_{61}, s_{107}, s_{154}, s_{193}, s_{230} and s_{289} by complementing the contents of all these stages, then the effect of these linear updates remain unchanged. However, some of these register stage bits are fed to the nonlinear feedback function which updates the register stage s_{292}.

ACORN uses different feedback functions f_I, f_A, f_E and f_T at various points during the different operation phases of the cipher. The quadratic terms involving the register stages s_0, s_{61}, s_{107}, s_{154}, s_{193}, s_{230} and s_{289} in these feedback functions always occur in an even number of XOR combinations. Complementing the values in these stages does not change the computed value of the sum of the quadratic terms, since the rest of the register stage values are same for both states. On the other hand, the linear terms involving these register stages in these feedback functions always occur in odd number, e.g., both f_I and f_A contain the XOR combination of the linear terms s_0, s_{61} and s_{154}. Complementing the values in these register stages will also complement the output from the the sum of the linear terms. Since the sum of the quadratic terms does not change and the sum of the linear terms does change, the computed value for the feedback functions will be complemented when the contents of these stages are all complemented. Hence state convergence is not possible during any of the operation phases of ACORN.

3.2 Collision of States from Different Inputs

During the initialization and encryption phase, external input messages are fed into the keystream generator. During the initialization phase the inputs to the keystream generator are the key, initialization vector and associated data, whereas during the encryption phase the external input is the plaintext. In this section we discuss the state collisions by exploiting these external messages during the initialization and encryption phase of ACORN.

In section 3.1 we noted that two different ACORN states which differ only in the contents of register stages s_0, s_{61}, s_{107}, s_{154}, s_{193}, s_{230} and s_{289}, will have identical register stages in the next clock for all stages except stage s_{292}. Note that the content of register stage s_{292} can be influenced at certain times by the external input M. We can therefore force a state collision by choosing the appropriate external input. For example, consider two different states S and S', which differ only in the register stages shown in Table 2.

Here, $\overline{s_i}$ corresponds to the complement of s_i, and $\overline{M}$ denotes the complement value of input M. Notice that the two states described in Table 2 will collide in the next clock, if the next input bit M is complemented for the corresponding time instant. So, there will be 128 possible combinations of values for the register stages shown in Table 2. We can group these into 64 complementary pairs. For

Table 2. Obtaining a collision with different inputs

State	Differences in the register stage	Input
S	s_0, s_{61}, s_{107}, s_{154}, s_{193}, s_{230}, s_{289}	M
S'	$\overline{s_0}$, $\overline{s_{61}}$, $\overline{s_{107}}$, $\overline{s_{154}}$, $\overline{s_{193}}$, $\overline{s_{230}}$, $\overline{s_{289}}$	$\overline{M}$

any combination of values in these stages and either choice for the external message bit, a collision can be obtained by complementing all of these stages and also complementing the external input message bit. In practice, this means if we take any state and any value for the next bit of input data, that combination will collide with the state that is identical apart from having the contents of register stages s_0, s_{61}, s_{107}, s_{154}, s_{193}, s_{230} and s_{289} complemented and also using the complementary value for the next bit of input data.

Now consider the different phases of ACORN to determine when we can obtain a state collision with chosen set of external input data. At the beginning, the 128-bit secret key and 128-bit public initialization vector is the input to the cipher for the key and initialization vector loading process of the initialization phase. The cipher will be clocked 256 times to load these key and initialization vector into the register stages. To manipulate the state collision with the selected input value, i.e., using chosen initialization vector for this instance, two states need to have complementary values in the above mentioned seven register stages. However, this will not happen after 256 clocks since the key-initialization vector mapping will reach up to register stage s_{37} and the rest of the register stages $s_0, \cdots, s_{36}$ still contain zero values. To force the collision, the two states need to have complementary values in s_0 which is not the case. Therefore, collisions can not be forced during the key and initialization vector loading phase by the above mentioned process.

During the associated data loading process in the initialization phase, the feedback function f_I is used to load the l bits of associated data into the register stages of the cipher. If $l + 256 > 293$ then the input to the cipher in bits is greater than the total internal state size. Therefore, if the associated data length is more than 37 bits, i.e, $l > 37$, then there must exist state collisions. At the beginning of the associated data loading process, the internal state consists of the key and initialization vector loaded state and the input to the cipher is the publicly known associated data. The key, initialization vector and associated data combination will be mapped into all the 293 register stages when there are more than 37 bits of associated data. Alternatively, if we consider a fixed secret key while varying both the initialization vector and associated data to obtain a state collision, then we need to have the associated data length, $l > 165$. At this point any two states which are identical except from having complementary values in the specific seven register stages will be forced to have a collision if they are fed with complementary values of associated data in the corresponding time instant. Therefore, a collision can be forced during the associated data loading process. Similar comments apply to the encryption phase of the cipher where one can obtain a state collision by manipulating the plaintext messages.

4 Finding State Collisions

In this section, we describe the procedure for finding state collisions in ACORN. We also discuss the feasibility of this approach.

To find the state collisions, a set of quadratic equations is defined, relating the contents of two ACORN states formed from two different input messages. We can solve this system of equations to obtain pairs of distinct input messages which generate the colliding states.

4.1 Equation Generation

Let S^t denote an internal state of ACORN at time t with an input message M. Our goal is to construct another input message M' for an internal state S'^t in such a way, that $M \neq M'$ and it results in identical state after processing the input messages. The input message M and M' can represent either the associated data or the plaintext.

We first define the contents of the internal state S^t with 293 variables, i.e., $S^t = (s_0^t, \cdots, s_{292}^t)$. Similarly, the contents of the internal state $S'^t = (s_0'^t, \cdots, s_{292}'^t)$. At this stage, we are not assuming that $S^t = S'^t$. To find the colliding states, we construct a set of equations by relating the contents of S^{t+293} and S'^{t+293}, formed from the input message M and M', respectively. For an input message of length 293 bits, processing the message results in changes in all the register stages.

While the input message is being loaded, the degree of the generated equation system increases because of the nonlinear feedback function. To keep the degree of the nonlinear feedback function in quadratic form we apply the relabeling technique at each iteration of the message loading. In total, the relabeling introduces 586 new variables and 586 new equations in the equation system. Also, 586 more variables are required to represent the input messages to the internal state S^t and S'^t.

After loading the 293 bit message, the two ACORN internal states are defined as: $S^{t+293} = s_0^{t+293}, \cdots, s_{292}^{t+293}$ and $S'^{t+293} = s_0'^{t+293}, \cdots, s_{292}'^{t+293}$. We obtain 293 equations which equate the content of each register stage in S^{t+293} and S'^{t+293} except for register stages numbered 0, 61, 107, 154, 193, 230 and 289. Equations for these remaining seven pairs of register stages are generated in a way that the corresponding register stages in S^{t+293} and S'^{t+293} contain complementary values. Thus after 293 clocks of message loading phase, the equations comparing the internal state S^{t+293} and S'^{t+293} are given by Equation 11.

$$s_i^t \oplus s_i'^t = \begin{cases} 1 & i = 0, 61, 107, 154, 193, 230, 289 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where $0 \leq i \leq 292$. In total there are 879 equations with 1758 variables. Obtaining solutions for M and M' by solving these system of equations means we can obtain all possible combinations of states and input messages which differ only by complementing the specific register stages. If a solution for this system

of equations can be obtained, then a collision can be forced by choosing complementary values for M and M' at the 294th clock of the associated data loading phase or plaintext loading phase.

4.2 Solving the Equations

In this section we discuss how to solve the generated system of equations to obtain inputs which demonstrate state collision in ACORN. Experiments to obtain collision examples are performed using Sage version 6.4.1 [6] on a standard 3.4GHz Intel core i7 PC with 16GB memory.

The generated equation system as described in the previous section is underdefined and the number of equations in the system is quite large, which makes it hard to solve. A large portion of the variables are due to the unknowns in the internal state S^t and S'^t. So, we reduced the size of the system of equations by fixing the contents in S^t and S'^t instead of defining them with unknown variables. More precisely the assumption here is that the internal states S^t and S'^t are known. The key and initialization vector loaded state is assumed to be known if associated data is used as the input message to generate the collision. The initial internal state is assumed to be known if plaintext is used as the input message to generate the collision. Also, we assume that the input message M for state S^t is known. Only the input message M' for state S'^t is represented with 293 unknown variables. These assumptions reduces the size of the equation system to 586 equations with 586 variables. These equations are solved using Gröbner basis methods. Solving this equation system give unique values for each of the unknown variables that constitute the bits of the input message M'.

Collisions for Different Associated Data Sets. This section demonstrate state collisions in ACORN for two distinct inputs of associated data. We used the technique described in the previous section to find two different sets of associated data which generate identical associated data loaded states.

In ACORN, associated data are fed into the key and initialization vector loaded state after $t = 1536$ iterations of the key and initialization vector loading phase. Let the key and initialization vector loaded states S^{1536} and S'^{1536} be formed using the key and initialization vector pair (K, V) and (K', V'), respectively. Suppose first that these two states S^{1536} and S'^{1536} are identical, i.e., $K = K'$ and $V = V'$. At this point, associated data D and D' are fed into S^{1536} and S'^{1536}, respectively. At each iteration of the associated data loading, a set of equations are generated relating the input associated data D and D' with the contents of the corresponding internal state. To keep the size of the equation system small, we assume that the key and IV loaded states S^{1536} and S'^{1536} are known. For a chosen associated data D, calculations are then performed to obtain the associated data D' generating state S^{1829} and S'^{1829}, respectively, which differ only in the register stages numbered 0, 61, 107, 154, 193, 230 and 289. A state collision is then forced at the 294th clock by choosing complementary values of the input associated data bits D and D' for that time instant. This

gives two identical associated data loaded state $S^{1830} = S'^{1830}$, for two distinct associated data sets D and D', respectively.

Table 3 gives an example of pair of input data sets which leads to a collision for same key and initialization vector pair with different associated data (All tabulated values are in hexadecimal notation). As shown in Table 3, except for the associated data all the other inputs are same, i.e., $K = K'$, $V = V'$ and $P = P'$. The input associated data D in the above example is changed to D' which results in a collided associated data loaded state after 294 iterations. Since the inputs to the states are kept same once the identical associated data loaded states are obtained, it will result in the same ciphertext $C = C'$ and tag $T = T'$ after processing the rest of the inputs.

Table 3. Example of collision for same K, V, with different D but same P

K	12 24 36 48 9a bc de f0 10 35 59 78 9b bc de f1
V	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
D	**ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 3f**
P	81 01 01 01 01 01 01 01 01 81
C	a6 46 3a 55 73 f6 13 bb dd 7f
T	c7 a8 06 c1 5a d1 40 50 62 59 7b 47 63 51 18 57
K'	12 24 36 48 9a bc de f0 10 35 59 78 9b bc de f1
V'	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
D'	**fe ff ff ff 19 35 bd 53 37 b8 eb 8c cd c5 bc a0 57 6b 21 fd dd 82 47 71 f9 46 f5 b6 21 72 7d fe 84 2c 9a 4a 1a**
P'	81 01 01 01 01 01 01 01 01 81
C'	a6 46 3a 55 73 f6 13 bb dd 7f
T'	c7 a8 06 c1 5a d1 40 50 62 59 7b 47 63 51 18 57

The same procedure can be also applied to the case of different key and initialization vector loaded state, i.e., $K \neq K'$ and $V \neq V'$, and therefore $S^{1536} \neq S'^{1536}$. Table 4 gives an example where $S^{1536} \neq S'^{1536}$, having two different associated data inputs D and D', respectively, which generate two identical associated data loaded states.

Table 4. Example of collision for different K, V and D with same P

K	**12 24 36 48 9a bc de f0 10 35 59 78 9b bc de f1**
V	**ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff**
D	**ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 3f**
P	81 01 01 01 01 01 01 01 01 81
C	a6 46 3a 55 73 f6 13 bb dd 7f
T	c7 a8 06 c1 5a d1 40 50 62 59 7b 47 63 51 18 57
K'	**10 20 36 40 9a bc de f0 10 35 59 78 9a be dd f0**
V'	**f0 f1 f0 f1 f0 f1 f0 f1 f0 f1 f0 f1 f0 f1 f0 f1**
D'	**f2 f1 c5 2f a3 cc 25 e5 d4 74 be 3b 54 8f d1 30 9d e2 6e ce e7 81 c9 ab 8f e3 db 0f c4 89 da ea 11 84 c0 28 18**
P'	81 01 01 01 01 01 01 01 01 81
C'	a6 46 3a 55 73 f6 13 bb dd 7f
T'	c7 a8 06 c1 5a d1 40 50 62 59 7b 47 63 51 18 57

The above-mentioned examples show that given the secret key, public initialization vector and at least 294 bits (37 bytes) of input associated data, a state collision can be forced against the authenticated encryption cipher ACORN. In practice a malicious sender who knows the secret key can find two distinct associated data strings which generate the same tag and therefore can break the integrity component of the cipher.

Note that, earlier we stated that there must exist state collision if the associated data length is more than 37 bits. However for our experiments we have used 294 bits of associated data to find the collision. This is because to find the collision after 37 bits of associated data loading, we need to manipulate the key and initialization vector with the 37 bits of associated data. In that case, the experiments need to go through 1536 clock of key and initialization vector loading phase which is infeasible. Therefore, we have used 294 bits of associated data to obtain state collisions in ACORN.

Collisions for Different Plaintexts. In this section we demonstrate collisions in the ACORN internal state for two distinct plaintext messages. The same technique used to obtain the collision during the associated data loading process can be also applied in the encryption phase to find a pair of input plaintext messages which results in identical plaintext loaded states. The resultant ciphertext will be different for this case but the generated tag after the final phase will be the same given that rest of the input plaintexts (if any) are the same once the colliding plaintext loaded states are obtained.

Plaintext bits are fed into the initial internal state of the keystream generator after $t = l + 2048$ iterations of initialization phase. Here, l denotes the length of the associated data. The initial internal states S^{l+2048} and S'^{l+2048} are formed using the key, initialization vector and associated data pair (K, V, D) and (K', V', D'), respectively. Suppose firstly that these two initial internal states S^{l+2048} and S'^{l+2048} are identical, i.e., $K = K'$, $V = V'$ and $D = D'$. Plaintexts P and P' are fed to the initial internal state S^{l+2048} and S'^{l+2048}, respectively. At each iteration of the plaintext loading, equations are generated relating the input plaintext P and P' with the contents of the corresponding states. To reduce the size of the equation system, we assume that the initial internal states S^{l+2048} and S'^{l+2048} are known. For a chosen input plaintext, P, calculations are then performed to determine the other plaintext, P' which will lead to the colliding plaintext loaded states. To obtain the collisions in the internal state the plaintext needs to be at least 294 bits since we need to map the plaintext inputs into all the internal state bits of the keystream generator. For the first 293 iterations of the encryption phase the input plaintexts are used to obtain two different states S^{l+2341} and S'^{l+2341} which differ only in the register stages numbered 0, 61, 107, 154, 193, 230 and 289. The collision is forced at the 294th iteration by having complementary values of input plaintexts P and P' for that time instant. This gives us two identical plaintext loaded states $S^{l+2342} = S'^{l+2342}$, for two distinct plaintext messages P and P', respectively.

Table 5 shows an example of obtaining state collision for two different input of plaintexts, P and P', with the same key $K = K'$, initialization vector $V = V'$ and associated data $D = D'$. As shown in the example, the resultant ciphertexts C and C' are different because the input plaintexts P and P' are different, however the tags T and T' generated after the final phase are the same, since no additional plaintexts are fed into the input after the colliding plaintext loaded states are obtained.

Table 5. Example of collision for same K, V and D with different P

K	12 24 36 48 9a bc de f0 10 35 59 78 9b bc de f1
V	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
D	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 3f
P	**51 75 65 65 6e 73 6c 61 6e 64 20 55 6e 69 76 65 72 73 69 74 79 20 6f 66 20 54 65 63 68 6e 6f 6c 6f 67 79 21 21**
C	**76 32 5e 31 1c 84 7e 9b e1 9a 9e ac 7f c9 01 fd 10 8c d7 88 ea 16 f3 72 bc 6d cb 9f bc 1e 96 58 02 bb 11 9a 51**
T	b5 83 b2 04 73 a5 35 e0 b4 df 98 1b 1f 40 08 9a
K'	12 24 36 48 9a bc de f0 10 35 59 78 9b bc de f1
V'	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
D'	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 3f
P'	**50 75 65 65 88 b9 2c 49 a6 23 f0 bb 7c 31 35 3d 85 8c 30 c5 f2 5d c5 d3 e9 49 cd c8 9a 4b 06 43 42 5e 20 3b 05**
C'	**77 32 5e 31 fa 4e 3e b7 29 dd 4e 52 6d 90 42 a5 e4 4e 86 6a 45 6b d1 b6 56 04 e3 a2 32 ba 9f 61 73 82 46 a1 75**
T'	b5 83 b2 04 73 a5 35 e0 b4 df 98 1b 1f 40 08 9a

For the example shown in Table 5, the two initial internal state pairs of ACORN are the same before the encryption phase. This is because for both pairs of internal state S^{l+2048} and S'^{l+2048}, we considered the same key, initialization vector and associated data input. Note that a collision can be still obtained in the encryption phase by using appropriate plaintext inputs even if any of these inputs are different, e.g., $K \neq K'$, $V \neq V'$ or $D \neq D'$, during the initialization phase. Table 6 shows a such example where the initial internal states are different, i.e., $S^{l+2048} \neq S'^{l+2048}$ and two collided plaintext loaded states are obtained by manipulating the plaintexts.

Table 6. Example of collision for same K, V with different D and P

K	12 24 36 48 9a bc de f0 10 35 59 78 9b bc de f1
V	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
D	**ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 3f**
P	**51 75 65 65 6e 73 6c 61 6e 64 20 55 6e 69 76 65 72 73 69 74 79 20 6f 66 20 54 65 63 68 6e 6f 6c 6f 67 79 21 21**
C	**76 32 5e 31 1c 84 7e 9b e1 9a 9e ac 7f c9 01 fd 10 8c d7 88 ea 16 f3 72 bc 6d cb 9f bc 1e 96 58 02 bb 11 9a 51**
T	b5 83 b2 04 73 a5 35 e0 b4 df 98 1b 1f 40 08 9a
K'	12 24 36 48 9a bc de f0 10 35 59 78 9b bc de f1
V'	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
D'	**aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa 2a**
P'	**62 af ff 7d fd 60 5b 3c f5 d7 73 72 50 8a 3e 10 a1 09 3f 05 61 a2 b7 a9 2e aa 56 90 63 8f 23 d0 7b ac ce d3 1a**
C'	**52 c5 0b 94 04 05 b2 06 39 0a 63 20 a7 49 5e 9c 60 9e 39 8c 06 b5 1d cf 18 5c 75 d1 4c c7 47 59 b5 84 64 a8 6a**
T'	b5 83 b2 04 73 a5 35 e0 b4 df 98 1b 1f 40 08 9a

As shown in Table 6, the key and initialization vector are the same for both states, i.e., $K = K'$ and $V = V'$. On the other hand both the associated data and the plaintext are different for both states S^t and S'^t, i.e., $D \neq D'$ and $P \neq P'$, and the state collision is forced by manipulating the plaintext. This means anyone having the knowledge of the secret key, e.g., a malicious sender, can forge both the associated data and the plaintext. For this instance, the changes in the associated data can be made arbitrarily, whereas changes in the plainext are determined by the equations that must be satisfied in order to obtain a collision. In this case, a malicious sender can apply a selective forgery attack on the associated data as he/she can select the associated data of his/her own choice.

Collisions for More than 294 Clocks. In this section we briefly discuss state collisions in ACORN if we manipulate more than 294 bits of input messages.

The examples shown in the previous sections are to obtain collided states after 294 iteration of the associated data or plaintext loading phase. Experiments are also performed to obtain a collision by manipulating more than 294 bits of input messages. As the number of iterations were increased the time to obtain the solutions increases. This is because when we increase the number of clocks there are more free variables in the equation system and the time complexity is expected to grow exponentially. However, the number of alternative inputs that can be used to obtain a collision doubles with the increase of each clock. Table 7 lists the required CPU time and memory for solving the equation system with 294 or more iterations of input message loading.

Table 7. Required resources to find a solution for the input message with more than 294 clocks

Number of Clocks	Number of Equations	CPU Time (Seconds)	Memory (KB)
294	586	3.57	127568
295	587	3.85	130872
296	588	4.19	140792
297	589	4.44	144828
298	590	4.68	145528
299	591	5.35	150748
300	592	5.58	152916
301	593	6.01	152792
302	594	6.44	152940
303	595	6.64	152828
304	596	6.89	152988
305	597	7.14	153396
306	598	7.40	153560
307	599	7.88	153744
308	600	8.21	153880
309	601	9.34	154196
310	602	11.18	167968

Even with a limited computational environment, the experiments were able to perform till 310 iterations of input message loading. The program runs out of allocated memory for experiments with 311 or more iterations.

4.3 Collisions for Unknown Internal State

In this section we discuss the feasibility of finding state collisions in ACORN when the internal state of the keystream generator is unknown. To do this, we defined the key and initialization vector loaded state S^{1536} and S'^{1536} in terms of unknown variables. For a chosen associated data, D, experiments are then conducted to determine the other associated data, D' which will generate two colliding associated data loaded states. However, it was found that the program

becomes very slow and runs out of computational resources when all the key and initialization vector loaded state bits are defined as variables.

So, instead of defining all the key and initialization vector loaded state bits as variables, we tried a guess and determine approach by defining some of the state bits as variable while keeping the remaining fixed. We started with defining one variable bit in the internal state of S^{1536} and S'^{1536}, while the remaining 292 bits are kept as fixed. Successive experiments were performed by increasing the number of unknown variables in the key and initialization vector loaded state. For this experiment, we have also applied the relabeling technique at each iteration of the associated data loading into the states S^{1536} and S'^{1536}. It was found that the experiment could be performed while there are less than 6 unknown variables in the key and initialization vector loaded state. For these attempts the required CPU time, memory, size of the equation system and maximum degree of the generated equations are given in Table 8.

Table 8. Required resources for solving the equation system for different number of unknown variable in the internal state of ACORN (with full relabeling)

Unknowns in the state	CPU Time (Seconds)	Memory (KB)	Number of equations	Number of variables	Max. degree
1	7.60	131348	879	880	2
2	7.80	139392	879	881	2
3	8.90	152948	879	882	2
4	16.22	320700	879	883	2
5	78.08	1585012	879	884	2

It can be seen that the complexity of solving the equation system is increasing as the number of unknown variable increases. The experiment seems to run out of memory when there are more than 5 unknown variables in the key and initialization vector loaded state. Table 8 shows that there is a significant increase in the required CPU time and memory when then number of unknown variables are increased to 5. This may be due to the limitations of the software. As given in Table 8, obtaining a solution with 5 unknown variables will require an exhaustive search to guess the contents of the other 288 register stages with a complexity of 2^{288}. The total complexity of the attack will be $2^{288} \times$ complexity of solving the equation system. This is worse than the exhaustive search on the key and therefore not a feasible approach.

Similar experiments are conducted to determine the feasibility of the above-mentioned experiment when no relabeling technique is applied while loading the associated data D into the internal state S^{1536}. In this case, relabeling technique is only applied while the associated data D' are being loaded to the internal state S'^{t}. For these attempts the required CPU time, memory, size of the equation system and maximum degree of the generated equations are given in Table 9.

Table 9. Required resources for solving the equation system for different number of unknown variable in the internal state of ACORN (with partial relabeling)

Unknowns in the state	CPU Time (Seconds)	Memory (KB)	Number of equations	Number of variables	Max. degree
1	3.46	128232	586	587	2
2	3.61	127732	586	588	2
3	4.20	131544	586	589	2
4	9.37	181712	586	590	3
5	16.38	251044	586	591	5
6	488.07	923560	586	592	5

The experiments were able to perform when there are less than 7 unknown variables in the internal state. Thus this will require an exhaustive search of 2^{286} over the internal state. The total complexity of this attack will be $2^{286} \times$ complexity of solving the equation system. The size of the generated equation system shown in Table 9 is small compare to the one given in Table 8. This is because relabeling is applied only when loading the associated data D' for the experiment results given in Table 9. However, this comes with the trade-off of an increase in the degree of the generated equation system, as indicated in Table 9. These comments also apply to the case for obtaining state collision by manipulating the plaintexts during the encryption phase of ACORN.

Experiments are also performed by representing the key bits as unknown variables, rather than defining the internal state bits as variables. For this case the experiment needs to go through a large 1536 iteration of key and initialization vector loading phase. Therefore, if there are too many unknown variables then the degree of the generated equations will be high and can reach up to the maximum degree.

On the other hand, relabeling 1536 iterations will keep the degree of the generated equations to quadratic, but this will introduce a large number of equations and variables in the system. Instead, we approach this without applying the relabeling during these 1536 iterations. One unknown variable is defined in the key space while the remaining are kept fixed. The experiment gives a solution when there is only one unknown variable in the key space. This is because the system of equation remains quadratic when there is only one unknown variable in the key. However, for more than one unknown variable the degree of the generated equations goes beyond quadratic and may reach up to the maximum degree. This makes it infeasible to obtain a solution when there are more than one unknown variables in the key space.

5 Conclusion

This paper identifies a weakness in ACORN. We have identified the phases during operation of the cipher where inputs to the state can be manipulated to

obtain state collisions. Our analysis shows that collision of the internal states can be forced by selecting complementary values in the external input messages if there exist two ACORN states which differ only in the values contained in seven particular register stages. The input bits that are manipulated can be either associated data bits or plaintext bits.

Experimental results show that for a chosen key and initialization vector, it is trivial to find two distinct messages which result in two ACORN states which differ only in the stages mentioned above. Given these states, a collision can be easily forced in the internal state of ACORN.

In our experiments, we assumed that the key is known. This is not necessarily true for an external attacker. However, the sender has access to the secret key and allowing the sender to create collisions is not a desirable property either. For ACORN, the sender can find a collision for any given key, initialization vector and input message combination, thereby compromising the integrity component of the cipher. A malicious sender can manipulate either the associated data or the plaintext messages to obtain the collision of the tag and therefore can frame a forged message accepted as legitimate.

Finally we note that the collisions found in ACORN are key dependent, thus obtaining a set of colliding input message must leak information about the key. We aim to investigate methods for exploiting this leakage in our future work.

Acknowledgements. Josef Pieprzyk was supported by Polish National Science Centre grant DEC-2014/15/B/ST6/05130. Md Iftekhar Salam was supported by the QUT Postgraduate Research Award (QUTPRA), QUT Higher Degree Research Tuition Fee Sponsorship and QUT Excellence Top Up Scholarship.

References

1. Wu, H., ACORN: A Lightweight Authenticated Cipher (v1). CAESAR Competition. Retrieved from http://competitions.cr.yp.to/round1/acornv1.pdf, Accessed 29 May 2015.
2. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Available from: http://competitions.cr.yp.to/index.html, Accessed 10 September 2015.
3. Wu, H., ACORN: A Lightweight Authenticated Cipher (v2). CAESAR Competition. Retrieved from http://competitions.cr.yp.to/round2/acornv2.pdf, Accessed 10 September 2015.
4. Liu, M. and Lin, D., Cryptanalysis of Lightweight Authenticated Cipher Acorn. Cryptographic Competitions Mailing List. Retrieved from https://groups.google.com/forum/#!topic/crypto-competitions/2mrDnyb9hfM, Accessed 29 May 2015.
5. Chaigneau C., Fuhr T., and H.., G., Full Key-Recovery on ACORN in Nonce-Reuse and Decryption-Misuse Settings. Cryptographic Competitions Mailing List. Retrieved from https://groups.google.com/forum/#!topic/crypto-competitions/RTtZvFZay7k, Accessed 10 August 2015.
6. Stein, W., et al., Sage Mathematics Software (Version 6.4.1), The Sage Development Team, 2015, http://www.sagemath.org.