Analysis of Skinny Cipher

BitBees

Indian Institute of Technology Bhilai

Abstract. We provide a report on the SKINNY cipher in this document. We present SKINNY, a highly hardware and software performant block cipher family that also offers significantly stronger security guarantees against differential and linear attacks. For resilience against side-channel attacks, SKINNY provides a range of key, tweak, and block sizes. It also benefits from exceptionally effective threshold implementations. In order to encrypt, SKINNY has the lowest overall percentage of logic gates used in software and a high implementation efficiency for microcontrollers. We described how this SKINNY encryption was built step-by-step. With the use of flowcharts and step-by-step diagrams, we elegantly explain how SKINNY is put together.

Keywords: Block Cipher · SKINNY · Lightweight · Tweakable · MILP · Cryptanalysis

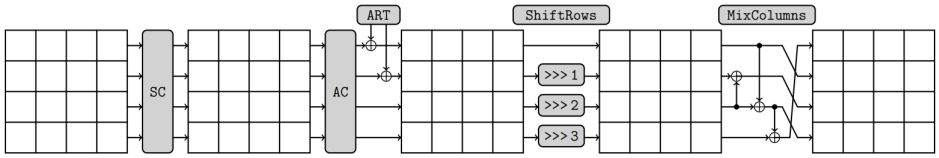
1 Construction of the Cipher

1.1 Internal State

We represent our internal state in a 4×4 matrix as shown below. We also represent our Tweakey in a 4×4 matrix. Please note that we consider row-wise indexing for our states. For internal state we use the notation IS and for the Tweakey we use the notation TK

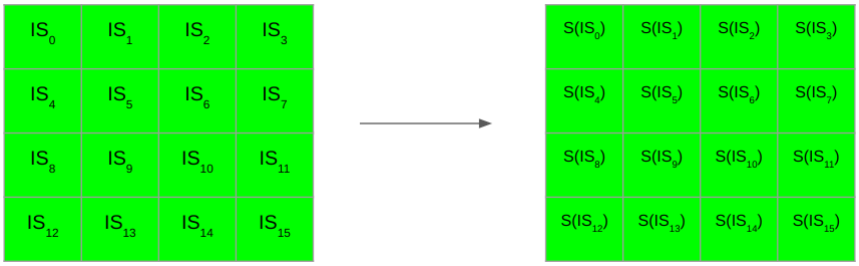
1.1.1 Overview

We have shown the entire construction summarised in the figure below. We will break it down in the following subsections.



1.2 SubCells

In this operation, we substitute each cell of our IS with the value in the S-Box.



The 4-bit S-Box is given below.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}_4[x]$	c	6	9	0	1	a	2	b	3	8	5	d	4	e	7	f
$\mathcal{S}_4^{-1}[x]$	3	4	6	8	c	a	1	e	9	2	5	7	0	b	d	f

The 8-bit S-box is given below.

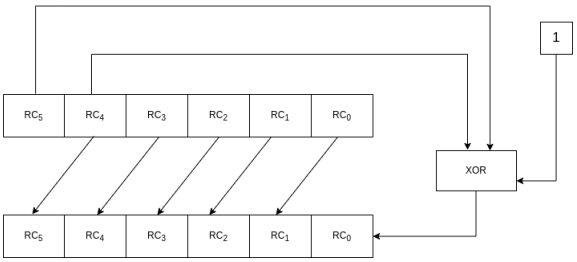
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	65	4c	6a	42	4b	63	43	6b	55	75	5a	7a	53	73	5b	7b
1	35	8c	3a	81	89	33	80	3b	95	25	98	2a	90	23	99	2b
2	e5	cc	e8	c1	c9	e0	c0	e9	d5	f5	d8	f8	d0	f0	d9	f9
3	a5	1c	a8	12	1b	a0	13	a9	05	b5	0a	b8	03	b0	0b	b9
4	32	88	3c	85	8d	34	84	3d	91	22	9c	2c	94	24	9d	2d
5	62	4a	6c	45	4d	64	44	6d	52	72	5c	7c	54	74	5d	7d
6	a1	1a	ac	15	1d	a4	14	ad	02	b1	0c	bc	04	b4	0d	bd
7	e1	c8	ec	c5	cd	e4	c4	ed	d1	f1	dc	fc	d4	f4	dd	fd
8	36	8e	38	82	8b	30	83	39	96	26	9a	28	93	20	9b	29
9	66	4e	68	41	49	60	40	69	56	76	58	78	50	70	59	79
a	a6	1e	aa	11	19	a3	10	ab	06	b6	08	ba	00	b3	09	bb
b	e6	ce	ea	c2	cb	e3	c3	eb	d6	f6	da	fa	d3	f3	db	fb
c	31	8a	3e	86	8f	37	87	3f	92	21	9e	2e	97	27	9f	2f
d	61	48	6e	46	4f	67	47	6f	51	71	5e	7e	57	77	5f	7f
e	a2	18	ae	16	1f	a7	17	af	01	b2	0e	be	07	b7	0f	bf
f	e2	ca	ee	c6	cf	e7	c7	ef	d2	f2	de	fe	d7	f7	df	f

1.3 AddConstants

A 6-bit affine LFSR, whose state is denoted $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ (with rc_0 being the least significant bit), is used to generate round constants. Its update function is defined as:

$$(rc_5 \parallel rc_4 \parallel rc_3 \parallel rc_2 \parallel rc_1 \parallel rc_0) \rightarrow (rc_4 \parallel rc_3 \parallel rc_2 \parallel rc_1 \parallel rc_0 \parallel rc_5 \oplus rc_4 \oplus 1)$$

The figure for the LFSR is given as follows:



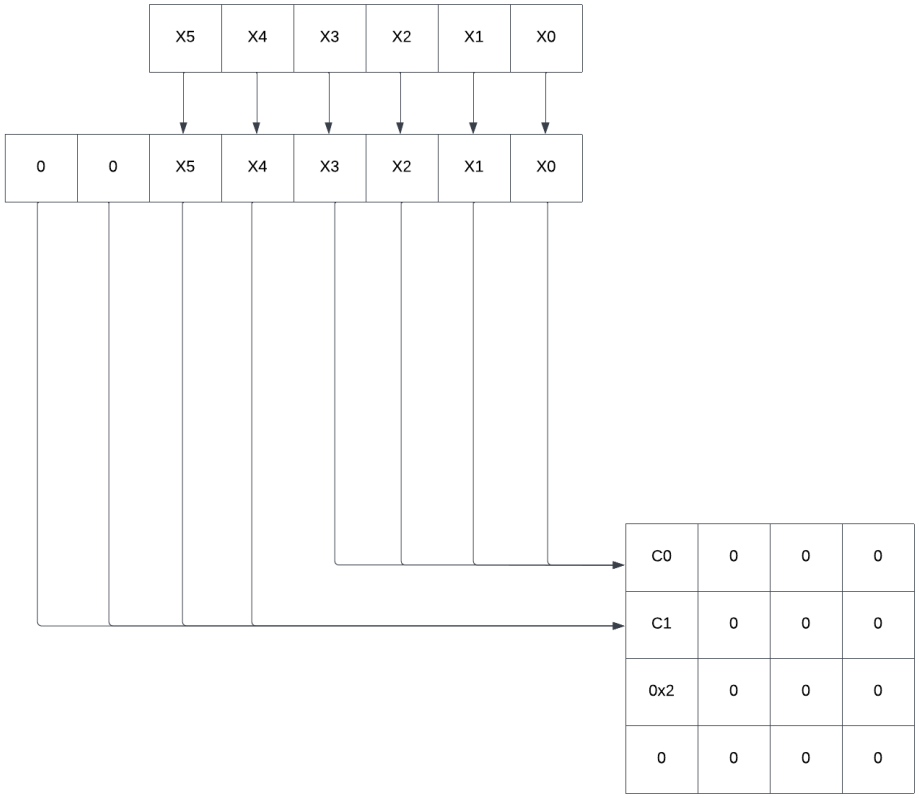
The six bits are initialized to zero, and updated before use in a given round.
We then use the values of the current LFSR state to form our constant matrix.

$$\begin{bmatrix} c_0 & 0 & 0 & 0 \\ c_1 & 0 & 0 & 0 \\ c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

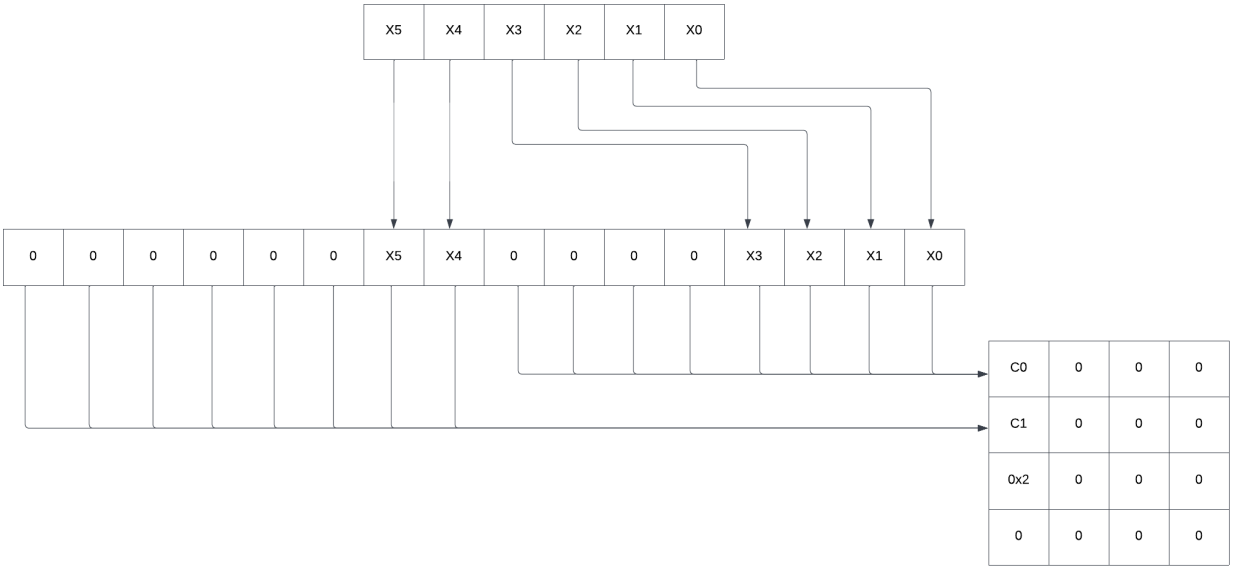
with $c_2 = \texttt{0x2}$ and

$$(c_0, c_1) = (\mathrm{rc}_3 \| \mathrm{rc}_2 \| \mathrm{rc}_1 \| \mathrm{rc}_0,\ 0 \| 0 \| \mathrm{rc}_5 \| \mathrm{rc}_4) \text{ when } s = 4$$
$$(c_0, c_1) = (0 \| 0 \| 0 \| 0 \| \mathrm{rc}_3 \| \mathrm{rc}_2 \| \mathrm{rc}_1 \| \mathrm{rc}_0,\ 0 \| 0 \| 0 \| 0 \| 0 \| 0 \| \mathrm{rc}_5 \| \mathrm{rc}_4) \text{ when } s = 8.$$

This is the entire operation for updating round constants when $s = 4$



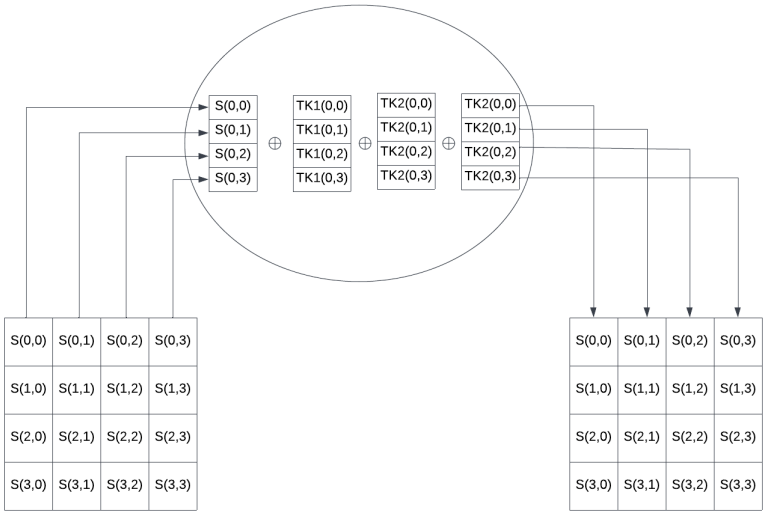
This is the entire operation for updating round constants when $s = 8$



The round constants are combined with the state, respecting array positioning, using bitwise exclusive-or.

1.4 AddRoundTweakey

The first and second rows of all Tweakey arrays are extracted and bitwise exclusive-ored to the cipher internal state, respecting the array positioning.



Formally, for $i = \{0, 1\}$ and $j = \{0, 1, 2, 3\}$, we have:

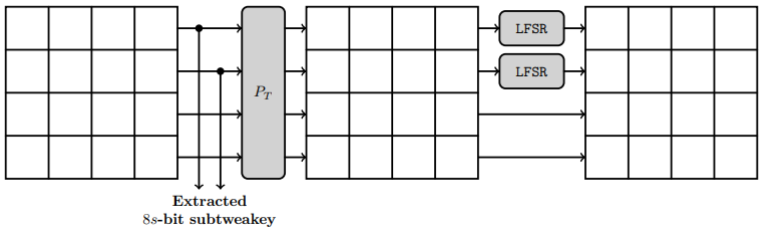
1. $IS_{i,j} = IS_{i,j} \oplus TK1_{i,j}$ when $z = 1$

2. $IS_{i,j} = IS_{i,j} \oplus TK1_{i,j} \oplus TK2_{i,j}$ when $z = 2$

3. $IS_{i,j} = IS_{i,j} \oplus TK1_{i,j} \oplus TK2_{i,j} \oplus TK3_{i,j}$ when $z = 3$

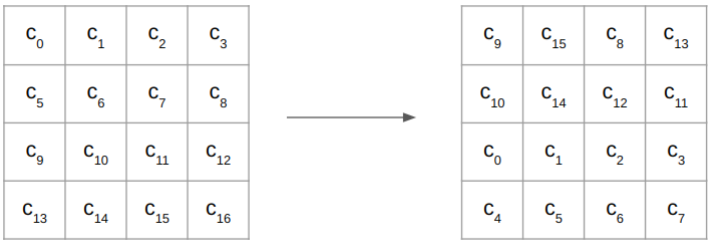
1.4.1 Tweakey Schedule

The tweakey is updated every round, For updating the tweakey we will pass the tweakey through a permutation block and after that a LFSR will be applied on `TK2` and `TK3` (ifthey are present), the flow diagram of the schedule is shown below.



Before we proceed, we need to know how the Tweakey is updated before going on to the next round. First, a permutation P_T is applied on the cells positions of all Tweakey arrays: for all $0 \leq i \leq 15$, we set $TK1_i \leftarrow TK1_{P_T[i]}$ with

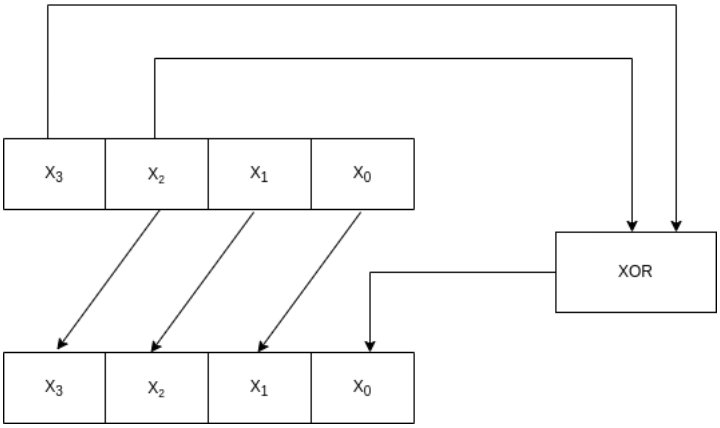
$$P_T = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7]$$



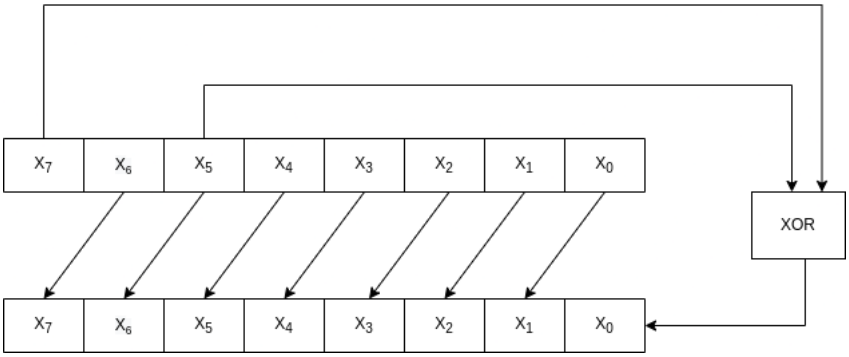
Then we apply LFSR to `TK2` and `TK3` (if they exist). The expressions for LFSR are given below.

TK	s	**LFSR**
$TK2$	4	$(x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_2\|\|x_1\|\|x_0\|\|x_3 \oplus x_2)$
	8	$(x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0\|\|x_7 \oplus x_5)$
$TK3$	4	$(x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_0 \oplus x_3\|\|x_3\|\|x_2\|\|x_1)$
	8	$(x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_0 \oplus x_6\|\|x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1)$

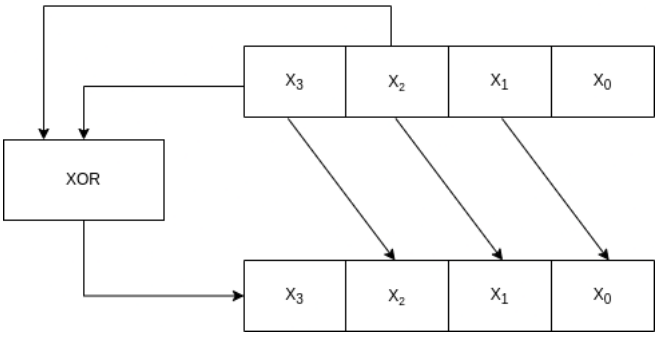
LFSR for TK2 when $s = 4$.



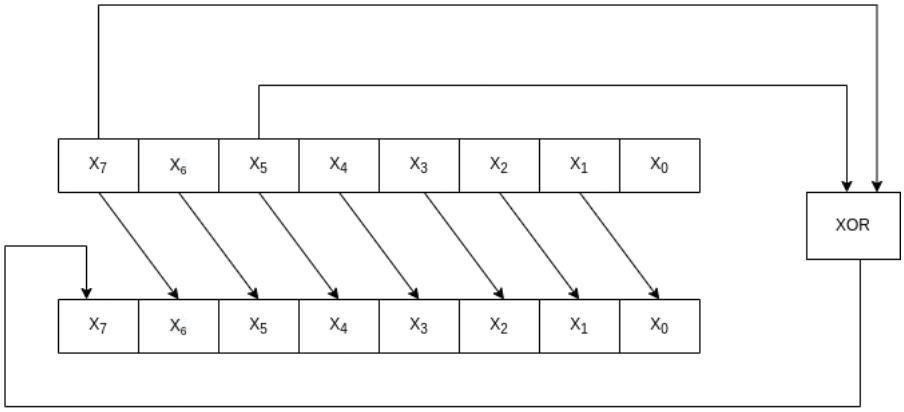
LFSR for TK2 when $s = 8$.



LFSR for TK3 when $s = 4$.

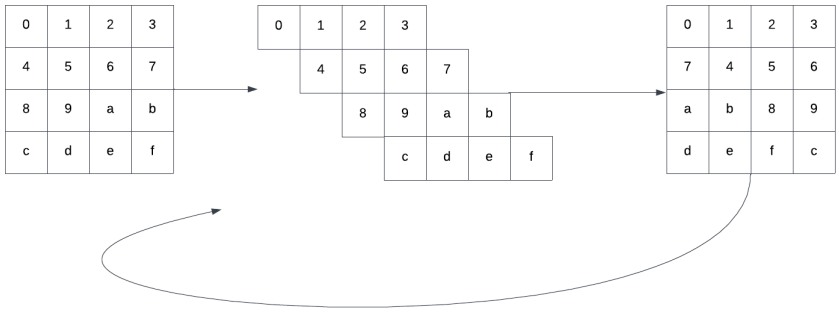


LFSR for TK3 when $s = 8$.



1.5 ShiftRows

As in AES, in this layer the rows of the cipher state cell array are rotated, but they are to the right. More precisely, the second, third, and fourth cell rows are rotated by 1, 2 and 3 positions to the right, respectively.



1.6 MixColumns

Each column of the cipher internal state array is multiplied by the following binary matrix M

$$M = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} IS_{0,0} & IS_{0,1} & IS_{0,2} & IS_{0,3} \\ IS_{1,0} & IS_{1,1} & IS_{1,2} & IS_{1,3} \\ IS_{2,0} & IS_{2,1} & IS_{2,2} & IS_{2,3} \\ IS_{3,0} & IS_{3,1} & IS_{3,2} & IS_{3,3} \end{bmatrix}$$

2 Analyses

2.1 S-box Analysis

2.1.1 S-box Construction/ ANF

Table 1. 4-bit Sbox $\mathcal{S}_4$ used in SKINNY when $s = 4$.

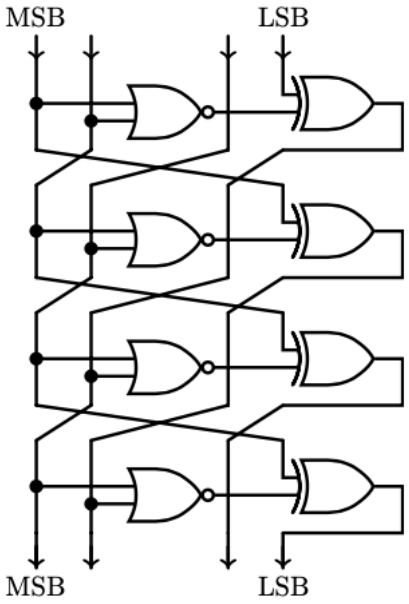
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}_4[x]$	c	6	9	0	1	a	2	b	3	8	5	d	4	e	7	f
$\mathcal{S}_4^{-1}[x]$	3	4	6	8	c	a	1	e	9	2	5	7	0	b	d	f

(1)

The 4-bit S-box uses 4 XOR and 4 NOR operations. The below transformation is applied, with x_0 being the least significant bit, and followed by a left shift bit rotation. This process is repeated four times, except for the last iteration where the bit rotation is omitted.

$$(x_3, x_2, x_1, x_0) \rightarrow (x_3, x_2, x_1, x_0 \oplus (\overline{x_3 \vee x_2}))$$

The following are the diagrams for the 4-bit S-box construction:



The 8-bit S-box follows the similar pattern as the 4-bit S-box with the following transformation:

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \rightarrow (x_7, x_6, x_5, x_4 \oplus (\overline{x_7 \vee x_6}), x_3, x_2, x_1, x_0 \oplus (\overline{x_3 \vee x_2}))$$

and followed by the bit permutation:

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \longrightarrow (x_2, x_1, x_7, x_6, x_4, x_0, x_3, x_5)$$

2.1.2 DDT analysis

[O'C94][SS18] The difference distribution table of an S-box counts how many pairs with a given input difference lead to a given output difference. It is an essential part of identifying

high-probability transitions for constructing the differential characteristics, later used in the attack. The following is the DDT(16x16) for 4-bit S-box.

in/out	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	0	0	0	0	0	4	4	4	4	0	0	0	0
2	0	4	0	4	4	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2
4	0	0	4	0	0	0	2	2	0	0	0	4	2	2	0	0
5	0	0	4	0	0	0	2	2	4	4	0	0	0	0	2	2
6	0	0	4	0	0	0	2	2	0	0	4	0	2	2	0	0
7	0	2	0	2	2	0	0	2	2	2	0	0	0	2	2	0
8	0	2	0	2	2	0	0	2	0	0	2	2	0	0	0	2
9	0	4	0	2	0	0	2	0	0	0	2	0	4	0	0	2
A	0	4	0	2	0	0	0	2	0	0	2	0	0	4	2	0
B	0	4	0	2	4	0	0	2	0	0	2	0	0	0	2	0
C	0	0	0	2	2	2	0	2	2	2	0	2	0	0	0	2
D	0	0	0	2	2	2	2	0	2	2	0	2	0	0	2	0
E	0	2	0	2	0	0	2	0	2	2	0	2	2	2	2	0
F	0	2	0	2	0	0	0	2	2	2	0	2	2	2	0	2

(2)

2.1.3 LAT analysis

Linear cryptanalysis is an attack that derives a linear approximation between bits of the plaintext, ciphertext, and key. This global approximation is constructed from the linear approximation tables of the nonlinear mappings used by the cipher, usually the S-boxes. For any pair of masks a, b, the linear approximation table (LAT) contains the bias of $S_b + \varphi_a$, where φ_a maps x to ax.

in/out	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	−2	2	0	0	−2	−2	−4	4	−2	−2	0	0	2	−2	0
2	0	4	0	0	0	−4	0	0	0	−4	0	0	0	−4	0	0
3	0	−2	2	4	0	−2	−2	0	0	2	2	0	4	−2	2	0
4	0	0	4	0	0	0	0	4	0	0	−4	0	0	0	0	4
5	0	−2	−2	0	0	−2	−2	0	−4	−2	2	0	0	2	−2	4
6	0	0	−4	4	0	0	0	0	0	0	−4	−4	0	0	0	0
7	0	2	−2	0	0	2	−2	0	0	−2	−2	4	4	2	2	0
8	0	2	0	−2	4	−2	0	−2	0	2	0	−2	0	2	4	2
9	0	0	2	−2	0	0	2	−2	−4	0	−2	−2	4	0	−2	−2
A	0	−2	0	−2	−4	2	0	−2	0	−2	0	−2	0	−2	4	2
B	0	0	2	2	0	0	2	2	0	−4	2	−2	0	4	2	−2
C	0	2	0	2	−4	−2	4	−2	0	2	0	2	0	2	0	2
D	0	0	2	2	0	0	−2	−2	−4	0	−2	2	0	0	2	−2
E	0	2	0	−2	−4	−2	−4	2	0	2	0	−2	0	2	0	−2
F	0	4	2	2	0	4	−2	−2	0	0	2	−2	0	0	−2	2

(3)

2.1.4 Differential-uniformity

The differential uniformity of an sbox gives us the measure of how balanced the DDT of the sbox is when it is being exploited against differential attacks. It is calculated as the

maximum value in the sbox's DDT. For 4-bit S-box it is 4.

2.1.5 Differential branch number

The differential branch number measures the diffusion power of a permutation. It is interesting to note that the differential branch number is related to the difference distribution table (DDT). DDT of a permutation ϕ of $\mathbb{F}_2^n$ denoted by $\mathcal{D}_\phi$ is a matrix of order $2^n \times 2^n$. Suppose for the input difference δ, the output difference of the permutation ϕ is Δ, i.e., $\phi(x) \oplus \phi(x \oplus \delta) = \Delta$. Let $\mathcal{D}_\phi(\delta, \Delta)$ be the number solutions of $\phi(x) \oplus \phi(x \oplus \delta) = \Delta$, then the (δ, Δ)-th element of DDT is $\mathcal{D}_\phi(\delta, \Delta)$. For 4-bit S-box it is 2.

2.2 The Playing Field

Here is a table comparing these parameters with all other sboxes of all other ciphers given in class.

Group Name	**Cipher Name**	**S-Box Size**	**DU**	**D.B.R**
gugu gaga	Midori	4-bit	4	2
SPL Encrypted	GIFT	4-bit	6	2
Hope we "3" SDVV	Serpent	4-bit	4	3
-.././cipher	Prince	4-bit	4	2
TechHeist 3.0	Pride	4-bit	4	2
Rook	Ascon	5-bit	8	3
Three Amigos	Klein	4-bit	4	2
Decryptor	PHOTON-beetle	4-bit	4	3
cryptoducks	LED	4-bit	4	3
Ping 999+	Elephant	4-bit	4	3
Kryptonian	Wage	8-bit	8	2
cipherbytes	Aria	8-bit	4	2
C14	Primates APE	5-bit	2	2
BitBees	**Skinny**	**4-bit**	**4**	**2**
Bash Ciphers	PRINT	3-bit	2	2
SHA69	Mysterion	4-bit	4	2
Hex Brains	Rectangle	4-bit	4	2
Brain fog	Pyjamask-128	4-bit	4	2

3 Cryptanalysis

3.1 Differential Analysis

To demonstrate SKINNY's resistance to differential and linear attacks, we computed lower bounds on the minimal number of active Sboxes in both the single-key and related-Tweakey models. In a differential (resp. linear) characteristic, a Sbox is said to be active if it has a non-zero input difference (resp. input mask). In contrast to the single-key model, where the round Tweakeys are constant and thus have no effect on the activity pattern, an attacker in the related-Tweakey model is allowed to introduce differences (resp. masks) within the Tweakey state. We looked at three scenarios: choosing input differences in `TK1` only, both `TK1` and `TK2`, and all Tweakey states `TK1`, `TK2`, and `TK3`, respectively. The table below presents lower bounds on the number of differential active Sboxes for 1 up to 30 rounds.

Model	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SK	1	2	5	8	12	16	26	36	41	46	51	55	58	61	66
TK1	0	0	1	2	3	6	10	13	16	23	32	38	41	45	49
TK2	0	0	0	0	1	2	3	6	9	12	16	21	25	31	35
TK3	0	0	0	0	0	0	1	2	3	6	10	13	16	19	24
SK Lin	1	2	5	8	13	19	25	32	38	43	48	52	55	58	64

Model	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
SK	75	82	88	92	96	102	108	(114)	(116)	(124)	(132)	(138)	(136)	(148)	(158)
TK1	54	59	62	66	70	75	79	83	85	88	95	102	(108)	(112)	(120)
TK2	40	43	47	52	57	59	64	67	72	75	82	85	88	92	96
TK3	27	31	35	43	45	48	51	55	58	60	65	72	77	81	85
SK Lin	70	76	80	85	90	96	102	107	(110)	(118)	(122)	(128)	(136)	(141)	(143)

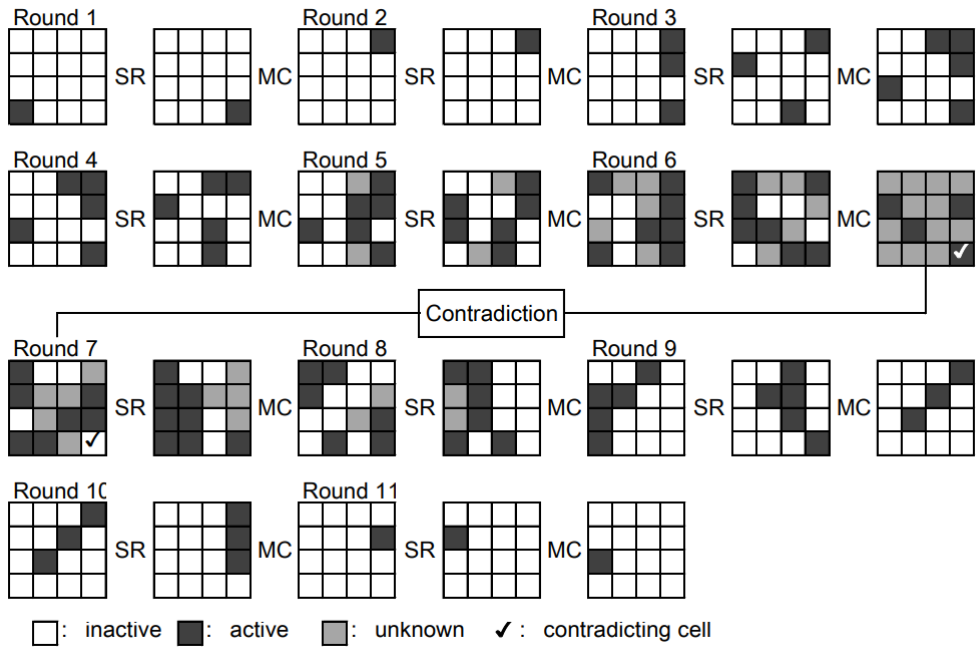
3.2 Impossible Differential Attack

Impossible differential attack finds two internal state differences Δ, Δ' such that Δ is never propagated to Δ'. The attacker then finds many pairs of plaintext/ciphertext and Tweakey values leading to (Δ, Δ'). Those Tweakey values are wrong values, thus Tweakey space can be reduced.

The impossible differential characteristic is found using the miss in the middle technique. In short, 16 input truncated differentials and 16 output truncated differentials with just a single active cell is propagated with encryption and decryption function until no cell can be active or inactive with probability 1. Then we pick the pair with contradicting each other in the middle.

The longest differential characteristics found by the designers was 11 and there were 16 such characteristics. An example of such characteristics is given below.

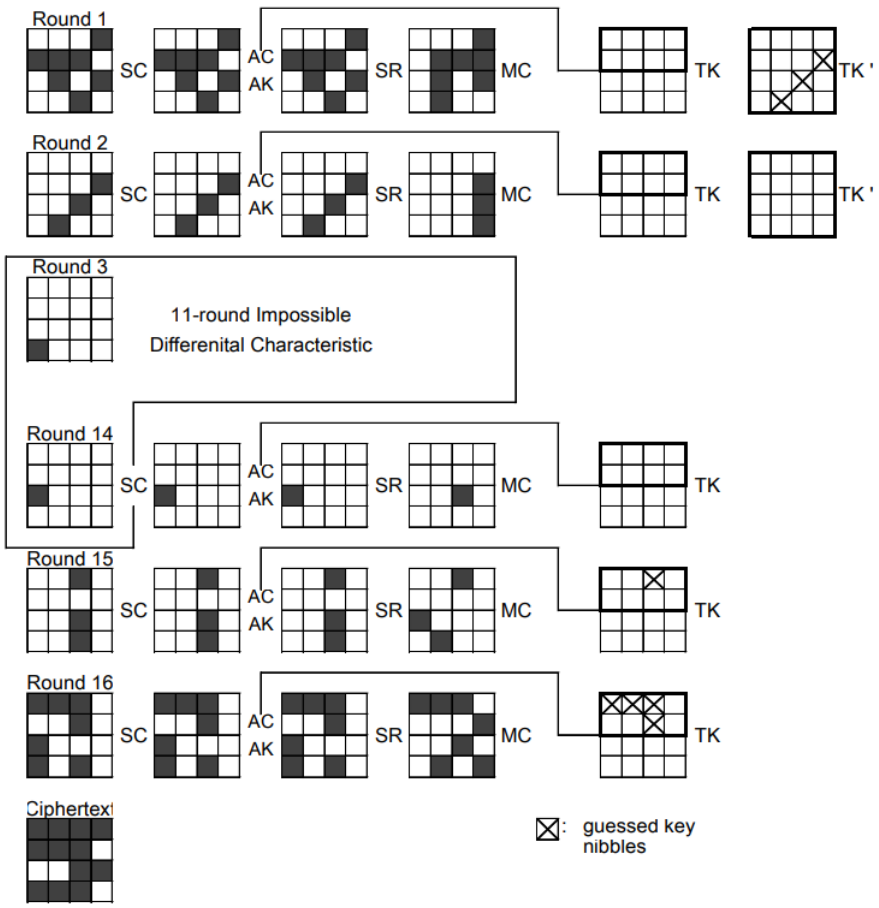
$$(0,0,0,0,0,0,0,0,0,0,0,0,\Delta,0,0,0) \longrightarrow (0,0,0,0,0,0,0,0,0,\Delta',0,0,0,0,0,0,0).$$



The above figure denotes a 11-round impossible differential characteristic. SR and MC stand for ShiftRows and MixColumns, respectively. SubCells, AddConstants and AddTweakey are omitted since they are not related to the impossible differential characteristic.

We use this 11-round characteristic as a starting point and we can append several rounds before or after this 11-round characteristic. The number of rounds appended is dependent on the key and the block size. For example, when the block size is equal to key size, we can append two round before and three rounds after the characteristics to make a 16-round key recovery.

We also make use of a special property of the `AddRoundTweakey(ARK)` operation. For the first two rounds, ARK can be moved after the `MixColumns` and `ShiftRows` operation after applying a linear transformation to the Tweakey. Then we can start our analysis by regarding the difference to Round 2 as a plaintext difference and this is masked using the Tweakey for the first round. For this characteristic, we use 3 Tweakey cells in the first round and 5 Tweakey cells in the last round, giving us a total of 8 Tweakey cells in total. The entire 16-round differential characteristics is given below.



3.2.1 Brief Walkthrough of the Attack

1. We start with the input state for Round 2. We work our way back to Round 1 initial state to get our plaintext. We make a guess on the Tweakey `TK'` for the `AddKey` suboperation for Round 1 in order to work our way backwards in Round 1.

2. Now that we have the plaintext, we then query the encryption oracle to get the ciphertext.

3. We repeat and find ciphertext for all the plaintexts(2^{x+3c} values). The attacker chooses the pairs which has 9 inactive cells after the last `MixColumns` operation.

4. We then work our way to the input of the Round 3 where our 11-Round impossible differential attack begins. We start our differential trail from Round 3 but before that we need to follow the next step.

5. Now we take our ciphertext pairs and work our way back till we reach Round 14. In Round 15, and Round 16, we make guesses for some Tweakey cells as shown in the figure above. After reaching Round 14, we take the difference in the pair of internal state and finally begin making our differential trail. We work our way from top and bottom in the differential trail Round 9 where we if we have a

contradiction(inactive/active cell), then we can say that the 8-cell Tweakey guess we made in the entire procedure is wrong.

6. We repeat the procedure from Step 4 for all the pairs.

3.2.2 Calculations and Complexity Analysis

The attacker first generates 2^x structures at the input to Round 2 and each structure contains 2^{3c} values where c is the cell size, i.e. 4 bits for `SKINNY-64` and 8 bits for `SKINNY-128`. From this we can construct a total of 2^{x+6c-1} pairs from 2^{x+3c} values. All the 2^{x+3c} values are inverted by the one keyless round to obtain the corresponding plaintext and then querying the encryption oracle to get the corresponding ciphertext. The attacker only picks the pair which has 9 inactive cells after inverting the last MC operation.

2^{x-3c-1} are the number of expected pairs to remain after filtering. For each such pair, the attacker can generate all Tweakey values for 8 cells leading to the impossible differential characteristic by guessing 5 internal state cells, which are 1-cell differences after MC in Round 3 and 4-cell differences before MC in Round 14 and Round 15. In the end, the attacker obtains $2^{x-3c-1+5c} = 2^{x+2c+1}$ wrong key suggestions for 8 Tweakey cells, which makes the remaining Tweakey space.

$$2^{8c} \cdot (1 - 2^{-8c})^{2^{x+2c-1}} = 2^{8c} \cdot e^{-2^{x-6c-1}}$$

When c = 8, we choose x = 54.5, which makes the remaining key space $2^6 4 \cdot 2^{-63.5} < 1$
When c = 4, we choose x = 29.5, which makes the remaining key space $2^6 4 \cdot 2^{-29.5} < 1$

In total, the data complexity amounts to 2^{x+3c} plaintexts chosen, and time and memory are $\max\{2^{x+3c}, 2^{x+2c-1}\}$.

For c = 8, we had chosen x = 54.5 which means complexity for `SKINNY-128` is $2^{88.5}$
For c = 4, we had chosen x = 29.5 which means complexity for `SKINNY-64` is $2^{41.5}$

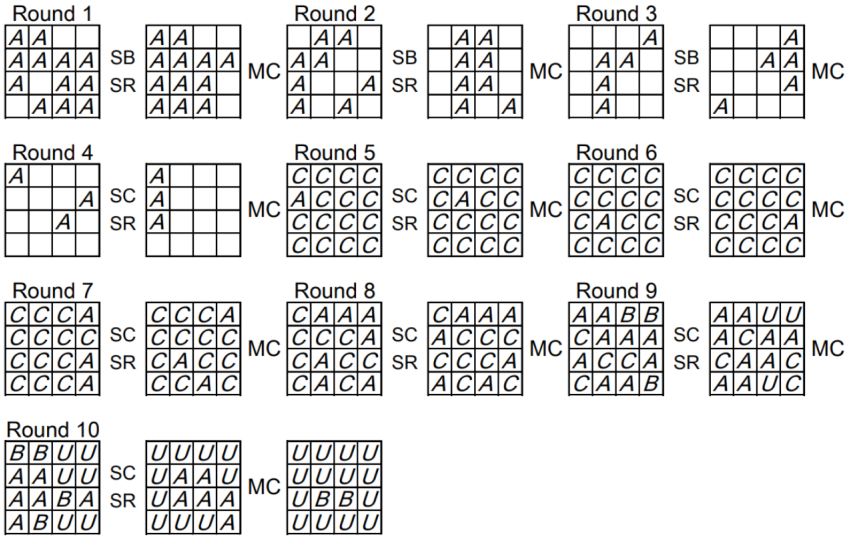
3.3 Integral Cryptanalysis

With the integral approach, we combine several plain-texts so that a select few cells can store all the values in the set while the others are set to a constant value. The properties of the multiset of internal state values are then examined after a few rounds of encryption. The four characteristics listed below are ones we pay particular attention to.

- All property(A): The cell's values all appear to be the same number.
- Balanced property(B): In the multiset, the sum of all values is 0.
- Constant property(C): Throughout the multiset the cell value is fixed.
- Unknown property(U): There doesn't exist any particular property.

To determine how many rounds can be played while still keeping any non-trivial characteristic, we employ an experimental method. The state of one active cell is established, and the remaining cells are processed using an encryption method until all of them have an

unknown status. This is tweaked and repeated multiple times with various constant-cell values. As a result, we found that an active cell in either of the third rows will produce two cells after seven cycles that satisfy the All condition. The property is then expanded to higher order by propagating in the backward-direction, the active cell.



3.3.1 Key Recovery

To develop a 14-round key-recovery attack, in the integral distinguisher, we have to append 4 rounds after the 10 rounds. The 4-round backward calculation is shown in diagram: 15, and the technique is as follows:

- To create the integral distinguisher, the attacker produces 2^{12c} plaintexts. For each ciphertext, the attacker performs an inverse MixColumns operation, then takes the parity of these 4 cell values required to go further with the backward-computation. This decreases the size of the remaining-text to 2^{8c}.
- The attacker calculates inverse SubCells and inverse MixColumns after guessing 4 cells of the latest Tweakey. After inverse Mix-Columns operation, the attacker then obtains parity of the 5-cell values once more. Finally, this stage conducts $2^{8c} \times 2^{4c} = 2^{12c}$ calculations, yielding 2^{5c} data for each Tweakey guess of four cells. It's worth noting that the guess and compress method can be used column-by-column, reducing the complexity of this phase.
- In round 13, the attacker estimates 2-cells of the Tweakey and then computes back to 2-cells following the inverse-Mix-Columns operation, using given 2^{5c} data. The parity of 2-cell values is used in the attack. 2^{2c} data is processed for $2^{4c+2c} = 2^{6c}$ Tweakey guesses, resulting in 2^{11c} calculations and 2^{2c} data for each Tweakey guess of 6 cells.
- Given 2^{2c} data, 1-cell of the Tweakey in round-12 can be retrieved from a 4-cell prediction for Tweakey in round 14, allowing the distinguisher's output to be recalculated back to the target cell. 22c data is processed for Tweakey estimations of $2^{4c+2c} = 2^{6c}$, requiring 2^{8c} round function computations.
- If the Balanced characteristic is met, on testing the computed results. The number of 7-cell key candidates is lowered by a factor of $\frac{1}{2^c}$.

- The 6-cell key candidates may be narrowed to 1 by iterating over the analysis 5 times more and the remaining 10-cells can be estimated exhaustively.

3.3.2 Complexity Analysis

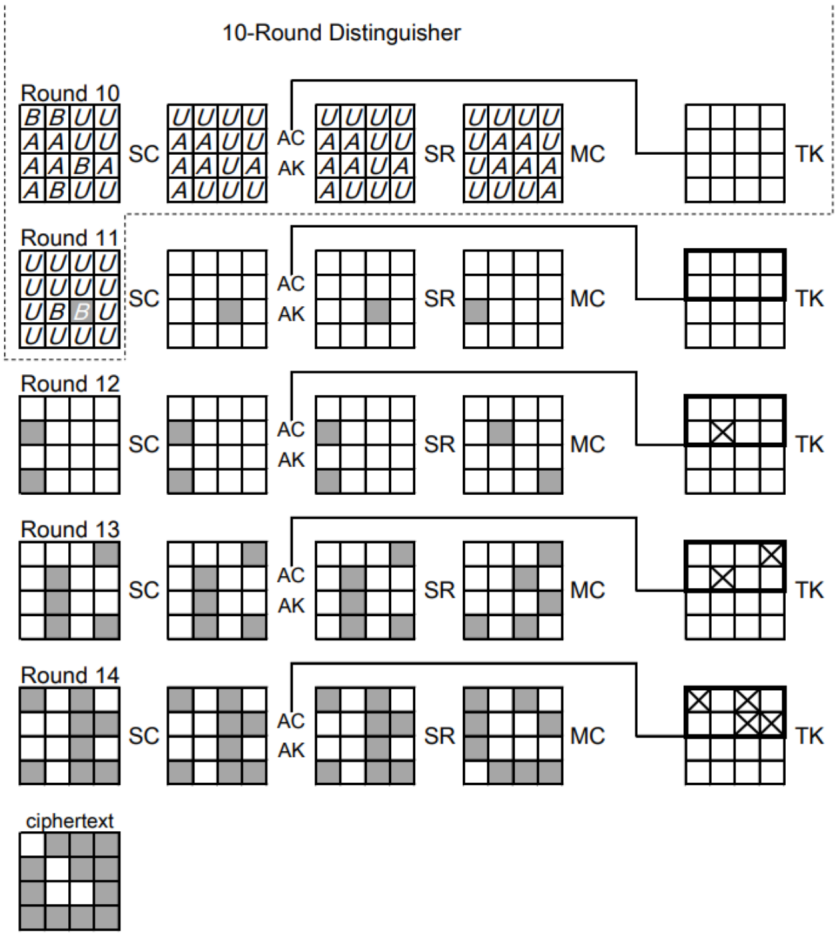
The data complexity is 2^{12c} plaintexts chosen, with memory accesses to deal with 2^{12c} ciphertext being the complexity bottleneck. The initial step, which saves 2^{8c} state values, after the first parity check, is also the memory complexity bottleneck.

3.3.3 Remarks on the Division Property

Todo et al.([Tod15]) suggested the division property as an extension of the integral property, which is particularly useful for evaluating higher-order integral properties precisely. However, when it comes to SKINNY, an experimental approach can go through more rounds. This is because of the very light round function, which allows a single active cell to have a relatively long integral property. In reality, the method by Todo shows that an evaluation algorithm against generic SPN ciphers only yields a 6-round division characteristic. To improve the integral attack by the division property, a more advanced evaluation method should be developed.

3.4 Slide Attack

In the slide attack, the search is for a pair of plaintexts where one of them also occurs in between the conversion of the first plaintext to the cipher. The *Add constant* step prevents a slide attack from occuring in the skinny cipher and the traditional slide attack can't be replicated. So, the segment of the cipher most vulnerable to a slide attack is the Affine LFSR which is used in the transformation of round constants. Differential characteristics of the cipher can be affected by the difference in the round constant which may be important and helpful for the attacker. So, after exhaustively calculating the difference between every round vulnerable to a sliding attack, mixed integer linear programming may be used to get the lower bounds of the number of active sboxes for all these constant differences. If the attacker is allowed to start from any round, then for this type of attack we found 36 active sboxes for 11 rounds and 41 sboxes for 12 rounds(these values are for lower bounds).



Slid round numbers achieving the minimal lower bounds. The notation (a,b) means that the first and second values of the pair start from round a and round b, respectively. In this table, round numbers start from 0.

36 Sboxes for 11 rounds
$(12, 31), (14, 50), (16, 30), (18, 40), (1, 25), (20, 28), (21, 38), (23, 32), (26, 47), (2, 34),$ $(33, 46), (36, 39), (3, 15), (42, 49), (44, 48), (4, 10), (6, 11), (7, 17), (8, 37), (9, 29)$
41 Sboxes for 12 rounds
$(0, 15), (10, 45), (11, 13), (12, 37), (16, 42), (17, 18), (19, 43), (21, 33), (22, 28), (24, 29),$ $(25, 35), (27, 47), (2, 44), (30, 49), (34, 48), (38, 46), (41, 50), (5, 32), (7, 40), (8, 31)$

3.5 Subspace Cryptanalysis

Affine subspaces that remain unchanged throughout the cipher round are used in invariant subspace cryptanalysis. The essential addition step, however, translates the invariant. Subspace cryptanalysis is therefore a threat to ciphers with weak keys, which prevent the translation of the affine subspace even after key addition. The key-schedule in SKINNY provides a respectable amount of resistance against these attacks for a sufficient number of rounds. However, subspaces that could spread through the sbox without changing (mainly

large-dimensional subspaces) would be an issue, but the designers have demonstrated that no such subspaces exist.

3.6 Algebraic Attacks

Probabilistically speaking, we may state that SKINNY is not in risk from an algebraic attack. The Sbox S4 and S8 have algebraic degrees of 3 and 6, respectively. According to the table below, there must be at least 26 active Sboxes in the single-key case for any consecutive 7-rounds of differential behaviour of the SKINNY.

Cipher	Model	Rounds															
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
SKINNY	**SK**	1	2	5	8	12	16	26	36	41	46	51	55	58	61	66	75
(36 rounds)	**TK2**	0	0	0	0	1	2	3	6	9	12	16	21	25	31	35	40

We have $a.26.[\frac{r}{7}] >>> n$ for all SKINNY variations, where r means number of rounds and n means the blocksize, which can be simply checked. Additionally, over GF(2), S4 is defined by e = 21, quadratic-equations in the v = 8 input or output variables. For a fixed key of SKINNY permutation, the whole system comprises $16 \times r \times e$ quadratic-equations in $16 \times r \times v$ variable. There are 10752 quadratic equations in 4096 variables in Skinny-64-64. A fixed-key AES permutation, on the other hand, has a total of 6400 equations in 2560 variables. These figures suggest that SKINNY provides a high level of security than AES against the Algebraic attacks.

3.7 Automated Cryptanalysis: MILP and Linear Attacks

Consider an n bit block cipher with input variable x belonging to the field of n bit base 2 integers. If we call v our input mask, u our output mask, and an $f-$function, then a linear approximation can be defined as follows:

$$x \longmapsto v.x \oplus u.f(x)$$

And the probability that the above linear approximation holds, is defined by:

$$p(x; u) = pr(v.x \oplus u.f(x) = 0)$$

The above equation is interesting in the view that a probability of $\frac{1}{2}$ indicates that there is no correlation between the masks and the function of the output, *i.e* there is zero correlation.
This is the goal, because in zero-correlation cryptanalysis, we look for a linear approximation for all the keys with zero correlation between them. The class of correlation attacks was inspired from the fact that differential attacks were thwarted by cipher-designers by simply adding more rounds to the cipher. This way the probability of the differential characteristic becomes worthless after a few more rounds.
Correlation attacks, on the other hand, attack the key or tweakey schedules. In order to find a zero-correlation, there are usually some XORs, branches and F-functions used in each round of any cipher. [BR14] gives three rules to understand the influence of these components on our linear cryptanalysis.

Lemma 1. ***(XOR Operation)*** *Either the three linear selection patterns at an XOR $\oplus$ are equal or the correlation over $\oplus$ is 0.*

Lemma 2. ***(Branching operation)*** *Either the three linear selection patterns at a branching point • sum up to* 0 *or the correlation over • is exactly* 0*.*

Lemma 3. ***(Permutation approximation)*** *Over a permutation ϕ, if the input and output selection patterns are neither both zero nor both nonzero, the correlation over ϕ is exactly zero. From* ***lemma 1****, we can infer that for a zero-correlation attack, inputs and outputs of an* ***XOR*** *operation should either be equal or we get a zero-correlation. And from the second lemma, the branching operation input must be the same as XOR of outputs or the correlation is zero.*

3.7.1 Zero-Correlation Attack Overview

In the first stage, the attacker should find a linear approximation with correlation zero for some rounds of the target cipher as a distinguisher. Then in the second stage, he adds some rounds before and after the distinguisher and tries to extract the subkeys of these additional rounds.

In a multidimensional case, there are m independent linear base approximations such that all the $2^m - 1$ non zero linear combinations have zero-correlation. In search of patterns, we may just have stumbled upon one. Consider the statistical value T,

$$T = \Sigma_{i=0}^{2^m-1} \frac{(V[i] - N2^{-m})^2}{N.2^{-m}(1-2^{-m})} = \frac{N.2^m}{(1-2^{-m})} \Sigma_{i=0}^{2^m-1} (\frac{V[i]}{N} - \frac{1}{2^m})^2$$

Setting aside the fact that we have a long equation on hand, it has been found that the above statistic follows a χ^2 distribution. The mean of the distribution is $\mu_0 = l(\frac{2^n-N}{2^n-1})$ and it has a variance of $\sigma_0^2 = 2l(\frac{2^n-N}{2^n-1})^2$, for the right key guess. For the wrong key guess, $\mu_1 = l$ and $\sigma_1^2 = 2l$ will be the χ^2 distribution's mean and variance respectively.
With error probability the first type as a and error probability type 2 as b, if one considers the decision threshold $t = \mu_0 + \sigma_0 z_{1-a} = \mu_1 - \sigma_1 z_{1-b}$, then the amount of required distinct known plaintexts (N) is as follows:

$$N = \frac{2^n(z_{1-a} + z_{1-b})}{\sqrt{\frac{l}{2}} - z_{1-b}}$$

This way, with knowledge of N plaintext-ciphertext pairs can be used to perform a known plaintext attack on the SKINNY family of block ciphers. An important point to notice is that the number of required pairs of plaintext-ciphertext depends on the number of linear approximations with correlation zero, block length, and error probabilities type 1 and 2.

3.7.2 MILP

In this MILP problem, the linear constraint phrase is designed to create the cryptosystem, and the objective function can be set to an expression that communicates the likelihood of the differential characteristic. As a result, we may determine the set cypher system's optimal probability of differential characteristics constructing the cipher system that corresponds to the MILP problem's solution. The differential characteristic, however, cannot be produced in the defined cypher system for that input and output differential value if we are unable to solve the MILP issue for a particular input or output differential. Therefore, the input and output differentials of the specified cipher system will be invalid differential characteristics.

The above approach can be used for searching impossible differential characteristic and zero-correlation linear distinguished (according to [CCJ+16]). When there is an invalid differential characteristic, Sasaki et al. [ST17] new search tool for impossible differential from the design and cryptanalysis aspects can be used. This paper shall not go in detail about the tool introduced specifically for impossible differential characteristics analysis.

4 Implementations

4.1 FPGA Implementation

FPGAs are awesome and getting more and more powerful day by day. The authors tested the designs on Virtex-7 FPGAs Xilinx a well know company for FPGAs. Below are the results, it's a trivial choice to make when using FPGAs to implement high throughput architectures because of the support provided by flipflops.[BJK+16] You can see the results of FPGA implementation in Figure 19 however it is tough to compare as many of the implementations are not tried in the same conditions or same architecture.

	Logic	**Memory**	**Frequency**	**T'put**	**Device**	**Ref.**
	LUT	FF	MHz	GBit/s	Xilinx	
SKINNY-64-64	3101	4000	403.88	25.85	Virtex-7	**New**
SKINNY-64-128	4247	6720	402.41	25.75	Virtex-7	**New**
SKINNY-64-192	6330	9952	400.48	25.63	Virtex-7	**New**
SKINNY-128-128	13389	10048	320.10	40.97	Virtex-7	**New**
SKINNY-128-256	17037	18048	355.62	45.52	Virtex-7	**New**
SKINNY-128-384	21966	28096	356.51	45.63	Virtex-7	**New**
ICEBERG-64-128	13616	-	297.00	19.01	Virtex-II	[40]
MISTY1-64-128	10920	8480	140.00	8.96	Virtex1000	[41]
KHAZAD-64-128	11072	9600	123.00	7.87	Virtex1000	[41]

4.2 Software Implementation

Skinny is made for hardware, so to use it in software bit-sliced implementation seems to be the most obvious choice The figure below provides results on different versions of micro-architecture with respect other ciphers from [BJK+16].

	Haswell			**Skylake**			**Ref.**
Parallelization ρ	16	32	64	16	32	64	
SKINNY-64-128	-	-	2.58	-	-	2.48	**New**
SIMON-64-128	-	-	1.58	-	-	1.51	[48]
LED-128	22.6	13.7	-	23.1	13.3	-	[4]
PRESENT-128	10.8	-	-	10.3	-	-	[4]
Piccolo-128	9.2	-	-	9.2	-	-	[4]
SKINNY-128-128	-	-	3.78	-	-	3.43	**New**
SIMON-128-128	-	-	2.38	-	-	2.21	[48]

4.3 Web Application

We have a web application that takes a file as input and encrypts with three mode of operations ECB, CBC, CTR. It takes the password and generates a key using PBKDF2.

We also setup a free SSL certificate from Let's Encrypt for the Web Application to secure the connection and ensure that we are connected to the right server.

5 And then there were some...

Brownie Section
1). Diagrams and Flowcharts: We have delicately curated flowcharts and diagrams that were absent in the initial proposal for ease of understanding of SKINNY.
2). Readable and Variant Independent Python Implementation: Our Python implementation of SKINNY is extremely comapct and readable. It also includes support for 3 modes of operation namely, **E**lectronic **C**ode **Book**, **C**ipher **B**lock **C**haining, and **C**oun**T**e**R** modes.
3). DDT and S-Box analysis: Please find the codes used in DDT and S-Box analysis here.
4). Security Analysis: We have provided discussions and security arguments on attacks other than differential and integral taking inspiration from [BJK+16].
5). Performance: We have discussed performance benchmarking and other comparisons for various implementations of SKINNY including FPGA, ASIC and Software.
6). Testing: We have implemented a test-suite using test vectors from the official website of the ***SKINNY*** cipher. Codes are included with the implementation code.
7). Crypto-Application: For this crypto-application, we have built a client-server application using Streamlit for the user interface. User may upload a file and provide a password to encrypt the file. Please find the application deployed here. It's code may be found here.

6 Acknowledgement

We would like to express our most sincere gratitude towards the efforts put in by the entire teaching team led by Dr. Dhiman Saha towards our crypto education. Special thanks to Sahiba Ma'am, Souvik Sir, and Arti Ma'am. This work is a culmination of all our efforts.

References

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The skinny family of block ciphers and its low-latency variant mantis. In *Annual International Cryptology Conference*, pages 123–153. Springer, 2016.

[BR14] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Designs, codes and cryptography*, 70(3):369–383, 2014.

[CCJ+16] Tingting Cui, Shiyao Chen, Keting Jia, Kai Fu, and Meiqin Wang. New automatic search tool for impossible differentials and zero-correlation linear approximations. *Cryptology ePrint Archive*, 2016.

[O'C94] Luke O'Connor. Properties of linear approximation tables. In *International Workshop on Fast Software Encryption*, pages 131–136. Springer, 1994.

[SS18] Sumanta Sarkar and Habeeb Syed. Bounds on differential and linear branch number of permutations. In *Australasian Conference on Information Security and Privacy*, pages 207–224. Springer, 2018.

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 185–215. Springer, 2017.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 287–314. Springer, 2015.