Lightweight Cryptography Term Paper

PHOTON-Beetle
Authenticated Encryption and Hash Family

Ajay Tarole

Indian Institute Of Technology, Bhilai, ajayt@iitbhilai.ac.in.

Abstract. This term paper reviews PHOTON-BEETLE, a crypto method which has reached Round 2 of the NIST competition for light-weight cryptography. This document also focuses on the security analysis of PHOTON-BEETLE. It uses a sponge authenticated encryption and sponge hash, and has a 256-bit permutation. This paper is organised into different sections starting with a brief description of PHOTON-BEETLE specifications and then discusses various attacks on photon beetle permutation.

Keywords: PHOTON beetle family

Contents

1 PHOTON-Beetle

In this document, we propose PHOTON-Beetle, an authenticated encryption and hash family, that uses a sponge-based mode Beetle with the P256 (used for the PHOTON hash [6]) being the underlying permutation. We denote this permutation by PHOTON 256 . Based on the functionalities, PHOTON-Beetle can be classified into two categories: a family of authenticated encryptions, dubbed as PHOTON-Beetle-AEAD and a family of hash functions, dubbed as PHOTON-Beetle-Hash. Both these families are parameterized by r, the rate of message absorption.

1.1 Notations

{0,1}*	Set of all strings
$\{0,1\}^n$	Set of strings of length n
$\|A\|$	Number of the bits in the string A
$A\|\|B$	The concatenation of A and B
$V_1\|\|\cdots\|\|V_n \xleftarrow{(a_1,\cdots,a_v)} V$	Parsing of the string V into v vectors of size $a_1, \cdots a_v$
ε ? $a : b$	Evaluates to a if ε holds and b otherwise
(ε_1 and ε_2) ? $a : b : c : d$	Evaluates to a if both ε_1 and ε_2 holds, b if only ε_1 holds, c if only ε_2 holds and d otherwise
$m\|n$	m divides n
X[i,j]	The element at i-th row and j-th column of X

1.2 Functions

$Trunc(V, i)$ is a function that returns the most significant i bits of the V and Ozs is the function that applies 10* padding on r bits, i.e. $Ozs_r(V) = V||1||0^{r-|V|-1}$ when $|V| < r$.

1.3 $PHOTON_{256}$ Permutation

We use $PHOTON_{256}$ as the underlying 256-bit permutation in our mode. It is applied on a state of 64 elements of 4 bits each, which is represented as a (8 $\times$ 8) matrix X. $PHOTON_{256}$ is composed of 12 rounds, each containing four layers AddConstant, SubCells, ShiftRows and MixColumnSerial. Informally, AddConstant adds fixed constants to the cells of the internal state. SubCells applies an 4-bit S-Box to each of the 64 4-bit cells. ShiftRows rotates the position of the cells in each of the rows and MixColumnSerial linearly mixes all the columns independently using a serial matrix multiplication. The multiplication with the coefficients in the matrix is in $GF(2^4)$ with $x^4 + x + 1$ being the irreducible polynomial.

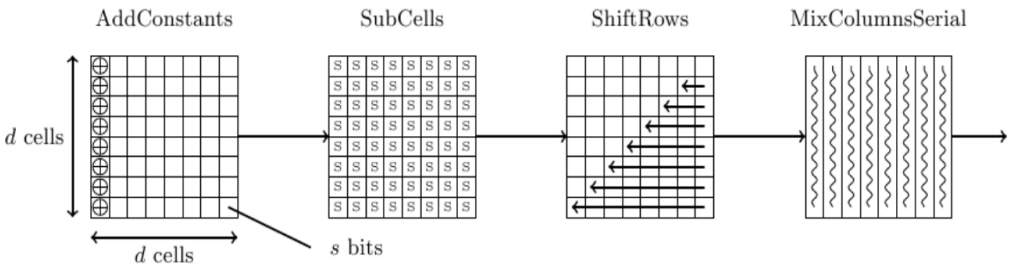

Figure 1: One Round of PHOTON Internal Permutation

PHOTON$_{256}(X)$

1 : for $i = 0$ to 11 :
2 : $X \leftarrow \mathsf{AddConstant}(X, i)$;
3 : $X \leftarrow \mathsf{SubCells}(X)$;
4 : $X \leftarrow \mathsf{ShiftRows}(X)$;
5 : $X \leftarrow \mathsf{MixColumnSerial}(X)$;
return X;

AddConstant(X, k)

1 : $RC[12] \leftarrow \{1, 3, 7, 14, 13, 11, 6, 12, 9, 2, 5, 10\}$;
2 : $IC[8] \leftarrow \{0, 1, 3, 7, 15, 14, 12, 8\}$;
3 : for $i = 0$ to 7 :
4 : $X[i, 0] \leftarrow X[i, 0] \oplus RC[k] \oplus IC[i]$;
return X;

SubCells(X)

1 : for $i = 0$ to $7, j = 0$ to 7 :
2 : $X[i, j] \leftarrow \text{S-Box}(X[i, j])$;
return X;

ShiftRows(X)

1 : for $i = 0$ to $7, j = 0$ to 7 :
2 : $X'[i, j] \leftarrow X[i, (j + i)\%8])$;
return X';

MixColumnSerial(X)

1 : $M \leftarrow \mathsf{Serial}[2, 4, 2, 11, 2, 8, 5, 6]$;
2 : $X \leftarrow M^8 \odot X$;
return X;

Figure 2: $PHOTON_{256}$ Permutation.

1.3.1 The PHOTON S-box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S-box	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

S-box have some propeties:

(1) Differential Branch Number(DBN)=3. (2) Linear Branch Number(LBN) = 2.
(3) Max degree = 3.
(4) Min degree = 2.
(5) Maximal difference probablity = $1/2^{-2} = 1/4 = 0.25$.
(6) Albebraic Normal Form(ANF)

Main componets function
y3 = x0*x1*x2 + x0*x1*x3 + x0*x2*x3 + x0 + x1*x2 + x1 + x3 + 1
y2 = x0*x1*x3 + x0*x1 + x0*x2*x3 + x0*x3 + x1*x3 + x2 + x3 + 1
y1 = x0*x1*x2 + x0*x1*x3 + x0*x2*x3 + x1*x3 + x1 + x2*x3 + x3
y0 = x0 + x1*x2 + x2 + x3

All componets function
y15 = x0*x1*x3 + x0*x1 + x0*x2*x3 + x0*x3 + x2*x3
y14 = x0*x1*x3 + x0*x1 + x0*x2*x3 + x0*x3 + x0 + x1*x2 + x2*x3 + x2 + x3
y13 = x0*x1*x2 + x0*x1 + x0*x3 + x1*x3 + x1 + x3
y12 = x0*x1*x2 + x0*x1 + x0*x3 + x0 + x1*x2 + x1*x3 + x1 + x2
y11 = x1*x3 + x2*x3 + x2 + x3 + 1, y10 = x0 + x1*x2 + x1*x3 + x2*x3 + 1
y9 = x0*x1*x2 + x0*x1*x3 + x0*x2*x3 + x1 + x2 + 1
y8 = x0*x1*x2 + x0*x1*x3 + x0*x2*x3 + x0 + x1*x2 + x1 + x3 + 1
y7 = x0*x1*x2 + x0*x1 + x0*x3 + x0 + x1*x2 + x1 + x2*x3 + x3 + 1
y6 = x0*x1*x2 + x0*x1 + x0*x3 + x1 + x2*x3 + x2 + 1
y5 = x0*x1*x3 + x0*x1 + x0*x2*x3 + x0*x3 + x0 + x1*x2 + x1*x3 + 1
y4 = x0*x1*x3 + x0*x1 + x0*x2*x3 + x0*x3 + x1*x3 + x2 + x3 + 1
y3 = x0*x1*x2 + x0*x1*x3 + x0*x2*x3 + x0 + x1*x2 + x1*x3 + x1 + x2*x3 + x2
y2 = x0*x1*x2 + x0*x1*x3 + x0*x2*x3 + x1*x3 + x1 + x2*x3 + x3
y1 = x0 + x1*x2 + x2 + x3
y0 = 0

1.4 Algorithms

(Run photon_hash.c to compute Hash value for any massage.)

PHOTON-Beetle-AEAD.ENC$[r](K, N, A, M)$

$\mathsf{IV} \leftarrow N \| K;\ C \leftarrow \lambda;$
if $A = \lambda, M = \lambda$:
  $T \leftarrow \mathsf{TAG}_{128}(\mathsf{IV} \oplus 1);$ **return** $(\lambda, T);$
$c_0 \leftarrow (M \neq \lambda$ **and** $r|\ |A|)?\ 1:2:3:4$
$c_1 \leftarrow (A \neq \lambda$ **and** $r|\ |M|)?\ 1:2:5:6$
if $A \neq \lambda$:
  $\mathsf{IV} \leftarrow \mathsf{HASH}_r(\mathsf{IV}, A, c_0);$
if $M \neq \lambda$:
  $M_1 \| \cdots \| M_m \xleftarrow{r} M;$
  for $i = 1$ **to** m :
    $Y \| Z \xleftarrow{(r,256-r)} \mathtt{PHOTON}_{256}(\mathsf{IV});$
    $(W, C_i) \leftarrow \rho(Y, M_i);$
    $\mathsf{IV} \leftarrow W \| Z;$
  $\mathsf{IV} \leftarrow \mathsf{IV} \oplus c_1;$
  $C \leftarrow C_1 \| \cdots \| C_m;$
$T \leftarrow \mathsf{TAG}_{128}(\mathsf{IV});$
return $(C, T);$

PHOTON-Beetle-AEAD.DEC$[r](K, N, A, C, T)$

$\mathsf{IV} \leftarrow N \| K;\ M \leftarrow \lambda;$
if $A = \lambda, C = \lambda$:
  $T^\star \leftarrow \mathsf{TAG}_{128}(\mathsf{IV} \oplus 1);$
  return $(T = T^\star)?\ \lambda : \perp;$
$c_0 \leftarrow (C \neq \lambda$ **and** $r|\ |A|)?\ 1:2:3:4$
$c_1 \leftarrow (A \neq \lambda$ **and** $r|\ |C|)?\ 1:2:5:6$
if $A \neq \lambda$:
  $\mathsf{IV} \leftarrow \mathsf{HASH}_r(\mathsf{IV}, A, c_0);$
if $C \neq \lambda$:
  $C_1 \| \cdots \| C_m \xleftarrow{r} C;$
  for $i = 1$ **to** m :
    $Y \| Z \xleftarrow{(r,256-r)} \mathtt{PHOTON}_{256}(\mathsf{IV});$
    $(W, M_i) \leftarrow \rho^{-1}(Y, C_i);$
    $\mathsf{IV} \leftarrow W \| Z;$
  $\mathsf{IV} \leftarrow \mathsf{IV} \oplus c_1;$
  $M \leftarrow M_1 \| \cdots \| M_m;$
$T^\star \leftarrow \mathsf{TAG}_{128}(\mathsf{IV});$
return $(T = T^\star)?\ M : \perp;$

PHOTON-Beetle-Hash$[r](M)$

if $M = \lambda$:
  $\mathsf{IV} \leftarrow 0 \| 0;$
  $T \leftarrow \mathsf{TAG}_{256}(\mathsf{IV} \oplus 1);$ **return** $T;$
if $|M| \leq 128$:
  $c_0 \leftarrow (|M| < 128)?\ 1:2$
  $\mathsf{IV} \leftarrow \mathsf{Ozs}_{128}(M) \| 0;$
  $T \leftarrow \mathsf{TAG}_{256}(\mathsf{IV} \oplus c_0);$ **return** $T;$
$M_1 \| M' \xleftarrow{(128,|M|-128)} M;$
$c_0 \leftarrow (r|\ |M'|)?\ 1:2$
$\mathsf{IV} \leftarrow M_1 \| 0$
$\mathsf{IV} \leftarrow \mathsf{HASH}_r(\mathsf{IV}, M', c_0);$
$T \leftarrow \mathsf{TAG}_{256}(\mathsf{IV});$
return $T;$

$\mathsf{HASH}_r(\mathsf{IV}, D, c_0)$

$D_1 \| \cdots \| D_d \xleftarrow{r} \mathsf{Ozs}_r(D);$
for $i = 1$ **to** d :
  $Y \| Z \xleftarrow{(r,256-r)} \mathtt{PHOTON}_{256}(\mathsf{IV});$
  $W \leftarrow Y \oplus D_i;$
  $\mathsf{IV} \leftarrow W \| Z;$
$\mathsf{IV} \leftarrow \mathsf{IV} \oplus c_0;$
return $\mathsf{IV};$

$\mathsf{TAG}_\tau(T_0)$

for $i = 1$ **to** $\lceil \tau/128 \rceil$:
  $T_i \leftarrow \mathtt{PHOTON}_{256}(T_{i-1});$
$T \leftarrow \mathsf{Trunc}(T_1, 128) \ \| \cdots \| \ \mathsf{Trunc}(T_{\tau/128}, 128)$
return $T;$

Figure 3: Formal Specification of PHOTON-Beetle-AEAD [r] := (PHOTON-Beetle-AEAD.ENC [r], PHOTON-Beetle-AEAD.DEC [r]) authenticated encryption and PHOTON-Beetle-Hash[r] hash mode.

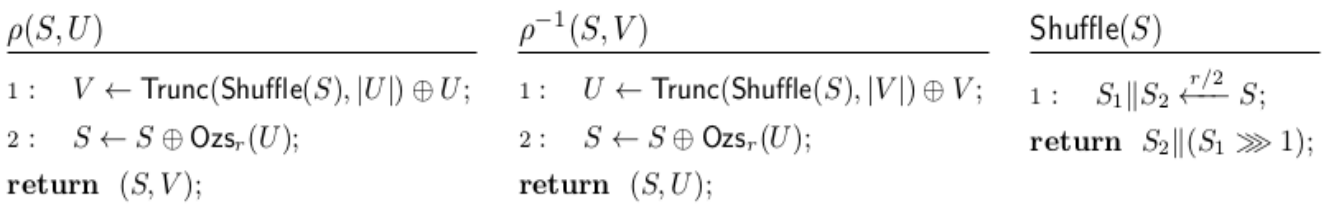

Figure 4: Mathematical Component ρ and ρ^{-1}

1.5 Mathematical Component ρ and ρ^{-1}

ρ is a linear function that receives as input a state $S \in \{0,1\}^r$ and an input data $U \in \{0,1\}^{\leq r}$. It produces an output data $V \in \{0,1\}^{|U|}$ using the simple xor operation of the shuffled state and the input data (padded with zeros to an r bit block), and then updates the state S by xoring it with the input data. By shuffled state, we mean shuffling of the state bits in some order such that both $S \rightarrow Shuffle(S)$ and $S \rightarrow Shuffle(S) \oplus S$ are linear functions with rank r and r 1 respectively. ρ^{-1} is the inverse function of ρ, which takes the state S and the output data V to reproduce the input data U and update the state.

1.6 PHOTON-Beetle-AEAD Authenticated Encryption

PHOTON-Beetle-AEAD.ENC[r] authenticated encryption takes an encryption key $K \in \{0,1\}^{128}$, a nonce $N \in \{0,1\}^{128}$, an associated data $A \in \{0,1\}^*$ and a message $M \in \{0,1\}^*$ as inputs and returns a ciphertext $C \in 0,1^{|M|}$ and a tag $T \in \{0,1\}^{128}$. Corresponding decryption algorithm PHOTON-Beetle-AEAD.DEC[r] takes a key $K \in \{0,1\}^{128}$, a nonce $N \in 0,1^{128}$, an associated data $A \in \{0,1\}^*$, a ciphertext $C \in \{0,1\}^*$ and a tag $T \in \{0,1\}^{128}$ as inputs and returns the plaintext $M \in \{0,1\}^{|C|}$ corresponding to C if the tag T is verified. The parameter r signifies the rate of message absorption.

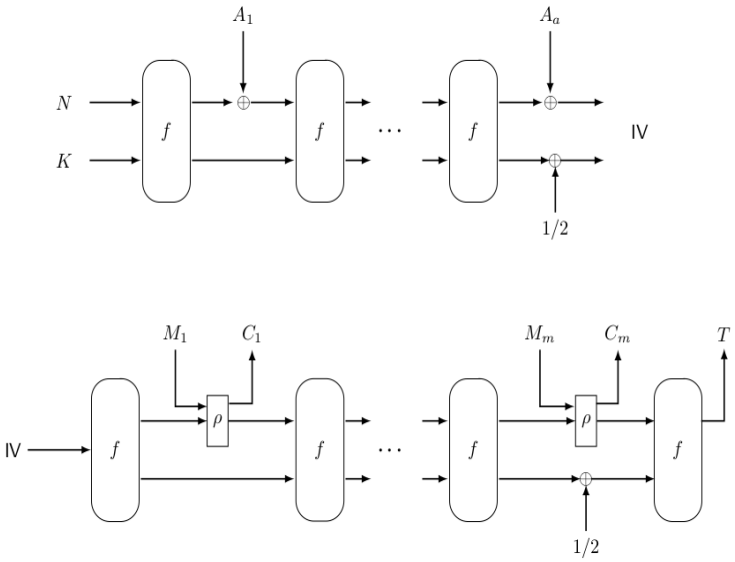

Figure 5: PHOTON-Beetle-AEAD.ENC with a AD blocks and m message blocks.

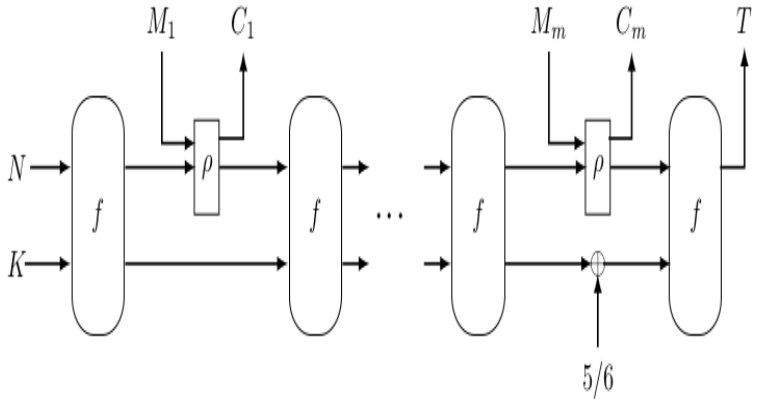

Figure 6: PHOTON-Beetle-AEAD.ENC with empty AD and m message blocks.

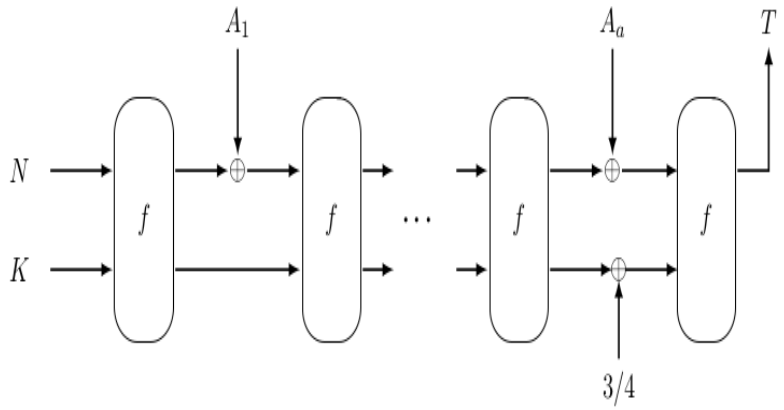

Figure 7: PHOTON-Beetle-AEAD.ENC Construction with a AD blocks and empty message.

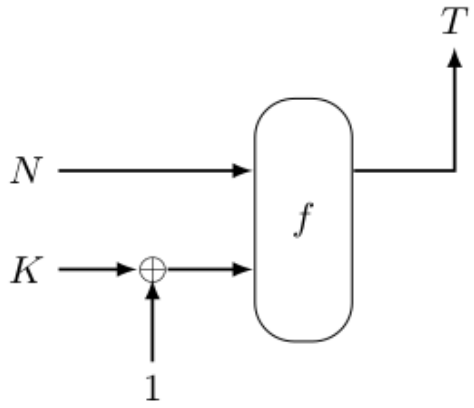

Figure 8: PHOTON-Beetle-AEAD.ENC Construction with empty AD and empty message.

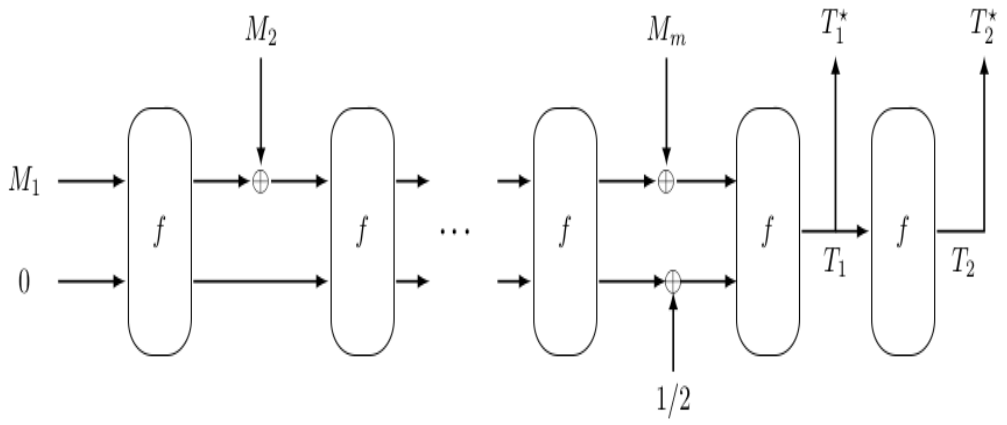

Figure 9: PHOTON-Beetle-Hash with m message blocks. Here $|M_1| = 128, |M_i| = r$, for $i = 2, \cdots, m-1$ and $|M_m| \leq r$. The tag T is computed as $T_1 * ||T_2*$, where $T_i = Trunc(T_i, 128)$.

1.7 PHOTON-Beetle-Hash Hash function

PHOTON-Beetle-Hash takes a message $M \in \{0,1\}^*$ and generates a tag $T \in \{0,1\}^{256}$. We first parse the message into 128-bit block (the first block) followed by r-bit blocks. The 256-bit tag is squeezed into 2 parts of 128 bits each.

1.8 Recommended Versions

1.8.1 Authenticated Encryption Family

Our recommended versions for authenticated encryption with associated data are:

1. PHOTON-Beetle-AEAD[128]. This is our primary AEAD member. This design aims to be implemented with low hardware footprint yet with high throughput. Here we keep the rate of absorption of this cipher to be r = 128.

2. PHOTON-Beetle-AEAD[32]. This is another AEAD member that aims to be implemented with extremely low hardware footprint without giving much importance to the throughput. Hence, we keep the rate of absorption of this cipher to only r = 32.

1.8.2 Hash Function Family

Our recommended version for hash function is:

1. PHOTON-Beetle-Hash[32]. This is our only recommended Hash. The hash function absorbs the first 128 bits of plaintext as the initial vector and successive rate of absorption is kept to r = 32 bits. This design also aims to be implemented with extremely low hardware footprint and it is in particular has excellent throughput and energy efficiency for smaller messages. Note that, for any plaintext of size less

than or equal to 128 bits, the hash function requires only 1 primitive call to process the message along with the two additional calls require to generate the hash value.

1.8.3 Combined AEAD and Hash Function Family

Based on our recommendations, we pair the following that provide both AEAD and hashing functionality. 1. PHOTON-Beetle-AEAD[32] + PHOTON-Beetle-Hash[32]. Both these AEAD and Hash operate on a 256-bit state, follow the sponge mode and use PHOTON 256 as the underlying permutation with the same rate of data absorption (i.e. r = 32). The associated data process phase in PHOTON-Beetle-AEAD[32] is exactly the same as the message process phase of PHOTON-Beetle-Hash[32]. PHOTON-Beetle-Hash[32] with input $X := X_1||X_0$, where $X_1 \in \{0,1\}^{128}$, functions exactly in the similar way as PHOTON-Beetle-AEAD[32] with $N = X_1, K = 0^{128}, A = X'$ and $M = \lambda$ except the fact that PHOTON-Beetle-Hash[32] makes an ad- ditional call to PHOTON to generate 256 bit tag (in contrast with 128 bit tags in PHOTON-Beetle-AEAD[32]). Hence, in a combined PHOTON-Beetle-AEAD[32], PHOTON-Beetle-Hash[32] implementation, the implementation of PHOTON-Beetle-Hash[32] comes at a free of cost.

2. PHOTON-Beetle-AEAD[128] + PHOTON-Beetle-Hash[32]. In this version, the state size, mode and the underlying permutation remain same. However, the rate of absorption is different for the AEAD and the hash. From the functional point of view, the main design components remain same.

2 SBox Analysis of PHOTON Beetle

2.1 Difference Distribution Table(DDT)

DDT(Δ_0, Δ_1)=$\#\{x \in \{0,1\}^n | (S(x) \oplus S(x \oplus \Delta_0) = \Delta_1\}$. (See figure 10.)

$\Delta_0 \setminus \Delta_1$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
a	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
b	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
c	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
d	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
e	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
f	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

2.1.1 1-1 bit DDT

(Verified using DDT1.c)

1-1 bit DDT as a sub-table of the DDT conitaining Hamming weight 1 difference.

Table: 1-1 bit DDT

$\Delta_x \setminus \Delta_y$	0001	0010	0100	1000
bit 0 = 0001	0	0	0	0
bit 1 = 0010	0	0	0	0
bit 2 = 0100	0	0	0	0
bit 3 = 1000	0	0	0	0

Here 1-1 bit DDT have all values are zero beacuse this 1-1 DDT made for BN3 S-box.

GI = {bit 3, bit 2, bit 1, bit 0}, GO = {bit 3, bit 2, bit 1, bit 0},
BI = {}, BO = {}.

So, Our all inputs are Good Input and all outputs are Good Outputs.

BOGI Permutation:
Necessary and sufficient condition for BOGI permutation:
$|BO| \leq |GI| \Rightarrow |GI| + |GO| \geq 4$.

And our Sbox or 1-1 bit DDT satisfy these conditions. So our S-box is compatible with a BOGI permutation.

Note: Score of S-box $|GI| + |GO| == 8$.

2.2 Linear Approximation Table(LAT)

$\Delta_0 \setminus \nabla_0$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	-4	0	-4	0	0	0	0	0	-4	0	+4
2	0	0	+2	+2	-2	-2	0	0	+2	-2	0	+4	0	+4	-2	+2
3	0	0	+2	+2	+2	-2	-4	0	-2	+2	-4	0	0	0	-2	-2
4	0	0	-2	+2	-2	-2	0	+4	-2	-2	0	-4	0	0	-2	+2
5	0	0	-2	+2	-2	+2	0	0	+2	+2	-4	0	+4	0	+2	+2
6	0	0	0	-4	0	0	-4	0	0	-4	0	0	+4	0	0	0
7	0	0	0	+4	+4	0	0	0	0	-4	0	0	0	0	+4	0
8	0	0	+2	-2	0	0	-2	+2	-2	+2	0	0	-2	+2	+4	+4
9	0	+4	-2	-2	0	0	+2	-2	-2	-2	-4	0	-2	+2	0	0
a	0	0	+4	0	+2	+2	+2	-2	0	0	0	-4	+2	+2	-2	+2
b	0	-4	0	0	-2	-2	+2	-2	-4	0	0	0	+2	+2	+2	-2
c	0	0	0	0	-2	-2	-2	-2	+4	0	0	-4	-2	+2	+2	-2
d	0	+4	+4	0	-2	-2	+2	+2	0	0	0	0	+2	-2	+2	-2
e	0	0	+2	+2	-4	+4	-2	-2	-2	-2	0	0	-2	-2	0	0
f	0	+4	-2	+2	0	0	-2	-2	-2	+2	+4	0	+2	+2	0	0

2.2.1 1-1 bit LAT

(Verified using LAT1.c)
1-1 bit LAT as a sub-table of the DDT conitaining Hamming weight 1 difference.

Table: 1-1 bit LAT

$\Delta_x \setminus \Delta_y$	0001	0010	0100	1000
bit 0 = 0001	0	0	0	0
bit 1 = 0010	0	+2	-2	+2
bit 2 = 0100	0	-2	-2	-2
bit 3 = 1000	0	+2	0	-2

2.3 The Boomerang Connectivity Table(BCT)

Definition: Let S be an invertible function from F_2^n to F_2^n, and Δ_0, $\nabla_0 \in F_0^n$. The boomerang connectivity table(BCT) of S is defined by a $2^n \times 2^n$ table, in which the entry for (Δ_0, ∇_0) is computed by:

BCT(Δ_0, ∇_0)=$\#\{x \in \{0,1\}^n | S^{-1}(S(x) \oplus \nabla_0) \oplus S^{-1}(S(x \oplus \Delta_0) \oplus \nabla_0) = \Delta_0\}$.

The generation of the BCT can be visualized in Figure 10. The ladder switch is captured by the BCT in the case where at least one of the index equals to zero. The S-box switch is captured by the BCT in the case where Δ_0 equals ∇_1. Morever, the incompatibility pointed out by Murthy simply corresponds to zero entries in the BCT.

In \ Out	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	16	0	4	4	0	16	4	4	4	4	0	0	4	4	0	0
2	16	0	0	6	0	4	6	0	0	0	2	0	2	2	2	0
3	16	2	0	6	2	4	4	2	0	0	2	2	0	0	0	0
4	16	0	0	0	0	4	2	2	0	6	2	0	6	0	2	0
5	16	2	0	0	2	4	0	0	0	6	2	2	4	2	0	0
6	16	4	2	0	4	0	2	0	2	0	0	4	2	0	4	8
7	16	4	2	0	4	0	2	0	2	0	0	4	2	0	4	8
8	16	4	0	2	4	0	0	2	0	2	0	4	0	2	4	8
9	16	4	2	0	4	0	2	0	2	0	0	4	2	0	4	8
a	16	0	2	2	0	4	0	0	6	0	2	0	0	6	2	0
b	16	2	0	0	2	4	0	0	4	2	2	2	0	6	0	0
c	16	0	6	0	0	4	0	6	2	2	2	0	0	0	2	0
d	16	2	4	2	2	4	0	6	0	0	2	2	0	0	0	0
e	16	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
f	16	8	0	0	8	0	0	0	0	0	0	8	0	0	8	16

2.4 Boomerang Difference Table(BDT)

(Run BDT.c to generate BDT.)

Definition: Let S be an invertible function from F_2^n to F_2^n, and Δ_0, Δ_1, $\nabla_0 \in F_0^n$. The Boomerang Difference Table(BDT) of S is defined by:

BDT(Δ_0, Δ_1, ∇_0)=
$\#\{x \in \{0,1\}^n | S^{-1}(S(x) \oplus \nabla_0) \oplus S^{-1}(S(x \oplus \Delta_0) \oplus \nabla_0) = \Delta_0,\ S(x) \oplus S(x \oplus \Delta_0) = \Delta_1\}$,
n is the S-box size.

BDT is a combination of BCT and DDT.
The time complexity for the construction is $O(2^{2n})$. Verified this complexity using **BDT.c**.

Properties:
(Verified using BDT_properties.c)

- $DDT(\Delta_0, \Delta_1) = BDT(\Delta_0, \Delta_1, 0)$
- $BCT(\Delta_0, \Delta_1) = \sum_{\Delta_1=0}^{2^n} BDT(\Delta_0, \Delta_1, \nabla_0)$
- $BDT(0, 0, \nabla_0) = 2^n$
- $(\Delta_0, \Delta_1, \nabla_0)$ is incompatible when the corresponding entry in BDT is 0.

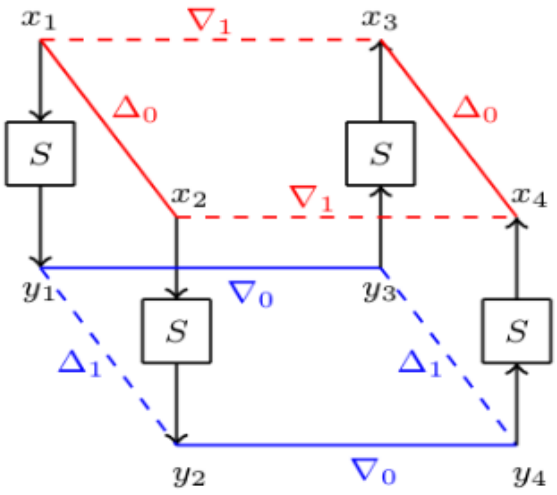

Figure 10: Generation of a right quarter at the S-box level

3 Logical Condition Model for S-box:

$(x_0, x_1, x_2, x_3)$and (y_0, y_1, y_2, y_3) represent the input and output bit-level differences of the S-box respectively, where x_3 and y_3 are the least significant bits. DDT tells impossible patterns of $(x_3x_2x_1x_0y_3y_2y_1y_0)$. Each impossible pattern can be removed one inequality.

Example: $Pr[(\Delta_i, \Delta_O) = (0x1, 0x2)] = 0$
$x_3x_2x_1x_0 = 0001$, $y_3y_2y_1y_0 = 0010$
$x_3 + x_2 + x_1 - x_0 + y_3 + y_2 - y_1 + y_0 \geq -1$

Out of 256 entries of DDT, about 159 entries are impossible. Each S-box can be modeled with about 159 inequalities.

Reducing the Number of Inequalities
Sun et al. pointed out that several impossible patterns of $(x_3x_2x_1x_0y_3y_2y_1y_0)$ can be removed simultaneously.

Example:
$Pr[(\Delta_i, \Delta_O) = (0x1, 0x2)] = 0 = Pr[(\Delta_i, \Delta_O) = (0x1, 0x6)] = 0$
$x_3x_2x_1x_0y_3y_2y_1y_0 = 00010010$
$x_3x_2x_1x_0y_3y_2y_1y_0 = 00010110$
$x_3 + x_2 + x_1 - x_0 + y_3 - y_1 + y_0 \geq -1$.

Each S-box can be modeled with less than 159 inequalities.

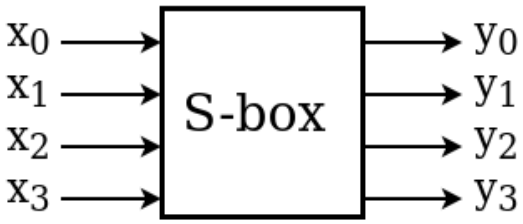

Figure 11:

Theorem 1. If we assume that all variables are 0-1 variables, then the logical condition that $(x_0, \cdots, x_{m-1}) = (\delta_0, \cdots, \delta_{m-1}) \in \{0,1\}^m \subseteq Z^m$ implies $y = \delta \in 0, 1 \subseteq Z$ can be described by the following linear inequality

$$\sum_{i=0}^{m-1}(-1)^{\delta_i}x_i + (-1)^{\delta+1}y - \delta + \sum_{i=0}^{m-1}\delta_i \geq 0. \quad (1)$$

where δ_i , δ are fixed constants and Z is the set of all integers.

Proof. Case 1. $\delta = 0$. We assume

$$(\delta_0, \cdots, \delta_{m-1}) = (\delta_0, \cdots, \delta_{s1-1};, \delta_{s1}, \cdots, \delta_{m-1}) = (1, 1, \cdots, 1;, 0, 0, \cdots, 0) = \Delta^*.$$

For other 0-1 patterns, it can be permuted into such a form and this will notaffect our proof.
First s1 δ_i are all 1s and other(or last) m-s1 δ_i are all 0s.

Verify: (Δ^*,0) is satisfy by (1).

$\delta_i = 0$. So, $(-1)^{\delta_i} = 1$ but $\delta_i = x_i$. So, $(-1)^{\delta_i}x_i = 0$.
$\delta_i = 1$. So, $(-1)^{\delta_i} = -1$ but $\delta_i = x_i$. So, $(-1)^{\delta_i}x_i = -1$.
So,
$\sum_0^{m-1}(-1)^{\delta_i}x_i = -s1$.

And $(-1)^{\delta+1}y = 0$ because $y = \delta = 0$.

And $\sum_{i=0}^{m-1}\delta_i = s1$.

So, $\sum_{i=0}^{m-1}(-1)^{\delta_i}x_i + (-1)^{\delta+1}y - \delta + \sum_{i=0}^{m-1}\delta_i = -s1 + 0 - 0 + s1 = 0$.

So, (Δ^*,0) satisfied by (1).

Verify: (Δ^*,1) is satisfy by (1).

$\delta_i = 0$. So, $(-1)^{\delta_i} = 1$ but $\delta_i = x_i$. So, $(-1)^{\delta_i}x_i = 0$.

$\delta_i = 1$. So, $(-1)^{\delta_i} = -1$ but $\delta_i = x_i$. So, $(-1)^{\delta_i} x_i = -1$.
So,
$\sum_0^{m-1} (-1)^{\delta_i} x_i = -s1$.

And $(-1)^{\delta+1} y = 1$ because $y = \delta = 1$.

And $\sum_{i=0}^{m-1} \delta_i = s1$.

So, $\sum_{i=0}^{m-1} (-1)^{\delta_i} x_i + (-1)^{\delta+1} y - \delta + \sum_{i=0}^{m-1} \delta_i = -s1 + 1 - 1 + s1 = 0$.

So, (Δ^*,0) satisfied by (1).

Fact 1. As we know the S-box of PRESENT-80 and the S-box Photon beetle is similar and the sbox has the following properties:
(i) $1001 \rightarrow ***0$: If the input difference of the S-box is $0x9 = 1001$, then the least significant bit of the output difference must be 0;
(ii) $0001 \rightarrow ***1$ and $1000 \rightarrow ***1$: If the input difference of the S-box is $0x1 = 0001$ or $0x8 = 1000$, then the least significant bit of the output difference mustbe 1;
(iii) $***1 \rightarrow 0001$ and $***1 \rightarrow 0100$: If the output difference of the S-box is $0x1 = 0001$ or $0x4 = 0100$, then the least significant bit of the input difference must be 1;and
(iv) $***0 \rightarrow 0101$: If the output difference of the S-box is $0x5 = 0101$, then theleast significant bit of the input difference must be 0.

Fact 2. Let 0-1 variables (x_0, x_1, x_2, x_3) and (y_0, y_1, y_2, y_3) represent the input and output bit-level differences of the S-box respectively, where x_3 and y_3 are the least significant bits. Then the logical conditions in Theorem 1 can be describedby the following linear inequalities:

$$-x_0 + x_1 + x_2 - x_3 - y_3 + 2 \geq 0 \tag{2}$$

$$\begin{aligned} &x_0 + x_1 + x_2 - x_3 + y_3 \geq 0 \\ &-x_0 + x_1 + x_2 - x_3 + y_3 \geq 0 \end{aligned} \tag{3}$$

$$\begin{aligned} &x_3 + y_0 + y_1 + y_2 - y_3 \geq 0 \\ &x_3 + y_0 - y_1 + y_2 + y_3 \geq 0 \end{aligned} \tag{4}$$

$$-x_3 + y_0 - y_1 + y_2 - y_3 + 2 \geq 0 \tag{5}$$

For example, the linear inequality (2) removes all differential patterns ofthe form $(x_0, \cdots, x_3, y_0, \cdots, y_3)$ $= (1, 0, 0, 1, *, *, *, 1)$, where $(x_0, \cdots, x_3)$ and $(y_0, \cdots, y_3)$ are the input and output differences of the S-box respectively. We call this group of constraints presented in (2), (3), (4), and (5) the constraints of conditional differential propagation (CDP constraints for short).

4 Zero-Sum Distinguisher for PHOTON Permutation f

(Verified using Zero_Sum.c)

4.1 Derivates of Boolean Functions

For a boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, the first derivative w.r.t i is defined as

$\frac{\delta f(x_1,x_2,\cdots,x_i,\cdots,x_n)}{\delta x_i} = f(x_1,x_2,\cdots,0,\cdots,x_n) \oplus f(x_1,x_2,\cdots,1,\cdots,x_n)$

Hence, the k th order differential w.r.t $x_{j_1}, x_{j_2}, \cdots, x_{j_k}$ can be subsequently defined as

$\frac{\delta^k f(x)}{\delta x_{j_1} \delta x_{j_2} \cdots, \delta x_{j_k}} = \bigoplus_{x_{j_1}, x_{j_2}, \cdots, x_{j_k}=0,1} f(x)$

$x = \{x_1, x_2, \cdots, x_n\}$

4.2 Zero Sum Distinguisher Using HODs

The Zero Sum Distinguisher for n-Round Distinguisher can be constructed as follows(using Boolean functions) -
We have to vary 2^{n+1} bits of an arbitrary state to generate 2^{2^n+1} plaintexts. So from the inputs or plaintext construction, XOR of all plaintexts in 0. Then we compute n Round PHOTON permutation to get outputs. Then by the properties of Zero Sum Distinguisher, the XOR of the results should also be equal to 0. Although very easy to theorize, such a distinguisher is infeasible to construct for almost all values of $n > 4$ on a reasonably powered machine.

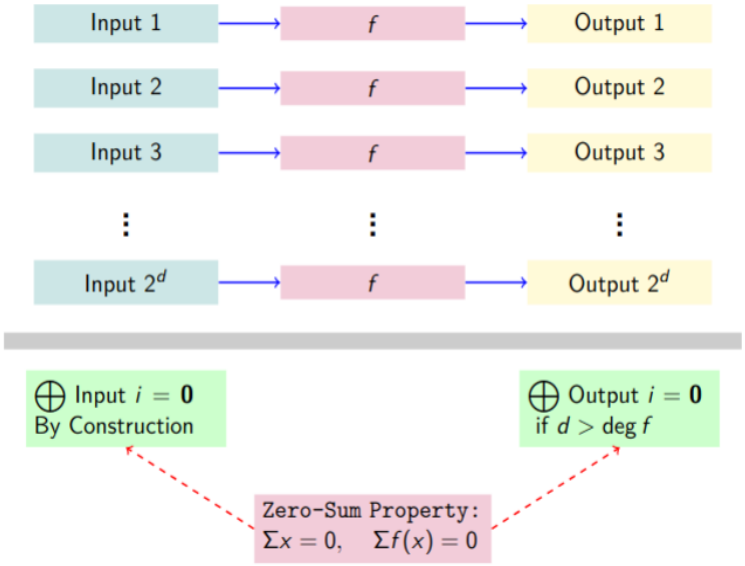

Figure 12: Zero-Sum Distinguisher

5 Refrences

https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/sbox.html
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9264171
https://www.isical.ac.in/ lightweight/beetle/
https://asecuritysite.com/encryption/beetle