The 23rd Annual International Conference on Information Security and Cryptology

ICISC 2020

December 2 (Wed) ~ December 4 (Fri), 2020 | Virtual Conference

Hosted by
Korea Institute of Information Security and Cryptology (KIISC)
National Security Research Institute (NSR)

Korea Institute of Information Security & Cryptology

Differential Fault based Key Recovery Attacks on TRIAD

Iftekhar Salam, Kim Young Law,
Luxin Xue, Wei-Chuen Yau

Dept. of Information and Communication Technology,
Xiamen University Malaysia

Outline

- Introduction
- Fault Attack
- TRIAD
- Fault Attack on TRIAD
- Results
- Conclusion

Introduction

- TRIAD – first round candidate of the NIST LWC
- Standardize lightweight cryptographic algorithms.
- TRIAD - an authenticated encryption (AE) stream cipher algorithm.
- Inspired by TRIVIUM, increases the security level to 112 bits.

Fault Attack

Differential fault attack(DFA)

Using XOR difference of fault free keystream and faulty keystream to recover the initial internal states.

- Number of faults: single-bit, multibyte.
- Fault type: bit-flipping, set to zero/one, random fault.
- Fault duration: temporary fault, permanent fault.

TRIAD

TRIADv1 – a family of lightweight AE stream cipher.

(TRIAD-AE + TRIAD-HASH)

TRIAD-AE – An authenticated encryption mode.

TRIAD-SC and TRIAD-MAC

128-bit key, $K \in \{0,1\}^{128}$ and 96-bit nonce, $N \in \{0,1\}^{96}$

Consist of three non-linear feedback shift registers.

$$a_i^{t+1} = \begin{cases} f_3^t \oplus msg^t & \text{for } i = 0 \\ a_{i-1}^t & \text{for } i = 1, \cdots, 79 \end{cases}$$

$$b_i^{t+1} = \begin{cases} f_1^t \oplus msg^t & \text{for } i = 0 \\ b_{i-1}^t & \text{for } i = 1, \cdots, 87 \end{cases}$$

$$c_i^{t+1} = \begin{cases} f_2^t \oplus msg^t & \text{for } i = 0 \\ c_{i-1}^t & \text{for } i = 1, \cdots, 87 \end{cases}$$

Input:

An arbitrary length byte-array plaintext M and an arbitrary length byte array for associated data A

Output: $z_t = a_{67}^t \oplus a_{79}^t \oplus b_{63}^t \oplus b_{87}^t \oplus c_{67}^t \oplus c_{87}^t \oplus b_{84}^t c_{84}^t$

Ciphertext $C \in \{0,1\}^{l_m}$, where l_m is the length of original plaintext M

Fault Attack on TRIAD

Theoretical observations → Attacks Implementation → Verification

- Algebraic Normal Form

- Bit-flipping Fault
- Random Fault

- Simulation

Fault Attack on TRIAD – Observations

Algebraic Normal Form (ANF) of the Keystream Function

- Generate a set of keystream equations
- Identify specific patterns

Initial internal states of TRIAD $S = \{s_0, \dots, s_{255}\}$

Initial internal states of registers a^t, b^t, c^t at time $t = 1024$

$$S = (s_0, \dots, s_{79}, s_{80}, \dots, s_{167}, s_{168}, \dots, s_{255}) = (a_0^{1024}, \dots, a_{79}^{1024}) \| (b_0^{1024}, \dots, b_{87}^{1024}) \| (c_0^{1024}, \dots, c_{87}^{1024})$$

Using Sage to generate the keystream z_t for different time instant t

Fault Attack on TRIAD - Observations

Example of the ANF of the first five keystream equations

$$z_0 = s_{164}s_{252} \oplus s_{67} \oplus s_{79} \oplus s_{143} \oplus s_{167} \oplus s_{235} \oplus s_{255}$$
$$z_1 = s_{163}s_{251} \oplus s_{66} \oplus s_{78} \oplus s_{142} \oplus s_{166} \oplus s_{234} \oplus s_{254}$$
$$z_2 = s_{162}s_{250} \oplus s_{65} \oplus s_{77} \oplus s_{141} \oplus s_{165} \oplus s_{233} \oplus s_{253}$$
$$z_3 = s_{161}s_{249} \oplus s_{64} \oplus s_{76} \oplus s_{140} \oplus s_{164} \oplus s_{232} \oplus s_{252}$$
$$z_4 = s_{160}s_{248} \oplus s_{63} \oplus s_{75} \oplus s_{139} \oplus s_{163} \oplus s_{231} \oplus s_{251}$$

Equations have a clear pattern :

- Quadratic terms(unique) Bit-flipping
- Linear terms(some of them are unique) Random fault

The first 160 keystreams are analyzed to identify equations consisting of such unique patterns.

Bit-flipping Fault Attack on TRIAD

Assumptions

- An adversary has access to the first 157 bits of the keystream.
- An adversary can complement the bit in a specific target register.

Faulty keystream $z'_{0,s_{252}} = s_{164}\overline{s_{252}} \oplus s_{67} \oplus s_{79} \oplus s_{143} \oplus s_{167} \oplus s_{235} \oplus s_{255}$

Faulty state

XORing fault free and faulty keystreams $z_0 \oplus z'_{0,s_{252}} = s_{164}(s_{252} \oplus \overline{s_{252}})$

$= s_{164}$ Can be recovered

Similarly, applying bit-flipping fault in s_{164} can recover s_{252}

Bit-flipping Fault Attack on TRIAD

Table 1. Useful equation patterns to recover $s_{80}, \ldots, s_{164}$ and $s_{168}, \ldots, s_{254}$ by injecting faults at registers $s_{168}, \ldots, s_{252}$

Fault target, s_j	Required faulty keystream, z'_{i,s_j}	Recovered bit
s_{252}	$z'_{0,s_{252}}$ $z'_{82,s_{252}}$	s_{164} s_{242}
s_{251}	$z'_{1,s_{251}}$ $z'_{83,s_{251}}$	s_{163} s_{241}
$\vdots$	$\vdots$	$\vdots$
s_{246}	$z'_{6,s_{246}}$ $z'_{88,s_{246}}$	s_{158} s_{236}
s_{245}	$z'_{7,s_{245}}$ $z'_{89,s_{245}}$	s_{157} s_{235}
s_{244}	$z'_{8,s_{244}}$ $z'_{90,s_{244}}$ $z'_{68,s_{244}}$	s_{156} s_{234} s_{254}
s_{243}	$z'_{9,s_{243}}$ $z'_{91,s_{243}}$ $z'_{69,s_{243}}$	s_{155} s_{233} s_{253}
$\vdots$	$\vdots$	$\vdots$
s_{179}	$z'_{73,s_{179}}$ $z'_{155,s_{179}}$ $z'_{133,s_{179}}$	s_{91} s_{169} s_{189}
s_{178}	$z'_{74,s_{178}}$ $z'_{156,s_{178}}$ $z'_{134,s_{178}}$	s_{90} s_{168} s_{188}
s_{177}	$z'_{75,s_{177}}$ $z'_{135,s_{177}}$	s_{89} s_{187}
s_{176}	$z'_{76,s_{176}}$ $z'_{136,s_{176}}$	s_{88} s_{186}
$\vdots$	$\vdots$	$\vdots$
s_{169}	$z'_{83,s_{169}}$ $z'_{143,s_{169}}$	s_{81} s_{179}
s_{168}	$z'_{84,s_{168}}$ $z'_{144,s_{168}}$	s_{80} s_{178}

Bit-flipping Fault Attack on TRIAD

Algorithm 1: Bit-flipping fault attack on TRIAD

Input: Fault target location(s): $s_{168}, \cdots, s_{252}$
Output: Initial state bits: $s_{80}, \cdots, s_{164}$, and $s_{168}, \cdots, s_{254}$

1 Initialise with the K and N
2 Generate and store the first 157 bits of the fault-free keystream $z_0, \cdots, z_{156}$
3 **for** $j \leftarrow 252$ **to** 168 **do**
4 Re-initialise the cipher with the K, N
5 Inject bit-flipping fault to s_j
6 Compute the faulty keystream z'_{252-j,s_j}
7 Output $s_{j-88} = z_{252-j,s_j} \oplus z'_{252-j,s_j}$ — $s_{80}, \ldots, s_{164}$
8 **if** $j \geq 178$ **then**
9 Compute the faulty keystream z'_{334-j,s_j}
10 Output $s_{j-10} = z_{334-j,s_j} \oplus z'_{334-j,s_j}$ — $s_{168}, \ldots, s_{242}$
11 **if** $j \geq 233$ *and* $j \leq 244$ **then**
12 Compute the faulty keystream z'_{312-j,s_j}
13 Output $s_{j+10} = z_{312-j,s_j} \oplus z'_{312-j,s_j}$ — $s_{243}, \ldots, s_{254}$

Bit-flipping Fault Attack on TRIAD

Theoretical result:

- The patterns in the first 160 keystream equations ⟶ recover all the initial state bits of TRAID (except s_{79}, s_{167}, s_{255})
- Requiring 253 faults.

Improving the process to minimize the number of faults:

- Fault injections (target only register $c_0^{1024}, \dots, c_{87}^{1024}$)
  - Recovering $s_{80}, \dots, s_{164}$ and $s_{168}, \dots, s_{254}$, which refers to $b_0^{1024}, \dots, b_{84}^{1024}$ and $c_0^{1024}, \dots, c_{86}^{1024}$, respectively.
- Solving equations
  - To recover the remaining bits $s_0, \dots, s_{79}, s_{165}, \dots, s_{167}$ and s_{255}

Random Fault Attack on TRIAD

Assumptions

- An adversary has access to the first 157 bits of the keystream.
- An adversary can inject a random fault several times in a specific target register.

Faulty keystream $z'_{3,s_{252}} = s_{161}s_{249} \oplus s_{64} \oplus s_{76} \oplus s_{140} \oplus s_{164} \oplus s_{232} \oplus s'_{252}$ Faulty state

XORing fault free and faulty keystreams

$$\begin{aligned} z'_{3,s_{252}} \oplus z_3 &= s_{252} \oplus s'_{252} \\ &= s_{252} \oplus s_{252} \oplus e_{252} \\ &= e_{252} \end{aligned}$$

Fault value

- $e_{252} = 0$, the fault did not complement register bit s_{252}
- $e_{252} = 1$, the fault has complemented register bit s_{252} ⟶ Recover s_{164}

Random Fault Attack on TRIAD

Table 2. Unique linear terms in the equations to identify the fault value e_j

Fault Target, s_j	Required faulty keystream, z'_{i,s_j}	Recovered faulty value, e_j
s_{252}	$z'_{3,s_{252}}$	e_{252}
s_{251}	$z'_{4,s_{251}}$	e_{251}
s_{250}	$z'_{5,s_{250}}$	e_{250}
$\vdots$	$\vdots$	$\vdots$
s_{169}	$z'_{86,s_{169}}$	e_{169}
s_{168}	$z'_{87,s_{168}}$	e_{168}

Random Fault Attack on TRIAD

Algorithm 2: Random fault attack on TRIAD

Input: Fault target location(s): $s_{168}, \cdots, s_{252}$
Output: Initial state bits: $s_{80}, \cdots, s_{164}$, and $s_{168}, \cdots, s_{254}$

1 Initialise with the K and N
2 Generate and store the first 157 bits of the fault-free keystream $z_0, \cdots, z_{156}$
3 **for** $j \leftarrow 252$ **to** 168 **do**
4 Re-initialise the cipher with the K, N
5 Inject a random fault to s_j
6 Compute the faulty keystream z'_{255-j,s_j}
7 **if** $z'_{255-j,s_j} \oplus z_{255-j,s_j} = 0$ **then**
8 Go back to step 4 and repeat
9 **else**
10 Output $s_{j-88} = z_{252-j,s_j} \oplus z'_{252-j,s_j}$
11 **if** $j \geq 178$ **then**
12 Compute the faulty keystream z'_{334-j,s_j}
13 Output $s_{j-10} = z_{334-j,s_j} \oplus z'_{334-j,s_j}$
14 **if** $j \geq 233$ *and* $j \leq 244$ **then**
15 Compute the faulty keystream z'_{312-j,s_j}
16 Output $s_{j+10} = z_{312-j,s_j} \oplus z'_{312-j,s_j}$

$$z_0 = s_{164}s_{252} \oplus s_{67} \oplus s_{79} \oplus s_{143} \oplus s_{167} \oplus s_{235} \oplus s_{255}$$
$$z_1 = s_{163}s_{251} \oplus s_{66} \oplus s_{78} \oplus s_{142} \oplus s_{166} \oplus s_{234} \oplus s_{254}$$
$$z_2 = s_{162}s_{250} \oplus s_{65} \oplus s_{77} \oplus s_{141} \oplus s_{165} \oplus s_{233} \oplus s_{253}$$
$$z_3 = s_{161}s_{249} \oplus s_{64} \oplus s_{76} \oplus s_{140} \oplus s_{164} \oplus s_{232} \oplus s_{252}$$
$$z_4 = s_{160}s_{248} \oplus s_{63} \oplus s_{75} \oplus s_{139} \oplus s_{163} \oplus s_{231} \oplus s_{251}$$

Example:
When $j = 252$, z_3 is used to identify the e_{252}
Fault value $e_{252} = 0$
The fault did not complement register bit s_{252}

Fault value $e_{252} = 1$
The fault has complemented register bit s_{252}
s_{164} can be recovered by z_0.

Recovering the Remaining State Bits

Fault injections (target only register $c_0^{1024}, \ldots, c_{87}^{1024}$)

The register bits $s_0, \ldots, s_{79}, s_{165}, \ldots, s_{167}$ and s_{255} are not recovered directly using the faults.

Substituting the values recovered using fault attacks ⟶ Reduce the degree of the output

Compute Gröbner bases ⟶ Solve low degree equations

Enable an adversary to recover all the initial state bits of TRIAD.

Verified via simulations

Comparison of Bit-flipping and Random Fault

Table 3. Comparison of bit-flipping and random fault attacks on TRIAD

Fault type	Total Number of required faults	Data complexity	Nonce reuse
Bit-flipping	85	$2^{7.43}$	$2^{6.41}$
Random	170	$2^{8.01}$	$2^{7.41}$

The experiment is repeated multiple times with 1,000 random keys and nonces

Update function of TRIAD is bijective ⟶ a state recovery ⟶ a key recovery

Conclusion

- Bit-flipping fault attack requires 85 faults to recover the secret key of TRIAD.
- Random fault attack requires 170 faults to recover the secret key.
- Bijective ⟶ State recovery = Key recovery.

Thank You for Watching

Q&A