Lightweight circuits with Shift and Swap

Subhadeep Banik[1], Francesco Regazzoni[2] and Serge Vaudenay[1]

[1] LASEC, École Polytechnique Fédérale de Lausanne, Switzerland, {subhadeep.banik,serge.vaudenay}@epfl.ch
[2] University of Lugano, Switzerland regazzoni@alari.ch

Abstract. In CHES 2017, Moradi et al. presented a paper on "Bit-Sliding" in which the authors proposed lightweight constructions for SPN based block ciphers like AES, PRESENT and SKINNY. The main idea behind these constructions was to reduce the length of the datapath to 1 bit and to reformulate the linear layer for these ciphers so that they require fewer scan flip-flops (which have built-in multiplexer functionality and so larger in area as compared to a simple flip-flop). In this paper we take the idea forward: is it possible to construct the linear layer using only 2 scan flip-flops? Take the case of PRESENT: in the language of mathematics, the above question translates to: can the PRESENT permutation be generated by some ordered composition only two types of permutations? The question can be answered in the affirmative by drawing upon the theory of permutation groups. However straightforward constructions would require that the "ordered composition" consist of a large number of simpler permutations. This would naturally take a large number of clock cycles to execute in a flip-flop array having only two scan flip-flops and thus incur heavy loss of throughput.
In this paper we try to analyze SPN ciphers like PRESENT and GIFT that have a bit permutation as their linear layer. We tried to construct the linear layer of the cipher using as little clock cycles as possible. As an outcome we propose smallest known constructions for PRESENT and GIFT block ciphers for both encryption and combined encryption+decryption functionalities. We extend the above ideas to propose the first known construction of the FLIP stream cipher.

Keywords: Lightweight circuit, PRESENT, GIFT, FLIP

1 Introduction

The block cipher family Katan [CDK09] and then later Simon [BSS+] were in some sense aimed to achieve a lower limit of lightweight encryption in terms of area occupied in silicon. Both these ciphers have shift register based update functions, which is efficient to implement in ASIC when the length of datapath is reduced to one bit. In CHES 2017, Moradi et al. presented the concept of "Bit-Sliding" [JMPS17]. The authors proposed lightweight constructions for SPN based block ciphers like AES [DR02], PRESENT [BKL+07] and SKINNY [BJK+16] that also had a datapath width of 1 bit. This was counter-intuitive because the block ciphers in question used 8/4-bit S-boxes and it was not immediately clear how the width of the data-path could be made smaller than the size of the S-box that the block cipher was employing. The main idea behind these constructions was to reformulate the linear layer for these ciphers so that they require fewer scan flip-flops (which have built-in multiplexer functionality at the input port and so larger in area as compared to a simple flip-flop). In particular, the PRESENT linear layer which is essentially a bit permutation over the state, was decomposed as $P_2^4 \circ P_1$, where P_1 was a permutation that operated on each 16-bit block of the 64-bit state and P_2 is some other permutation. This decomposition allowed the authors of [JMPS17] to implement the linear layer using only

24 scan flip-flops and 40 regular flip-flops, whereas previous implementations [RPLP08] have required all 64 flip-flops holding the state to have additional multiplexer at its input.

Contribution: Thus the main idea behind [JMPS17] was that the fewer scan flip-flops one uses to construct the circuit is likely to translate into a lowering of the total hardware area of the circuit. Taking this idea forward, in this paper we try to answer is it possible to construct the linear layer if only 2 of the 64 flip-flops used to store the state are scan flip-flops? The question can be answered in the affirmative by drawing upon the theory of permutation groups and the methods presented in [Con]. However a straightforward application of the ideas in [Con] is computation intensive: for example we will subsequently show that to execute the PRESENT permutation P using just 2 scan flip-flops using the ideas in [Con] requires decomposing P into around 36000 simpler permutations which naturally will an equal number of clock cycles to implement in a flip-flop array. Hence such a construction would incur heavy lowering of throughput of the cipher which is very impractical. Much of the theory developed in this paper tries to investigate if we can speed-up the execution of P in the 2 scan flip-flop setup.

As a result of the theoretical foundations built in the paper, we construct lightweight implementations of the PRESENT and GIFT [BPP$^+$17] circuits for both encryption (E) and combined encryption+decryption (ED) modes. Both PRESENT and GIFT are block ciphers in which the linear layer is composed with a bit permutation over the internal state. In a particular configuration, the circuits of both PRESENT at 727 GE and GIFT at 925 GE are the smallest reported in literature so far. In the ED mode, the PRESENT and GIFT circuits occupy 809 GE and 1050 GE which are also the smallest reported thus far. (Note all circuits have been synthesized with the standard cell library CORE90GPHVT v 2.1.a of the STM 90nm CMOS logic process). We take the ideas forward and look at the stream cipher FLIP [MJSC16] whose core state update function is also a bit permutation. We propose three circuits for FLIP: the first is a direct implementation of the ideas in [Con]. This version however takes time proportional to the cube of the size of the secret key to produce a single keystream bit and is hence not practical. The second circuit we construct takes quadratic time and occupies only 3581 GE. The third circuit we propose uses slightly different ideas for bit swapping and can achieve the FLIP functionality in linear time. This circuit has an area of around 8605 GE. These are the first reported hardware implementations of FLIP.

Organization: The paper is organized in the following manner. In Section 2, we start with some preliminary definitions and notations along with a brief sketch of the proofs presented in [Con]. The main mathematical background is then developed in Section 3. This section is mainly concerned with the PRESENT block cipher. The theory built up in this section is done in various stages: in each stage we try to decrease the number permutations required to describe the the PRESENT bit permutation. Section 4 contains a circuit level description of the cipher along with a cycle by cycle operational details of its functions. Thereafter we extend these ideas to the GIFT block cipher in Section 5. Finally, we look at the circuit construction of FLIP in Section 6. Section 7 concludes the paper.

2 Preliminaries

We use the symbol S_n to denote the permutation group on n elements. Naturally we have, $|S_n| = n!$ and the group is non-commutative. A k-cycle $\pi \in S_n$ (for $1 \le k \le n$) is generally expressed as the k-tuple $(i_1, i_2, \ldots, i_k)$ which implies

- $\pi(i_1) = i_2,\ \pi(i_2) = i_3,\ \cdots, \pi(i_k) = i_1$, and

- $\pi(i) = i, \ \forall i \notin \{i_1, i_2, \dots, i_k\}$

This is a permutation of order equal to k. A transposition (or a swap) $\tau \in S_n$ is a 2-cycle. Denote by $\mathbb{A}_\pi$ the set $\{i_1, i_2, \dots, i_k\}$. In general, if π is a composition of several cycles of different orders, then define

$$\mathbb{A}_\pi = \{x : \ \pi(x) \neq x\}$$

The cycles π_1 and π_2 of orders k_1 and k_2 respectively are called disjoint if $\mathbb{A}_{\pi_1}$ and $\mathbb{A}_{\pi_2}$ are disjoint, i.e. have no elements in common. It is easy to see all disjoint cycles commute under the composition operation. It is well known that every permutation in S_n can be expressed as a composition of disjoint k-cycles, uniquely up to ordering of the k-cycles. To begin discussions, we cite a couple of results from [Con].

Lemma 1. **[Con, Theorem 2.1]** *For $n \geq 2$, S_n is generated by its transpositions.*

The above is not particularly difficult to prove. We know that the identity permutation can be written as τ^2 where τ is any transposition. Of course, any permutation can be expressed as compositions of k-cycles and any k-cycle $(i_1, i_2, \dots, i_k)$ can be written as $(i_1, i_2) \circ (i_2, i_3) \circ \dots \circ (i_{k-1}, i_k)$ and so the result follows.

Lemma 2. **[Con, Theorem 2.5]** *For $n \geq 2$, S_n is generated by the transposition $(1, 2)$ and the n-cycle $(1, 2, \dots, n)$.*

The proof of the above may be found in [Con], but for the benefit of the reader we give a small sketch. First note that the set $G_1 = \{(1, 2), \ (2, 3), \ \cdots, (n-1, n)\}$ also generates S_n, since any arbitrary transposition $(i, j) = (i, i+1) \circ (i+1, j) \circ (i, i+1)$. The first and third transpositions are already in G_1. If $|i + 1 - j| > 1$, then $(i + 1, j)$ can be further written as $(i+1, i+2) \circ (i+2, j) \circ (i+1, i+2)$, and so on till the term in the middle is in G_1. Given the following identity

$$\pi \circ (i_1, i_2, \dots, i_k) \circ \pi^{-1} = (\pi(i_1), \pi(i_2), \dots, \pi(i_k)),$$

for all k-cycles and $\pi \in S_n$, it is possible to show that any transposition of the form $(i, i+1)$ can be generated by $(1, 2)$ and the n-cycle $(1, 2, \dots, n)$. This is true since, if we denote $\sigma = (1, 2, \dots, n)$, then we have

$$\sigma^{i-1} \circ (1, 2) \circ \sigma^{-(i-1)} = (\sigma^{i-1}(1), \sigma^{i-1}(2)) = (i, i+1).$$

This completes the proof.

3 Application to PRESENT

The bit-permutation layer in PRESENT is given in Table 1. The round function specifies that the i-th state bit is moved to the $P(i)$-th position after application of the permutation layer. Let us look at the decomposition of P into its disjoint k-cycles. The disjoint decomposition of P consists of a total of twenty 3-cycles, since there are 4 fixed points. The 3-cycles are listed as follows:

- $(1, 16, 4)$, $(2, 32, 8)$, $(3, 48, 12)$, $(5, 17, 20)$, $(6, 33, 24)$,
- $(7, 49, 28)$, $(9, 18, 36)$, $(10, 34, 40)$, $(11, 50, 44)$, $(13, 19, 52)$,
- $(14, 35, 56)$, $(15, 51, 60)$, $(22, 37, 25)$, $(23, 53, 29)$, $(26, 38, 41)$,
- $(27, 54, 45)$, $(30, 39, 57)$, $(31, 55, 61)$, $(43, 58, 46)$, $(47, 59, 62)$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Table 1: Specifications of PRESENT bit-permutation layer.

i	c_i	$s_i \circ t_i$	i	c_i	$s_i \circ t_i$
0	$(1,16,4)$	$(4,16) \circ (1,4)$	10	$(14,35,56)$	$(14,35) \circ (35,56)$
1	$(2,32,8)$	$(8,32) \circ (2,8)$	11	$(15,51,60)$	$(15,51) \circ (51,60)$
2	$(3,48,12)$	$(12,48) \circ (3,12)$	12	$(22,37,25)$	$(25,37) \circ (22,25)$
3	$(5,17,20)$	$(5,17) \circ (17,20)$	13	$(23,53,29)$	$(29,53) \circ (23,29)$
4	$(6,33,24)$	$(24,33) \circ (6,24)$	14	$(26,38,41)$	$(26,38) \circ (38,41)$
5	$(7,49,28)$	$(28,49) \circ (7,28)$	15	$(27,54,45)$	$(45,54) \circ (27,45)$
6	$(9,18,36)$	$(9,18) \circ (18,36)$	16	$(30,39,57)$	$(30,39) \circ (39,57)$
7	$(10,34,40)$	$(10,34) \circ (34,40)$	17	$(31,55,61)$	$(31,55) \circ (55,61)$
8	$(11,50,44)$	$(44,50) \circ (11,44)$	18	$(43,58,46)$	$(46,58) \circ (43,46)$
9	$(13,19,52)$	$(13,19) \circ (19,52)$	19	$(47,59,62)$	$(47,59) \circ (59,62)$

Table 2: Decomposition of the c_i's in the PRESENT permutation

Let the above k-cycles be labeled by the symbols c_0 to c_{19}. Note that since all the c_i's are disjoint, the composition of all of them in any order will result in P. Each c_i may be further expressed as a composition of two transpositions: $c_i = s_i \circ t_i$. Table 2 lists all such decompositions explicitly.

Note that if we were to compose a permutation consisting of application of all the t_i's (in any order) followed by application of all the s_i's (again in any order) we would get back P. That is to say

$$P = s_{b_0} \circ s_{b_1} \circ \cdots \circ s_{b_{19}} \circ t_{a_0} \circ t_{a_1} \circ \cdots \circ t_{a_{19}},$$

where $a_0, a_1, \ldots a_{19}$ and $b_0, b_1, \ldots b_{19}$ are any arbitrary orderings of the set $\{0, 1, \ldots, 19\}$. We will prove a generalized form of the above statement in the following Lemma.

Lemma 3. *Let π be a permutation in S_n whose disjoint cycle decomposition consists of the cycles $c_0, c_1, \ldots, c_{m-1}$ each with orders $i_0, i_1 \ldots, i_{m-1}$ respectively (with $\sum_{j=0}^{m-1} i_j = n$), i.e.*

$$\pi = c_0 \circ c_1 \circ \cdots \circ c_{m-1}.$$

Let $i_0 \leq i_1 \leq \cdots \leq i_{m-1}$. Let each c_j be expressed as composition of $i_j - 1$ transpositions $s_j(1), s_j(2), \ldots, s_j(i_j - 1)$. So we have

$$\begin{array}{lccccccc}
s_{m-1}(i_{m-1}-1)\circ & \cdots\circ & \cdots & \circ\cdots\circ & s_{m-1}(2)\circ & s_{m-1}(1) & = & c_{m-1}\\
 & & & \vdots & & & & \\
 & s_j(i_j-1)\circ & \cdots & \circ\cdots\circ & s_j(2)\circ & s_j(1) & = & c_j\\
 & & & \vdots & & & & \\
 & & s_0(i_0-1) & \circ\cdots\circ & s_0(2)\circ & s_0(1) & = & c_0\\
\hline\hline
\textit{Sets:}\ \chi_{i_{m-1}-1} & \chi_{i_j-1} & \chi_{i_0-1} & \cdots & \chi_2 & \chi_1 & &
\end{array}$$

Define the set $\chi_k = \{s_{m-1}(k), s_{m-2}(k), \ldots\}$ (for $1 \le k < i_{m-1}$) as explained above. Let θ_k be the composition of all transpositions in χ_k in any arbitrary order. Then we must have

A *Each θ_k is invariant of the order in which the transpositions in χ_k are applied.*

B *We must have $\pi = \theta_{i_{m-1}-1} \circ \cdots \theta_{i_j-1} \circ \cdots \theta_2 \circ \theta_1$.*

Proof. We start with **A** as it is not difficult to prove. Note that c_j's are themselves disjoint decompositions of π. Thus it is easy to verify that any $s_{j_1}(a)$ and $s_{j_2}(b)$ will be disjoint for any $j_1 \neq j_2$ and any a, b. In particular, they are of course disjoint when $a = b$. This proves that all transpositions in any given χ_k are disjoint. Since disjoint cycles commute, composing the elements of χ_k in any order, gives the same permutation. This proves that θ_k is invariant with respect to ordering.

Denote by $\mu_j[x \to y] = s_j(x) \circ s_j(x-1) \circ \cdots \circ s_j(y)$ (for $x \ge y$). Naturally we have $\mu_j[i_j - 1 \to 0] = c_j$. Although c_j is a cycle of order i_j, for the completeness of the proof, let us define $s_j(i_j), s_j(i_j+1), \ldots, s_j(i_{m-1}-1)$ to be the identity permutation with $\mathbb{A}_{s_j(i_j)}, \mathbb{A}_{s_j(i_j+1)}, \ldots, \mathbb{A}_{s_j(i_{m-1}-1)}$ equal to $\varnothing$. With this definition we also have $\mu_j[i_{m-1} - 1 \to 0] = c_j$. Now to prove **B**, consider the following composition $\theta_2 \circ \theta_1$.

$$\begin{aligned}
\theta_2 \circ \theta_1 &= s_{m-1}(2) \circ s_{m-2}(2) \circ \cdots \circ s_0(2) \circ s_{m-1}(1) \circ s_{m-2}(1) \circ \cdots \circ s_0(1) && (1)\\
&= s_{m-2}(2) \circ \cdots \circ s_0(2) \circ (s_{m-1}(2) \circ s_{m-1}(1)) \circ s_{m-2}(1) \circ \cdots \circ s_0(1) && (2)\\
&= s_{m-2}(2) \circ \cdots \circ s_0(2) \circ \mu_{m-1}[2 \to 1] \circ s_{m-2}(1) \circ \cdots \circ s_0(1) && (3)\\
&= \mu_{m-1}[2 \to 1] \circ s_{m-2}(2) \circ \cdots \circ s_0(2) \circ s_{m-2}(1) \circ \cdots \circ s_0(1) && (4)\\
&= \mu_{m-1}[2 \to 1] \circ \mu_{m-2}[2 \to 1] \circ \cdots \circ \mu_0[2 \to 1] && (5)
\end{aligned}$$

(1) $\to$ (2) is true because all θ_k's are invariant to internal ordering of transpositions as proven in **A**. (2) $\to$ (3) follows from the definition of $\mu_j[x \to y]$. To prove (3) $\to$ (4), we start with the fact that $s_{j_1}(a)$ and $s_{j_2}(b)$ are disjoint for any $j_1 \neq j_2$ and any a, b, which is to say

$$\mathbb{A}_{s_{j_1}(a)} \cap \mathbb{A}_{s_{j_2}(b)} = \varnothing,\ \forall j_1 \neq j_2, \forall\ a, b$$

Therefore, we have, for all $j \in [0, m-2]$, the following relation:

$$\begin{aligned}
\mathbb{A}_{\mu_{m-1}[2\to1]} \cap \mathbb{A}_{s_j(2)} &= (\mathbb{A}_{s_{m-1}(2)} \cup \mathbb{A}_{s_{m-1}(1)}) \cap \mathbb{A}_{s_j(2)}\\
&= (\mathbb{A}_{s_{m-1}(2)} \cap \mathbb{A}_{s_j(2)}) \cup (\mathbb{A}_{s_{m-1}(1)} \cap \mathbb{A}_{s_j(2)})\\
&= \varnothing \cup \varnothing = \varnothing
\end{aligned}$$

This proves that $\mu_{m-1}[2 \to 1]$ is disjoint with all of $s_{m-2}(2), s_{m-3}(2) \ldots, s_0(2)$ and so (3) $\to$ (4) follows. (4) $\to$ (5) is just a generalization of steps (2), (3), (4) for the indices $m-2, m-3, \ldots, 0$. Proceeding as in mathematical induction, we can follow exactly the steps above to prove that $\theta_3 \circ \theta_2 \circ \theta_1 = \mu_{m-1}[3 \to 1] \circ \mu_{m-2}[3 \to 1] \circ \cdots \circ \mu_0[3 \to 1]$ and ultimately the fact that

$$\begin{aligned}
\theta_{i_{m-1}-1} \circ \theta_{i_{m-2}-1} \circ \cdots \circ \theta_1 &= \mu_{m-1}[i_{m-1} - 1 \to 1] \circ \mu_{m-2}[i_{m-1} - 1 \to 1] \circ \cdots \circ \mu_0[i_{m-1} - 1 \to 1]\\
&= c_{m-1} \circ c_{m-2} \circ \cdots \circ c_0 = \pi
\end{aligned}$$

□

The PRESENT permutation P follows a specific instance of the above lemma, with $m = 20$ and $i_0 = i_1 = \cdots = i_{19} = 3$. Thus the fact that

$$P = s_{b_0} \circ s_{b_1} \circ \cdots \circ s_{b_{19}} \circ t_{a_0} \circ t_{a_1} \circ \cdots \circ t_{a_{19}},$$

is a corollary of the above lemma.

3.1 Implementation using 2 scan flip-flops

We will now to try to implement the PRESENT permutation using only 2 scan flip-flops. In Lemma 2, we have proven that any permutation in S_n can be generated by the cycles $(1, 2)$ and $(1, 2, \ldots, n)$. By using the index notations commonly used in bock ciphers, we relabel the set of 64 elements by the indices $\{63, 62, \ldots, 0\}$. After this relabeling, we can analogously claim that S_{64} is generated by the cycles $w = (62, 63)$ and $r = (0, 1, 2, \ldots, 63)$. The idea is to implement all the transpositions t_i followed by all the s_i's. In order to do so let us first see how any arbitrary transposition can be implemented only using r and w.

Implementing a transposition (x, y) for($x > y$) and $x, y \in [0, 63]$: Let $\overline{x} = 63 - x$, $\overline{y} = 63 - y$. As per the proofs outlined in Lemmas 1 and 2, we have :

$$\begin{aligned}
(x, y) &= (x, x-1) \circ (x-1, y) \circ (x, x-1) \\
&= (x, x-1) \circ (x-1, x-2) \circ (x-2, y) \circ (x-1, x-2) \circ (x, x-1) \\
&= (x, x-1) \circ (x-1, x-2) \circ \cdots \circ (y+1, y) \circ \cdots \circ (x-1, x-2) \circ (x, x-1) \\
&= (r^{-\overline{x}} \circ w \circ r^{\overline{x}}) \circ (r^{-1-\overline{x}} \circ w \circ r^{1+\overline{x}}) \circ \cdots \circ (r^{1-\overline{y}} \circ w \circ r^{\overline{y}-1}) \circ \cdots \circ \\
&\quad (r^{-1-\overline{x}} \circ w \circ r^{1+\overline{x}}) \circ (r^{-\overline{x}} \circ w \circ r^{\overline{x}}) \\
&= r^{-\overline{x}} \circ w \circ (r^{-1} \circ w)^{x-y-1} \circ (r \circ w)^{x-y-1} \circ r^{\overline{x}} \\
&= r^{64-\overline{x}} \circ w \circ (r^{63} \circ w)^{x-y-1} \circ (r \circ w)^{x-y-1} \circ r^{\overline{x}} \\
&= r^{1+x} \circ w \circ (r^{63} \circ w)^{x-y-1} \circ (r \circ w)^{x-y-1} \circ r^{63-x}
\end{aligned}$$

Given the decomposition (x, y) in terms of r and w as given above, the next question naturally arises as to how to implement it using 2 scan flip-flops. Consider the circuit in Figure 1. It consists of an array of 64 flip-flops, with the 2 at the extreme ends being scan flip-flops controlled by a Sel signal. When Sel is 0, the data in the flip-flops simply rotate bitwise towards the left. When Sel is 1, the b_{63} bit is held in place, and the data in the remaining 63 flip-flops is rotated left bitwise. Implementing a particular permutation $\pi \in S_{64}$ on this circuit, essentially tries to answer the following question: If we consider $b_i(t),\ i \in [0, 63],\ t \geq 0$ to be the bit value stored on the i^{th} flip-flop at time t, does there exist some sequence of Sel signals $s_0, s_1, \ldots, s_{T-1}$ such that for all $b_0(0), \ldots, b_{63}(0)$, setting Sel to s_t at clock cycle t implies that $b_{\pi(i)}(T) = b_i(0)$ for all i. The length T of the sequence is the number of clock cycles needed to perform the permutation π.

Lemma 4. *Consider the circuit in Figure 1. Implementing an arbitrary swap operation (x, y) using it requires at most $64(x - y)$ clock cycles.*

Proof. To begin with, note that r is a function that performs a rotation operation by one location towards the left. In Figure 1, setting the select signal Sel to 0, causes the shift register to implement the r function, as data follows the circular path marked in the bottom. Setting Sel to 1, brings about the following transformation:

$$(b_{63}, b_{62}, b_{61}, \ldots, b_1, b_0) \rightarrow (b_{63}, b_{61}, b_{60}, \ldots, b_0, b_{62})$$

This is same as applying the function $(r \circ w)$. It is easy to see that r and $(r \circ w)$ also generate S_{64}. Thus by controlling the Sel signal, we can make the shift register circuit

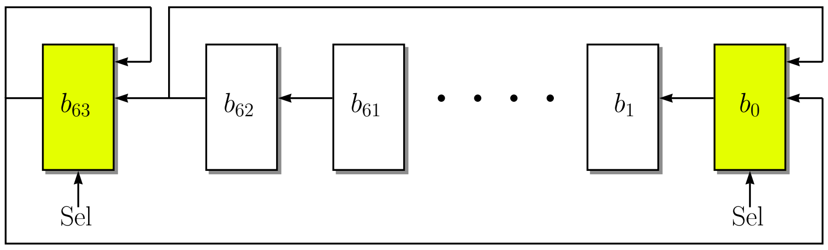


Figure 1: Shift register circuit with 2 scan flip-flops

alternate between r and $v = (r \circ w)$ functions. Note that (x, y) can be rewritten in blocks of 64 operations each, in the following manner:

$$\begin{aligned}(x,y) &= r^{1+x} \circ w \circ (r^{63} \circ w)^{x-y-1} \circ (r \circ w)^{x-y-1} \circ r^{63-x} \\ &= \left[r^{x} \circ v \circ r^{63-x}\right] \circ \left[r^{x-1} \circ v \circ r^{64-x}\right] \circ \cdots \circ \left[r^{y+2} \circ v \circ r^{61-y}\right] \circ \left[r^{y+1} \circ v^{x-y} \circ r^{63-x}\right]\end{aligned}$$

Each block of operations in square braces in the above equation is a set of 64 operations, and thus would take 64 clock cycles to execute using the shift register circuit. Since there are a total of $(x - y)$ braces, the result follows. $\square$

Corollary 1. *Employing the shift register circuit in figure 1, one round of the* PRESENT *bit permutation can be executed in 36480 clock cycles.*

Proof. The idea is to execute the PRESENT permutation P by executing each of the transpositions t_i and then s_i sequentially. Denoting $t_i = (x_i, y_i)$ and $s_i = (x_{20+i}, y_{20+i})$ for $i \in [0, 19]$, (with $x_i > y_i$) the number of clock cycles can be calculated as $\sum_{i=0}^{39} 64 \cdot (x_i - y_i) = 36480$. $\square$

The above result is a pessimistic one since it implies that to perform the PRESENT encryption operation on a shift register based circuit as given in Figure 1, would result in heavy loss of throughput. In the following subsections, we will try to see if the number of operations can be reduced in any way.

3.2 Decreasing the number of operations

Before we outline the method used to reduce the number of operations, let us look at the following definition.

Definition 1. As in Lemma 4, let π be a permutation in S_n whose disjoint cycle decomposition consists of the cycles $c_0, c_1, \ldots, c_{m-1}$ each with orders $i_0, i_1 \ldots, i_{m-1}$ respectively. Let each c_j be expressed as composition of $i_j - 1$ transpositions $s_j(1), s_j(2), \ldots, s_j(i_j - 1)$. Denote the transposition $s_j(k) = (x_j(k), y_j(k))$ with $x_j(k) > y_j(k)$. π is said to be a special permutation of the type κ, if κ is the largest integer for which the following holds:

$$x_j(k) - y_j(k) \equiv 0 \bmod \kappa, \quad \forall\, j \in [0, m-1], \forall\, k \in [0, i_j - 1]$$

It is easy to see from Table 2, that the PRESENT permutation P is a special permutation of type 3. Before we proceed, let us look at a result concerning special permutations of type κ.

Lemma 5. *Let G_κ denote the set of all the special permutations of S_{64} of type κ. Then G_κ can be generated by the permutations $w_\kappa = (63 - \kappa, 63)$ and $r = (0, 1, \ldots, 63)$.*

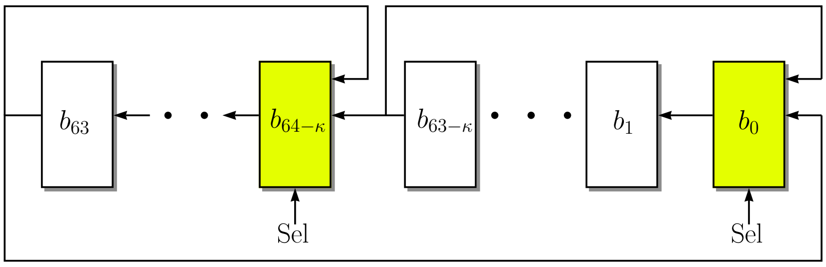


Figure 2: Shift register circuit with 2 scan flip-flops

Proof. The only thing we need to show is that any transposition (x, y) with $x > y$ and $x \equiv y \bmod \kappa$, can be generated using w_κ and r. Let $z = \frac{x-y}{\kappa}$. We have

$$\begin{aligned}
(x, y) &= (x, x-\kappa) \circ (x-\kappa, y) \circ (x, x-\kappa) \\
&= (x, x-\kappa) \circ (x-\kappa, x-2\kappa) \circ (x-2\kappa, y) \circ (x-\kappa, x-2\kappa) \circ (x, x-\kappa) \\
&= (x, x-\kappa) \circ (x-\kappa, x-2\kappa) \circ \cdots \circ (y+\kappa, y) \circ \cdots \circ (x-\kappa, x-2\kappa) \circ (x, x-\kappa) \\
&= (r^{-\overline{x}} \circ w_\kappa \circ r^{\overline{x}}) \circ (r^{-\kappa-\overline{x}} \circ w_\kappa \circ r^{\kappa+\overline{x}}) \circ \cdots \circ (r^{\kappa-\overline{y}} \circ w_\kappa \circ r^{\overline{y}-\kappa}) \circ \cdots \circ \\
&\quad (r^{-\kappa-\overline{x}} \circ w_\kappa \circ r^{\kappa+\overline{x}}) \circ (r^{-\overline{x}} \circ w_\kappa \circ r^{\overline{x}}) \\
&= r^{-\overline{x}} \circ w_\kappa \circ (r^{-\kappa} \circ w_\kappa)^{z-1} \circ (r^{\kappa} \circ w_\kappa)^{z-1} \circ r^{\overline{x}} \\
&= r^{64-\overline{x}} \circ w_\kappa \circ (r^{64-\kappa} \circ w_\kappa)^{z-1} \circ (r^{\kappa} \circ w_\kappa)^{z-1} \circ r^{\overline{x}} \\
&= r^{1+x} \circ w_\kappa \circ (r^{64-\kappa} \circ w_\kappa)^{z-1} \circ (r^{\kappa} \circ w_\kappa)^{z-1} \circ r^{63-x}
\end{aligned}$$

The next step naturally is too see how any transposition (x, y) with $x \equiv y \bmod \kappa$ can be implemented in a shift register structure using only 2 scan flip-flops using a method that requires lesser number of cycles as compared to the previous construction. We try to address this is the next lemma.

Lemma 6. *Consider the circuit in Figure 2. Implementing an arbitrary swap operation (x, y) with $x > y$ and $x \equiv y \bmod \kappa$ using it can be implemented in $\frac{64(x-y)}{\kappa} = 64z$ clock cycles.*

Proof. As before, setting Sel to 0, executes the rotate function r. Setting Sel to 1, achieves the following transformation:

$$(b_{63}, b_{62}, b_{61}, \ldots, b_1, b_0) \to (b_{62}, b_{61}, \ldots, b_{64-\kappa}, b_{63},\ b_{62-\kappa}, b_{61-\kappa}, \ldots, b_0, b_{63-\kappa})$$

This is same as applying the transformation $v_\kappa = r \circ w_\kappa$. Thus, as before, controlling Sel makes the circuit alternate between r and v_κ operations. As before we express (x, y) in blocks of 64 operations:

$$\begin{aligned}
(x, y) &= r^{1+x} \circ w_\kappa \circ (r^{64-\kappa} \circ w_\kappa)^{z-1} \circ (r^{\kappa} \circ w_\kappa)^{z-1} \circ r^{63-x} \\
&= [r^{x} \circ v_\kappa \circ r^{63-x}] \circ [r^{x-\kappa} \circ v_\kappa \circ r^{63-x+\kappa}] \circ \cdots \circ [r^{x-(z-2)\kappa} \circ v_\kappa \circ r^{63-x+(z-2)\kappa}] \circ \\
&\quad [r^{y+1} \circ (r^{\kappa-1} \circ v_\kappa)^{z} \circ r^{63-x}]
\end{aligned}$$

Operations in each of the square braces takes 64 cycles and since there are exactly z such braces, the result follows. □

Corollary 2. *Using the shift register circuit in Figure 2, one round of the* PRESENT *bit permutation P can be executed in 12160 clock cycles.*

Proof. We have already noted that P is a special permutation of type 3. As in the previous corollary, let $t_i = (x_i, y_i)$ and $s_i = (x_{20+i}, y_{20+i})$ for $i \in [0, 19]$, (with $x_i > y_i$). For performing all the t_i's followed by all the s_i's sequentially, the number of clock cycles can be calculated as $\sum_{i=0}^{39} 64 \cdot \frac{(x_i - y_i)}{3} = 12160$. $\square$

By using the modified shift register structure, we obtain a threefold increase of throughput in computation of the PRESENT permutation. However, this is still way too slow, and in the subsequent sections we will try to find if the computations can be further sped up.

3.3 Further reduction

Till now, we were executing each transposition operation sequentially, i.e. one after the other. However in the interest of speeding up computations, let us investigate if it is at all possible to execute some of the swap operations concurrently.

Definition 2. Define Let $\sigma = (x, y)$ be a transposition in S_{64} with $x > y$. $\overrightarrow{\mathsf{Sel}}_\sigma$ to be the vector of Sel signals that achieves the computation of σ using the circuit in Figure 2. The length of $\overrightarrow{\mathsf{Sel}}_\sigma$ is therefore $\frac{64(x-y)}{\kappa}$. For example, let $\kappa = 3$, as in PRESENT. Consider $\sigma = (60, 51)$, for which $z = 3$. We have

$$\begin{aligned}
\sigma &= [r^x \circ v_\kappa \circ r^{63-x}] \circ [r^{x-\kappa} \circ v_\kappa \circ r^{63-x+\kappa}] \circ \cdots \circ [r^{x-(z-2)\kappa} \circ v_\kappa \circ r^{63-x+(z-2)\kappa}] \circ \\
&\quad [r^{y+1} \circ (r^{\kappa-1} \circ v_\kappa)^z \circ r^{63-x}] \\
&= [r^{60} \circ v_3 \circ r^3] \circ [r^{57} \circ v_3 \circ r^6] \circ [r^{52} \circ (r^2 \circ v_3)^3 \circ r^3] \\
\overrightarrow{\mathsf{Sel}}_\sigma &= \underset{\longleftarrow \text{ Increasing Index}}{\underleftarrow{0^{60}\ 1\ 0^3 \quad 0^{57}\ 1\ 0^6 \quad 0^{52}\ 0^2 1\ 0^2 1\ 0^2 1\ 0^3}}
\end{aligned}$$

Note that to keep notations consistent as to the order of application of the permutations, the rightmost element in the vector is denoted as the 0^{th} element, and the index is increased as we go left. This is consistent with the order of application in the composition notation, in which the rightmost permutation of the composition is applied first. Let us now re-write the permutations r and v_κ in functional form:

$$r(\alpha) = (\alpha + 1) \bmod 64, \quad v_\kappa(\alpha) = \begin{cases} 64 - \kappa, & \text{if } \alpha = 63, \\ 0, & \text{if } \alpha = 63 - \kappa, \\ (\alpha + 1) \bmod 64, & \text{otherwise.} \end{cases}$$

We can see that r and v_κ differ on only two inputs 63 and $63 - \kappa$. By stretching notations slightly, let $\overrightarrow{\mathsf{Sel}}_p$ also denote a random 64 bit binary vector that implements the permutation p when fed to the Sel port of the circuit in Figure 2 over 64 consecutive clock cycles. Let $\mathbb{B}_p$ be the set of elements that denote the positions of 1's in $\overrightarrow{\mathsf{Sel}}_p$. From the functional equations of r and v_κ, it is not difficult to deduce that (a simple code in programming language is sufficient to do this) $\mathbb{A}_p = \mathbb{U}_p \cup \mathbb{V}_p$, where

$$\mathbb{U}_p = \{63 - \alpha : \alpha \in \mathbb{B}_p\}, \ \mathbb{V}_p = \{63 - \alpha - \kappa \bmod 64 : \alpha \in \mathbb{B}_p\}$$

It is also possible to deduce p from $\mathbb{B}_p$. If $\mathbb{B}_p$ contains elements $b, b+\kappa, b+2\kappa, \ldots, b+(l-1)\kappa$ which are in an arithmetic sequence with common difference κ then we will have

$$p(63 - b - i\kappa) = 63 - b - (i-1)\kappa, \ \forall i \in [1, l], \text{ and } \ p(63 - b) = 63 - b - l\kappa$$

For all other elements $\hat{b}$ in $\mathbb{B}_p$ that are not part of an arithmetic sequence with common difference κ, we have $p(63 - \hat{b}) = 63 - \hat{b} - \kappa$ and $p(63 - \hat{b} - \kappa) = 63 - \hat{b}$. For all other elements we have $p(b) = b$.

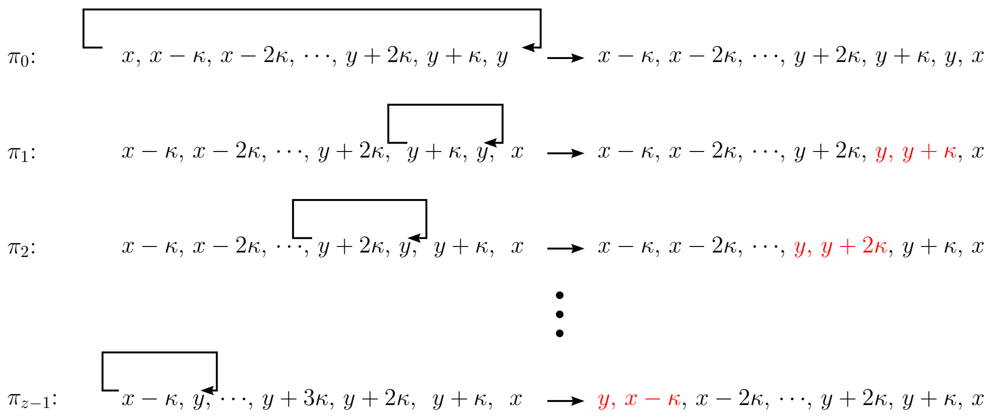

π_0:
$x,\ x-\kappa,\ x-2\kappa,\ \cdots,\ y+2\kappa,\ y+\kappa,\ y$
$x-\kappa,\ x-2\kappa,\ \cdots,\ y+2\kappa,\ y+\kappa,\ y,\ x$
π_1:
$x-\kappa,\ x-2\kappa,\ \cdots,\ y+2\kappa,\ y+\kappa,\ y,\ x$
$x-\kappa,\ x-2\kappa,\ \cdots,\ y+2\kappa,\ y,\ y+\kappa,\ x$
π_2:
$x-\kappa,\ x-2\kappa,\ \cdots,\ y+2\kappa,\ y,\ y+\kappa,\ x$
$x-\kappa,\ x-2\kappa,\ \cdots,\ y,\ y+2\kappa,\ y+\kappa,\ x$
π_{z-1}:
$x-\kappa,\ y,\ \cdots,\ y+3\kappa,\ y+2\kappa,\ y+\kappa,\ x$
$y,\ x-\kappa,\ x-2\kappa,\ \cdots,\ y+2\kappa,\ y+\kappa,\ x$

Figure 3: Rotations in each π_i

Example 1. For example if $\mathbb{B}_p = \{6, 9, 19, 29, 53, 56, 60, 61\}$ with $\kappa = 3$, we see that we have 2 arithmetic sequences of common difference 3: $6, 9$ and $53, 56$. So we have $\mathbb{A}_p = \{0, 2, 3, 4, 7, 10, 31, 34, 41, 44, 51, 54, 57, 63\}$ We have $p = (51, 54, 57) \circ (4, 7, 10) \circ (44, 41) \circ (34, 31) \circ (3, 0) \circ (2, 63)$.

Consider every 64 bit block of the $\overrightarrow{\mathsf{Sel}}_\sigma$ vector. Let π_i (for $i = 0$ to $z-1$) be the composition of all the permutations in the i^{th} 64-bit block. Let us use the notation

$$\overrightarrow{\mathsf{Sel}}_\sigma = \overrightarrow{\mathsf{Sel}}_{\pi_{z-1}} || \overrightarrow{\mathsf{Sel}}_{\pi_{z-2}} || \cdots || \overrightarrow{\mathsf{Sel}}_{\pi_2} || \overrightarrow{\mathsf{Sel}}_{\pi_1} || \overrightarrow{\mathsf{Sel}}_{\pi_0}.$$

Of course we have $\sigma = \pi_{z-1} \circ \pi_{z-2} \circ \cdots \circ \pi_2 \circ \pi_1 \circ \pi_0$. In the above example, for $\sigma = (60, 51)$ we have $\mathbb{B}_{\pi_0} = \{3, 6, 9\}$, $\mathbb{B}_{\pi_1} = \{6\}$, $\mathbb{B}_{\pi_2} = \{3\}$. Generalizing the above we can see that $\mathbb{B}_{\pi_0} = \{63 - x, 63 - x - \kappa, \ldots, 63 - y - \kappa\}$. $\mathbb{B}_{\pi_0}$ only contains elements that are $\overline{x} = 63 - x \bmod \kappa$. And we have that $\mathbb{B}_{\pi_i} \subset \mathbb{B}_{\pi_0}$, $\forall\ i > 0$. From the analysis presented above, it can be deduced that for all i,

$$\pi_i(\alpha) = \alpha,\ \forall \alpha\ \not\equiv x \bmod \kappa.$$

This is because the 1's (equivalently v_κ's) in this block appear at distances of κ. If we apply each function in π_i one by one, for any input $\alpha\ \not\equiv x \bmod \kappa$, the corresponding input to v_κ is never 63 or $63 - \kappa$, and so a plain rotation is effectively executed. Therefore all the π_i's perform shuffling on only a subset of elements that are $x \bmod \kappa$ and leave the others untouched. From the equation $\mathbb{A}_p = \mathbb{U}_p \cup \mathbb{V}_p$, we can also deduce that $\mathbb{A}_{\pi_0} = \{x, x - \kappa, x - 2\kappa, \ldots, y\}$. Thus each π_i is effectively a permutation function on only a subset of $\{0, 1, 2, 3, \ldots, 63\}$ that are $x \bmod \kappa$ and effectively follows the transition shown in Figure 3.

Figure 3 provides more insights into the working of each π_i. π_0 effectively rotates the elements in the set $\{x, x - \kappa, x - 2\kappa, \ldots, y\}$ by one location to the left, and delivers x to location y. The other π_i's are mini-swaps, that lets y bubble up to position x after $z - 1$ executions. From the figure we can see that $\mathbb{A}_{\pi_1 \circ \pi_0} = \mathbb{A}_{\pi_0} - \{y + \kappa\}$ and more generally, $\mathbb{A}_{\pi_i \circ \cdots \circ \pi_0} = \mathbb{A}_{\pi_{i-1} \circ \cdots \circ \pi_0} - \{y + i\kappa\}$.

Lemma 7. *Let $\overrightarrow{\mathsf{Sel}}_{p_1}$ and $\overrightarrow{\mathsf{Sel}}_{p_2}$ be two 64 bit signal vectors implementing permutations p_1 and p_2 on the circuit of Figure 2. If $\mathbb{A}_{p_1} \cap \mathbb{A}_{p_2} = \varnothing$, then $p_1 \circ p_2$ can be concurrently executed on this circuit using the signal vector $\overrightarrow{\mathsf{Sel}}_{p_1}\,\hat{}\,\overrightarrow{\mathsf{Sel}}_{p_2}$, where $\hat{}$ denotes a bitwise OR operation on the vectors.*

Proof. To begin with we have p_1 and p_2 disjoint, as $\mathbb{A}_{p_1} \cap \mathbb{A}_{p_2} = \varnothing$. Note that this implies $\mathbb{B}_{p_1} \cap \mathbb{B}_{p_2} = \varnothing$ (although the converse may not always be true). This means that the 1's in the $\overrightarrow{\mathsf{Sel}}_{p_1}$ and $\overrightarrow{\mathsf{Sel}}_{p_2}$ vectors are not aligned. Which is to say $\overrightarrow{\mathsf{Sel}}_{p_1} \hat{} \overrightarrow{\mathsf{Sel}}_{p_2}$ has 1's in all the locations in which either $\overrightarrow{\mathsf{Sel}}_{p_1}$ or $\overrightarrow{\mathsf{Sel}}_{p_2}$ has 1. Let $\overrightarrow{\mathsf{Sel}}_p = \overrightarrow{\mathsf{Sel}}_{p_1} \hat{} \overrightarrow{\mathsf{Sel}}_{p_2}$. We already know that $\mathbb{B}_p$ would contain all elements of $\mathbb{B}_{p_1}$ and $\mathbb{B}_{p_2}$. Thus the arithmetic sequence structures of both $\mathbb{B}_{p_1}$ and $\mathbb{B}_{p_2}$ are preserved in $\mathbb{B}_p$. Furthermore, $\mathbb{A}_{p_1} \cap \mathbb{A}_{p_2} = \varnothing$ ensures that no new arithmetic sequence of common difference κ is created $\mathbb{B}_p$ that are already not present in $\mathbb{B}_{p_1}$ or $\mathbb{B}_{p_2}$. We will prove this by contradiction: if possible let $\exists b_1 \in \mathbb{B}_{p_1},\ b_2 \in \mathbb{B}_{p_2}$ such that $b_2 = b_1 + \kappa$. Then by definition $63 - b_1, 63 - b_1 - \kappa \in \mathbb{A}_{p_1}$ and $63 - b_2, 63 - b_2 - \kappa \in \mathbb{A}_{p_2}$. But $63 - b_1 - \kappa = 63 - b_2$, and so this contradicts the fact that $\mathbb{A}_{p_1} \cap \mathbb{A}_{p_2} = \varnothing$. Since the arithmetic structures are preserved, p essentially executes p_1 and p_2 concurrently: we have $\forall \alpha \in \mathbb{A}_{p_1}$, $p(\alpha) = p_1(\alpha)$ and $\forall \alpha \in \mathbb{A}_{p_2}$, $p(\alpha) = p_2(\alpha)$. Also $p(\alpha) = \alpha$ for all $\alpha \notin \mathbb{A}_{p_1} \cup \mathbb{A}_{p_2}$. Thus we have $p = p_1 \circ p_2$. □

Lemma 8. *Let $\sigma_1 = (x_1, y_1)$ and $\sigma_2 = (x_2, y_2)$ be two special transpositions ($x_i > y_i$, $i = 1, 2$) in S_{64} of type κ. Without loss of generality let $\ell_1 = (x_1 - y_1) \geq (x_2 - y_2) = \ell_2$, and $z_i = \frac{\ell_i}{\kappa}$. Let the respective decompositions are denoted by the symbols π_i and θ_i, i.e. $\sigma_1 = \pi_{z_1-1} \circ \pi_{z_1-2} \circ \cdots \circ \pi_2 \circ \pi_1 \circ \pi_0$ and $\sigma_2 = \theta_{z_2-1} \circ \theta_{z_2-2} \circ \cdots \circ \theta_2 \circ \theta_1 \circ \theta_0$. $\overrightarrow{\mathsf{Sel}}_{\sigma_1}$ and $\overrightarrow{\mathsf{Sel}}_{\sigma_2}$ may not be of the same length, in which case append $64(z_1 - z_2)$ zeroes to $\overrightarrow{\mathsf{Sel}}_{\sigma_2}$[1] to make them of the same length. If $\mathbb{A}_{\pi_0} \cap \mathbb{A}_{\theta_0} = \varnothing$, then it is possible to execute σ_1 and σ_2 concurrently on the circuit in Figure 2 and achieve $\sigma_1 \circ \sigma_2$ in $64 \cdot z_1$ clock cycles. Let $\overrightarrow{\mathsf{Sel}}_{\sigma_1 \circ \sigma_2}$ be the vector of Sel signals required to achieve this. Then $\overrightarrow{\mathsf{Sel}}_{\sigma_1 \circ \sigma_2} = \overrightarrow{\mathsf{Sel}}_{\sigma_1} \hat{} \overrightarrow{\mathsf{Sel}}_{\sigma_2}$.*

Proof. Since $\mathbb{A}_{\pi_0} \cap \mathbb{A}_{\theta_0} = \varnothing$, from the result of the previous lemma, we can certainly use $\overrightarrow{\mathsf{Sel}}_{\pi_0} \hat{} \overrightarrow{\mathsf{Sel}}_{\theta_0}$ to get $\pi_0 \circ \theta_0$. Since all $\mathbb{A}_{\pi_i}$'s and $\mathbb{A}_{\theta_i}$'s are subsets of $\mathbb{A}_{\pi_0}$ and $\mathbb{A}_{\theta_0}$ respectively, we also have $\mathbb{A}_{\pi_i} \cap \mathbb{A}_{\theta_i} = \varnothing$ for all $0 \leq i \leq z_1 - 1$. We can then use $\overrightarrow{\mathsf{Sel}}_{\pi_i} \hat{} \overrightarrow{\mathsf{Sel}}_{\theta_i}$ to get $\pi_i \circ \theta_i$ for all $0 \leq i \leq z_1 - 1$. Thus if $\overrightarrow{\mathsf{Sel}}_p = \overrightarrow{\mathsf{Sel}}_{\sigma_1} \hat{} \overrightarrow{\mathsf{Sel}}_{\sigma_2}$, we naturally have

$$p = (\pi_{z_1-1} \circ \theta_{z_1-1}) \circ (\pi_{z_1-2} \circ \theta_{z_1-2}) \circ \cdots \circ (\pi_1 \circ \theta_1) \circ (\pi_0 \circ \theta_0)$$

Denote by $\pi[i \to j] = \pi_i \circ \pi_{i-1} \circ \cdots \circ \pi_0$ and $\theta[i \to j] = \theta_i \circ \theta_{i-1} \circ \cdots \circ \theta_0$. Note that $\mathbb{A}_{\pi[i_1 \to j_1]} \cap \mathbb{A}_{\theta[i_2 \to j_2]} = \varnothing$, since the parent sets $\mathbb{A}_{\pi_0}$ and $\mathbb{A}_{\theta_0}$ are themselves disjoint. So we have

$$\begin{aligned}
p &= (\pi_{z_1-1} \circ \theta_{z_1-1}) \circ (\pi_{z_1-2} \circ \theta_{z_1-2}) \circ \cdots \circ (\pi_1 \circ \theta_1 \circ \pi_0 \circ \theta_0) && (6)\\
&= (\pi_{z_1-1} \circ \theta_{z_1-1}) \circ (\pi_{z_1-2} \circ \theta_{z_1-2}) \circ \cdots \circ (\theta_1 \circ \pi_1 \circ \pi_0 \circ \theta_0) && (7)\\
&= (\pi_{z_1-1} \circ \theta_{z_1-1}) \circ (\pi_{z_1-2} \circ \theta_{z_1-2}) \circ \cdots \circ (\theta_1 \circ \pi[1 \to 0] \circ \theta_0) && (8)\\
&= (\pi_{z_1-1} \circ \theta_{z_1-1}) \circ (\pi_{z_1-2} \circ \theta_{z_1-2}) \circ \cdots \circ (\theta_1 \circ \theta_0 \circ \pi[1 \to 0]) && (9)\\
&= (\pi_{z_1-1} \circ \theta_{z_1-1}) \circ (\pi_{z_1-2} \circ \theta_{z_1-2}) \circ \cdots \circ (\theta[1 \to 0] \circ \pi[1 \to 0]) && (10)
\end{aligned}$$

$(6 \to 7)$ follows because $\mathbb{A}_{\pi_1} \cap \mathbb{A}_{\theta_1} = \varnothing$. $(8 \to 9)$ follows because $\mathbb{A}_{\pi[1 \to 0]} \cap \mathbb{A}_{\theta_0} = \varnothing$. The remaining statements follow from definition. The steps in the above equations can be repeated for $i = 2$ to $z_1 - 1$ to get $p = \pi[z_1 - 1 \to 0] \circ \theta[z_1 - 1 \to 0] = \sigma_1 \circ \sigma_2$. □

The above result may be extended to a set of any number of special transpositions σ_i ($i = 1$ to k) of the type κ, provided that the respective $\mathbb{A}_{\pi_0}$ sets are pairwise disjoint. In that case we have

$$\overrightarrow{\mathsf{Sel}}_{\sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_k} = \overrightarrow{\mathsf{Sel}}_{\sigma_1} \hat{} \overrightarrow{\mathsf{Sel}}_{\sigma_2} \hat{} \cdots \hat{} \overrightarrow{\mathsf{Sel}}_{\sigma_k}$$

Corollary 3. *Let $\sigma_1 = (x_1, y_1)$ and $\sigma_2 = (x_2, y_2)$ be two transpositions ($x_i > y_i$, $i = 1, 2$) in S_{64} such that $x_1 - y_1 \equiv x_2 - y_2 \bmod \kappa$, and $x_1 \not\equiv x_2 \bmod \kappa$. Without loss of generality let $\ell_1 = (x_1 - y_1) \geq (x_2 - y_2) = \ell_2$, and $z_i = \frac{\ell_i}{\kappa}$. As before, let the respective decompositions are denoted by the symbols π_i and θ_i and append $64(z_1 - z_2)$ zeroes to $\overrightarrow{\mathsf{Sel}}_{\sigma_2}$ to make the two $\overrightarrow{\mathsf{Sel}}$ vectors of the same length. It is possible to execute σ_1 and σ_2 concurrently on the*

[1]Since r^{64} is the identity function, this does not affect either permutation

circuit in Figure 2 and achieve $\sigma_1 \circ \sigma_2$ in $64 \cdot z_1$ clock cycles by using $\overrightarrow{\mathsf{Sel}}_{\sigma_1} \hat{} \ \overrightarrow{\mathsf{Sel}}_{\sigma_2}$ as the select signal vector.

Proof. We have already seen that for any transposition $\sigma = (x, y) = \pi_{z-1} \circ \cdots \circ \pi_0$, we have $\mathbb{A}_{\pi_0} = \{x, x - \kappa, x - 2\kappa, \ldots, y\}$. Thus $\mathbb{A}_{\pi_0}$ contains elements that are only $x \bmod \kappa$. Since x_1, y_1 and x_2, y_2 belong to different equivalence classes modulo κ, $\mathbb{A}_{\pi_0} \cap \mathbb{A}_{\theta_0} = \varnothing$. Thus the result follows. $\square$

Corollary 4. *Let $\sigma_1 = (x_1, y_1)$ and $\sigma_2 = (x_2, y_2)$ be two transpositions ($x_i > y_i$, $i = 1, 2$) in S_{64} such that $y_1 > x_2$. Without loss of generality let $\ell_1 = (x_1 - y_1) \geq (x_2 - y_2) = \ell_2$, and $z_i = \frac{\ell_i}{\kappa}$. Let the respective decompositions are denoted by the symbols π_i and θ_i. Then after making the $\overrightarrow{\mathsf{Sel}}$ vectors of the same length by appending zeroes, it is possible to execute σ_1 and σ_2 concurrently on the circuit in Figure 2 and achieve $\sigma_1 \circ \sigma_2$ in $64 \cdot z_1$ clock cycles by using $\overrightarrow{\mathsf{Sel}}_{\sigma_1} \hat{} \ \overrightarrow{\mathsf{Sel}}_{\sigma_2}$ as the select signal vector.*

Proof. We have $\mathbb{A}_{\pi_0} = \{x_1, x_1 - \kappa, x_1 - 2\kappa, \ldots, y_1\}$ and $\mathbb{A}_{\theta_0} = \{x_2, x_2 - \kappa, x_2 - 2\kappa, \ldots, y_2\}$. Since $y_1 > x_2$, clearly $\mathbb{A}_{\pi_0} \cap \mathbb{A}_{\theta_0} = \varnothing$. Thus the result follows. $\square$

We can use the results in the above two corollaries to further reduce the execution time of the PRESENT permutation. We have to execute all the transpositions t_i followed by the transpositions s_i. The idea is to execute as many permutations concurrently which have pairwise disjoint $\mathbb{A}_{\pi_0}$'s. We can easily partition the transpositions modulo $\kappa = 3$. Transpositions that are in different classes modulo 3 can obviously be executed concurrently. Also transpositions in the same class modulo 3, which have disjoint $\mathbb{A}_{\pi_0}$'s can also be executed together. For the t_i's we can think of the following solution given in Table 4, that takes $(11 + 7 + 1) \cdot 64 = 704 + 448 + 64 = 1216$ cycles. All the swaps in i^{th} group can be executed concurrently, thereby reducing the number of cycles.

Group	mod3	t_i	$\max(x_i - y_i)$	#Cycles
1	0	$(57,39),\ (36,18),\ (12,3)$	33	704
	1	$(61,55),\ (52,19),\ (4,1)$		
	2	$(62,59),\ (44,11),\ (8,2)$		
2	0	$(60,51),\ (45,27),\ (24,6)$	21	448
	1	$(46,43),\ (40,34),\ (28,7)$		
	2	$(56,35),\ (29,23),\ (20,17)$		
3	1	$(25,22)$	3	64
	2	$(41,38)$		

Table 3: Concurrent execution of the t_i's in the PRESENT permutation

A similar construction for the s_i's will take $(12 + 12 + 7 + 4) \cdot 64 = 2240$ cycles. So a total of $1216 + 2240 = 3456$ cycles are required which is already way better than our previous construction of 12160 cycles.

3.4 Final Optimization

In this final subsection we see if the number of clock cycles can be further optimized. Specifically we want to see if it is possible to implement transpositions σ_1 and σ_2 concurrently, even if the corresponding $\mathbb{A}_{\pi_0} \cap \mathbb{A}_{\theta_0} \neq \varnothing$. We start with a well known result in permutation theory.

Group	mod3	s_i	$\max(x_i - y_i)$	#Cycles
1	0	$(51, 15)$	36	768
	1	$(55, 31),\ (19, 13)$		
	2	$(53, 29),\ (17, 5)$		
2	0	$(48, 12)$	36	768
	1	$(58, 46),\ (34, 10)$		
	2	$(59, 47),\ (32, 8)$		
3	0	$(54, 45),\ (39, 30),\ (18, 9)$	21	448
	1	$(49, 28),\ (16, 4)$		
	2	$(50, 44),\ (35, 14)$		
4	0	$(33, 24)$	12	256
	1	$(37, 25)$		
	2	$(38, 26)$		

Table 4: Concurrent execution of the s_i's in the PRESENT permutation

Theorem 1. *For every permutation $\sigma \in S_{64}$, and every transposition $(x, y) \in S_{64}$:*

$$f = \sigma \circ (x, y) = (\sigma(x), \sigma(y)) \circ \sigma$$

The above is not difficult to prove, $\forall \alpha \notin \{x, y\}$, we have $f(\alpha) = \sigma(\alpha)$. And both sides evaluates to $f(x) = \sigma(y)$ and $f(y) = \sigma(x)$.

Lemma 9. *Let $\sigma_1 = (x_1, y_1)$ and $\sigma_2 = (x_2, y_2)$ be two special disjoint transpositions ($x_i > y_i$, $i = 1, 2$) in S_{64} of type κ. Without loss of generality let $\ell_1 = (x_1 - y_1) \geq (x_2 - y_2) = \ell_2$, and $z_i = \frac{\ell_i}{\kappa}$. Let the respective decompositions are denoted by the symbols π_i and θ_i, i.e. $\sigma_1 = \pi_{z_1-1} \circ \pi_{z_1-2} \circ \cdots \circ \pi_2 \circ \pi_1 \circ \pi_0$ and $\sigma_2 = \theta_{z_2-1} \circ \theta_{z_2-2} \circ \cdots \circ \theta_2 \circ \theta_1 \circ \theta_0$. Let us have $\mathbb{A}_{\pi_0} \cap \mathbb{A}_{\theta_0} \neq \varnothing$. Denote by $p = (\pi[i \to 0](x_2), \pi[i \to 0](y_2))$, for some $i \in [0, z_1 - 1]$. Let the decomposition of p be denoted as*

$$p = \gamma_{q-1} \circ \gamma_{q-2} \circ \cdots \circ \gamma_1 \circ \gamma_0.$$

Now denote $\overrightarrow{\mathsf{Sel}_1} = \overrightarrow{\mathsf{Sel}}_{\pi_{z_1-1}} || \overrightarrow{\mathsf{Sel}}_{\pi_{z_1-2}} || \cdots || \overrightarrow{\mathsf{Sel}}_{\pi_{i+1}}$ and $\overrightarrow{\mathsf{Sel}_2} = \overrightarrow{\mathsf{Sel}}_p$. After appending with zeroes to make $\overrightarrow{\mathsf{Sel}_1}$ and $\overrightarrow{\mathsf{Sel}_2}$ of the same length, the following vector

$$\overrightarrow{\mathsf{Sel}_1} \,\hat{}\, \overrightarrow{\mathsf{Sel}_2} \ ||\ \overrightarrow{\mathsf{Sel}}_{\pi[i \to 0]}$$

will execute $\sigma_1 \circ \sigma_2$ on the circuit in Figure 2, if $\mathbb{B}_{\gamma_0} \cap \mathbb{B}_{\pi_{i+1}} = \varnothing$.

Proof. First of all, let us clarify what we are trying to do. We want to implement

$$\begin{aligned}
\sigma_1 \circ \sigma_2 &= \pi_{z_1-1} \circ \pi_{z_1-2} \circ \cdots \circ \pi_2 \circ \pi_1 \circ \pi_0 \circ (x_2, y_2) \\
&= \pi_{z_1-1} \circ \pi_{z_1-2} \circ \cdots \circ \pi_{i+1} \circ \pi[i \to 0] \circ (x_2, y_2) \\
&= \pi_{z_1-1} \circ \pi_{z_1-2} \circ \cdots \circ \pi_{i+1} \circ (\pi[i \to 0](x_2), \pi[i \to 0](y_2)) \circ \pi[i \to 0] \\
&= \pi_{z_1-1} \circ \pi_{z_1-2} \circ \cdots \circ \pi_{i+1} \circ p \circ \pi[i \to 0]
\end{aligned}$$

We are therefore trying to implement $\pi_{z_1-1} \circ \pi_{z_1-2} \circ \cdots \circ \pi_{i+1} = \pi[z_1 - 1 \to i + 1]$ and p concurrently after implementing $\pi[i \to 0]$. Now $\mathbb{B}_{\pi_{i+1}}, \mathbb{B}_{\pi_{i+2}}, \ldots$ are singleton sets and so are $\mathbb{B}_{\gamma_1}, \mathbb{B}_{\gamma_2}, \ldots$. If $\mathbb{B}_{\gamma_0} = \{g_1, g_1+\kappa, g_1+2\kappa, \ldots, h_1\}$ and $\mathbb{B}_{\pi_{i+1}} = \{g_2\}$ are disjoint (note we have taken $h_1 > g_1$), then we have

$$\begin{aligned}\mathbb{B}_{\gamma_1} &= \{h_1 - \kappa\}, \ \ \mathbb{B}_{\pi_{i+2}} = \{g_2 - \kappa\} \\ \mathbb{B}_{\gamma_2} &= \{h_1 - 2\kappa\}, \ \ \mathbb{B}_{\pi_{i+3}} = \{g_2 - 2\kappa\} \\ &\vdots\end{aligned}$$

Thus $\mathbb{B}_{\gamma_j}$ and $\mathbb{B}_{\pi_{i+j+1}}$ are not only disjoint (for $j \geq 1$), but the distance between the single elements in the sets equals $g_2 - h_1$ which is a non-zero constant. Note that we have $g_1 \equiv h_1 \equiv g_2 \bmod \kappa$, since $\mathbb{A}_{\pi_0} \cap \mathbb{A}_{\theta_0} \neq \varnothing$. Since $\mathbb{B}_{\gamma_0} \cap \mathbb{B}_{\pi_{i+1}} = \varnothing$, we must have either $g_2 \geq h_1 + \kappa$ or $g_2 < g_1 - \kappa$. $g_2 = g_1 - \kappa$ is not possible as it leads to a contradiction: if $g_2 = g_1 - \kappa$, then the largest element in $\mathbb{B}_{\pi_0}$ is $g_1 + i\kappa$, and so $\sigma_1 = (u, 63 - g_1 - (i+1)\kappa)$, for some u. We have $p = (63 - h_1, 63 - g_1 - \kappa) = (63 - h_1, \pi[i \to 0](63 - g_1 - (i+1)\kappa))$, which means $\sigma_2 = (v, 63 - g_1 - (i+1)\kappa)$, for some v. This contradicts the fact that σ_1 and σ_2 are disjoint. Denote $z_3 = \pi[i \to 0](x_2) - \pi[i \to 0](y_2))$. We have

$$\begin{aligned}\mathbb{A}_{\gamma_0} = \{63 - g_1, 63 - g_1 - \kappa, \ldots, 63 - h_1, 63 - h_1 - \kappa\}, &\ \ \mathbb{A}_{\pi_{i+1}} = \{63 - g_2, 63 - g_2 - \kappa\} \\ \mathbb{A}_{\gamma_1} = \{63 - h_1 + \kappa, 63 - h_1\}, &\ \ \mathbb{A}_{\pi_{i+2}} = \{63 - g_2 + \kappa, 63 - g_2\} \\ \mathbb{A}_{\gamma_2} = \{63 - h_1 + 2\kappa, 63 - h_1 + \kappa\}, &\ \ \mathbb{A}_{\pi_{i+3}} = \{63 - g_2 + 2\kappa, 63 - g_2 + \kappa\} \\ &\vdots \\ \mathbb{A}_{\gamma_{q-1}} = \{63 - g_1, 63 - g_1 - \kappa\}.&\end{aligned}$$

Thus $\mathbb{A}_{\gamma_j}$ and $\mathbb{A}_{\pi_{i+j+1}}$ are non-disjoint (for $j \geq 0$) only if $g_2 = h_1 + \kappa$ or $g_2 = g_1 - \kappa$. Also note that

$$\mathbb{A}_{\pi[i+j+1 \to i+1]} = \{63 - g_2 - \kappa, 63 - g_2, 63 - g_2 + \kappa, \ldots, 63 - g_2 + j\kappa\}$$

We want to find $\mathbb{A}_{\pi[i+j+1 \to i+1]} \cap \mathbb{A}_{\gamma_j}$. The numerical maximum of $\mathbb{A}_{\pi[i+j+1 \to i+1]}$ is $63 - g_2 + j\kappa$ and numerical minimum of $\mathbb{A}_{\gamma_j}$ is $63 - h_1 + (j-1)\kappa$. The min - max difference comes out to be $g_2 - h_1 - \kappa$. If $g_2 > h_1 + \kappa$, this is always greater than 0 and so the sets are disjoint. If $g_2 < g_1 - \kappa$, then the minimal element of $\mathbb{A}_{\pi[i+j+1 \to i+1]}$, i.e. $63 - g_2 - \kappa > 63 - g_1$ which is the maximal element in the $\mathbb{A}_{\gamma_j}$'s. Here too the sets are disjoint. So we have three cases to analyze (A) $g_2 > h_1 + \kappa$ or $g_2 < g_1 - \kappa$, (B) $g_2 = h_1 + \kappa$. So let us split the analysis into two cases:

A: $g_2 > h_1 + \kappa$ or $g_2 < g_1 - \kappa$: We have $\mathbb{A}_{\gamma_j} \cap \mathbb{A}_{\pi_{i+j+1}} = \mathbb{B}_{\gamma_j} \cap \mathbb{B}_{\pi_{i+j+1}} = \varnothing$ for all $j \geq 0$. We also have $\mathbb{A}_{\pi[i+j+1 \to i+1]} \cap \mathbb{A}_{\gamma_j} = \varnothing$. This means that $\overrightarrow{\mathsf{Sel}_1}\,\hat{}\,\overrightarrow{\mathsf{Sel}_2}$ has 1's in locations where either $\overrightarrow{\mathsf{Sel}_1}$ or $\overrightarrow{\mathsf{Sel}_2}$ is 1. Let z be the final length of $\overrightarrow{\mathsf{Sel}_1}, \overrightarrow{\mathsf{Sel}_2}$ after padding. By Lemma 7, if $\overrightarrow{\mathsf{Sel}_\Pi} = \overrightarrow{\mathsf{Sel}_1}\,\hat{}\,\overrightarrow{\mathsf{Sel}_2}$, then

$$\begin{aligned}\Pi &= (\pi_{\mathsf{z}+i} \circ \gamma_{\mathsf{z}-1}) \circ (\pi_{\mathsf{z}+i-1} \circ \gamma_{\mathsf{z}-2}) \circ \cdots \circ (\pi_{i+2} \circ \gamma_1) \circ (\pi_{i+1} \circ \gamma_0) &(11)\\ &= (\pi_{\mathsf{z}+i} \circ \gamma_{\mathsf{z}-1}) \circ (\pi_{\mathsf{z}+i-1} \circ \gamma_{\mathsf{z}-2}) \circ \cdots \circ (\gamma_1 \circ \pi_{i+2}) \circ (\pi_{i+1} \circ \gamma_0) &(12)\\ &= (\pi_{\mathsf{z}+i} \circ \gamma_{\mathsf{z}-1}) \circ (\pi_{\mathsf{z}+i-1} \circ \gamma_{\mathsf{z}-2}) \circ \cdots \circ (\gamma_1 \circ \pi[i+2 \to i+1] \circ \gamma_0) &(13)\\ &= (\pi_{\mathsf{z}+i} \circ \gamma_{\mathsf{z}-1}) \circ (\pi_{\mathsf{z}+i-1} \circ \gamma_{\mathsf{z}-2}) \circ \cdots \circ (\pi[i+2 \to i+1] \circ \gamma_1 \circ \gamma_0) &(14)\\ &= (\pi_{\mathsf{z}+i} \circ \gamma_{\mathsf{z}-1}) \circ (\pi_{\mathsf{z}+i-1} \circ \gamma_{\mathsf{z}-2}) \circ \cdots \circ (\pi[i+2 \to i+1] \circ \gamma[1 \to 0]) &(15)\\ &= \pi[\mathsf{z}+i \to i+1] \circ \gamma[\mathsf{z}-1 \to 0] = \pi[\mathsf{z}+i \to i+1] \circ p &(16)\end{aligned}$$

$(11 \to 12)$ follows because $\mathbb{A}_{\gamma_j} \cap \mathbb{A}_{\pi_{i+j+1}} = \varnothing$ for all j. $(13 \to 14)$ follows because $\mathbb{A}_{\pi[i+j+1 \to i+1]} \cap \mathbb{A}_{\gamma_j} = \varnothing$ for all j. $(15 \to 16)$ follows after repeating $(11 \to 15)$ for $j = 0, 1, 2 \ldots$ etc. The remaining statements follow by definition.

B: $g_2 = h_1 + \kappa$**:** Before we analyze this case, let us restate a result in permutation theory

$$(x_{n_1}, x_{n_2}, \ldots, x_{n_l}) \circ (x_{n_l}, x_{n_{l+1}}, \ldots, x_{n_k}) = (x_{n_1}, x_{n_2}, \ldots, x_{n_l}, \ldots, x_{n_k}). \tag{17}$$

Since $g_2 = h_1 + \kappa$, the following is easy to verify (denote $\overline{h}_1 = 63 - h_1, \overline{g}_1 = 63 - g_1$)

$$\begin{aligned} \pi_{i+j+1} = &\quad (\overline{h}_1 + (j-2)\kappa,\ \overline{h}_1 + (j-1)\kappa), \text{ and} \\ \gamma_j = &\begin{cases} (\overline{h}_1 - \kappa,\ \overline{h}_1,\ \overline{h}_1 + \kappa,\ \ldots,\ \overline{g}_1), & \text{if } j = 0, \\ (\overline{h}_1 + (j-1)\kappa,\ \overline{h}_1 + j\kappa), & \text{otherwise.} \end{cases} \end{aligned}$$

By directly applying equation (17), we can obtain the following

$$\pi_{i+j+1} \circ \gamma_j = \begin{cases} (\overline{h}_1 - 2\kappa,\ \overline{h}_1 - \kappa,\ \overline{h}_1,\ \overline{h}_1 + \kappa,\ \ldots,\ \overline{g}_1), & \text{if } j = 0, \\ (\overline{h}_1 + (j-2)\kappa,\ \overline{h}_1 + (j-1)\kappa,\ \overline{h}_1 + j\kappa), & \text{otherwise.} \end{cases} \tag{18}$$

$$\pi[i+j+1 \to i+1] = (\overline{h}_1 + (j-1)\kappa,\ \ldots,\ \overline{h}_1,\ \overline{h}_1 - \kappa,\ \overline{h}_1 - 2\kappa)$$

From figure 3, (or by induction) it is easy to deduce that

$$\gamma[j \to 0] = (\overline{h}_1 - \kappa, \quad \overline{h}_1 + j\kappa,\ \overline{h}_1 + (j+1)\kappa, \ldots,\ \overline{g}_1)$$

From the above two equations we can deduce that

$$\pi[i+j+1 \to i+1] \circ \gamma[j \to 0] = (\overline{h}_1 - 2\kappa,\ \overline{h}_1 + (j-1)\kappa, \ldots, \overline{h}_1, \overline{h}_1 - \kappa, \overline{h}_1 + j\kappa, \ldots, \overline{g}_1) \tag{19}$$

Note that if we denote $\mathbb{B}_{q_j} = \mathbb{B}_{\gamma_j} \cup \mathbb{B}_{\pi_{i+j+1}}$, then $\mathbb{B}_{q_0} = \{g_1, g_1 + \kappa, \ldots, h_1, h_1 + \kappa\}$ and $\mathbb{B}_{q_j} = \{h_1 - (j-1)\kappa,\ h_1 + j\kappa\}$ for $j > 0$. From this it is easy to deduce that $q_j = \pi_{i+j+1} \circ \gamma_j$ for all j, (only that this time π_{i+j+1} and γ_j do not commute). Thus as per the analysis of case (A) we again have

$$\Pi = (\pi_{z+i} \circ \gamma_{z-1}) \circ (\pi_{z+i-1} \circ \gamma_{z-2}) \circ \cdots \circ (\pi_{i+2} \circ \gamma_1) \circ (\pi_{i+1} \circ \gamma_0)$$

where Π is such that $\overrightarrow{\mathsf{Sel}}_\Pi = \overrightarrow{\mathsf{Sel}}_1 \hat{}\ \overrightarrow{\mathsf{Sel}}_2$. In spite of the fact that π_{i+j+1} and γ_j do not commute, we intend to prove that

$$(\pi_{i+j+1} \circ \gamma_j) \circ \cdots \circ (\pi_{i+2} \circ \gamma_1) \circ (\pi_{i+1} \circ \gamma_0) = \pi[i+j+1 \to i+1] \circ \gamma[j \to 0],\ \forall\ j$$

which would prove equation (16) for this case too. We proceed by mathematical induction: for $j = 1$, from equation (19), we have

$$\pi[i+2 \to i+1] \circ \gamma[1 \to 0] = (\overline{h}_1 - 2\kappa,\ \overline{h}_1,\ \overline{h}_1 - \kappa, \overline{h}_1 + \kappa, \ldots, \overline{g}_1)$$

Also $(\pi_{i+2} \circ \gamma_1) \circ (\pi_{i+1} \circ \gamma_0)$ can be calculated from equation (18) as:

$$\begin{aligned} &(\overline{h}_1 - \kappa,\ \overline{h}_1,\ \overline{h}_1 + \kappa) \circ (\overline{h}_1 - 2\kappa,\ \overline{h}_1 - \kappa,\ \overline{h}_1,\ \overline{h}_1 + \kappa,\ \ldots,\ \overline{g}_1) \\ &= (\overline{h}_1 - 2\kappa,\ \overline{h}_1,\ \overline{h}_1 - \kappa, \overline{h}_1 + \kappa, \ldots, \overline{g}_1) = \pi[i+2 \to i+1] \circ \gamma[1 \to 0]. \end{aligned}$$

We will now prove instance $j+1$ assuming all instances from $1 \to j$ are correct. From equations (18), (19) we can calculate $(\pi_{i+j+2} \circ \gamma_{j+1}) \circ \pi[i+j+1 \to i+1] \circ \gamma[j \to 0]$ as follows:

$$\begin{aligned} &(\overline{h}_1 + (j-1)\kappa,\ \overline{h}_1 + j\kappa,\ \overline{h}_1 + (j+1)\kappa) \ \circ\ (\overline{h}_1 - 2\kappa,\ \overline{h}_1 + (j-1)\kappa, \ldots, \overline{h}_1, \\ &\quad \overline{h}_1 - \kappa, \overline{h}_1 + j\kappa, \ldots, \overline{g}_1) \\ &= (\overline{h}_1 - 2\kappa, \overline{h}_1 + j\kappa,\ \overline{h}_1 + (j-1)\kappa, \ldots, \overline{h}_1, \overline{h}_1 - \kappa, \overline{h}_1 + (j+1)\kappa, \ldots, \overline{g}_1) \\ &= \pi[i+j+2 \to i+1] \circ \gamma[j+1 \to 0]. \end{aligned}$$

This concludes proof for case (B).

□

Example 2. An immediate application of the above is to construct a $\overrightarrow{\mathsf{Sel}}$ vector to execute $(51,15)$ and $(48,12)$ concurrently on the $\mathsf{PRESENT}$ circuit. In the previous subsection we had executed them sequentially which had cost us $12 \cdot 64 = 768$ cycles each. Start with $\sigma_1 = (48,12)$. We have $\mathbb{B}_{\pi_0} = \{15,18,21,24,27,30,33,36,39,42,45,48\}$, $\mathbb{B}_{\pi_j} = \{48-3j\}$ for $1 \le j \le 11$. For $\sigma_2 = (51,15)$, we observe that $(\pi_0(51), \pi_0(15)) = (51,18)$. If we let $p = (51,18)$, then $\mathbb{B}_{\gamma_0} = \{12,15,18,21,24,27,30,33,36,39,42\}$. Since $\mathbb{B}_{\gamma_0} \cap \mathbb{B}_{\pi_1} = \varnothing$, this choice of p will work. Also $\overrightarrow{\mathsf{Sel}}_1$ and $\overrightarrow{\mathsf{Sel}}_2$ will be of the same length, due to which zero padding is also not required. So we have $\mathbb{B}_{\gamma_j} = \{42-3j\}$ for all $1 \le j \le 10$. Since $\overrightarrow{\mathsf{Sel}}_{\sigma_1 \circ \sigma_2} = \overrightarrow{\mathsf{Sel}}_1 \hat{\ } \overrightarrow{\mathsf{Sel}}_2 \ || \ \overrightarrow{\mathsf{Sel}}_{\pi[i \to 0]}$, denoting the decomposition of $\sigma_1 \circ \sigma_2$ by the symbols η_j, we have $\mathbb{B}_{\eta_0} = \mathbb{B}_{\pi_0}$, $\mathbb{B}_{\eta_{j+1}} = \mathbb{B}_{\pi_{j+1}} \cup \mathbb{B}_{\gamma_j}$ for $0 \le j \le 10$. This therefore implements $\sigma_1 \circ \sigma_2$ in only $12 \cdot 64 = 768$ cycles.

In the next lemma, we take things forward. If σ is a permutation which implements a set of disjoint transpositions (instead of just a single transposition), it may be possible to implement another transposition σ' concurrently along with σ if certain conditions are met.

Lemma 10. *Let σ be a special permutation of type κ that is a composition of several pairwise disjoint transpositions. Let $\sigma' = (x,y)$ be a special transposition $(x > y)$ of type κ, that is also pairwise disjoint with each of the transpositions that compose σ. Let the respective decompositions are denoted by the symbols π_i and θ_i, i.e. $\sigma = \pi_{z_1-1} \circ \pi_{z_1-2} \circ \cdots \circ \pi_2 \circ \pi_1 \circ \pi_0$ and $\sigma' = \theta_{z_2-1} \circ \theta_{z_2-2} \circ \cdots \circ \theta_2 \circ \theta_1 \circ \theta_0$. Let us have $\mathbb{A}_{\pi_0} \cap \mathbb{A}_{\theta_0} \neq \varnothing$. Denote by $p = (\pi[i \to 0](x), \pi[i \to 0](y))$, for some $i \in [0, z_1-1]$. Let the decomposition of p be denoted as*

$$p = \gamma_{q-1} \circ \gamma_{q-2} \circ \cdots \circ \gamma_1 \circ \gamma_0.$$

Now denote $\overrightarrow{\mathsf{Sel}}_1 = \overrightarrow{\mathsf{Sel}}_{\pi_{z_1-1}} || \overrightarrow{\mathsf{Sel}}_{\pi_{z_1-2}} || \cdots || \overrightarrow{\mathsf{Sel}}_{\pi_{i+1}}$ and $\overrightarrow{\mathsf{Sel}}_2 = \overrightarrow{\mathsf{Sel}}_p$. After appending with zeroes to make $\overrightarrow{\mathsf{Sel}}_1$ and $\overrightarrow{\mathsf{Sel}}_2$ of the same length, the following vector

$$\overrightarrow{\mathsf{Sel}}_1 \hat{\ } \overrightarrow{\mathsf{Sel}}_2 \ || \ \overrightarrow{\mathsf{Sel}}_{\pi[i \to 0]}$$

will execute $\sigma \circ \sigma'$ on the circuit in Figure 2, if $\mathbb{B}_{\gamma_0} \cap \mathbb{B}_{\pi_{i+1}} = \varnothing$.

Proof. We give a sketch of the proof as a complete analytical proof is likely to be quite complicated. The idea is similar to the ideas explained in the proof of Lemma 9. Note that $\mathbb{B}_{\pi_{i+j+1}}$ $(j \ge 0)$ will be the union of the corresponding $\mathbb{B}$ sets of the several transpositions that compose σ. One has to iterate the "disjoincy" arguments introduced in Lemma 9, for $\mathbb{B}_{\gamma_0}$ and each of those $\mathbb{B}$ sets to arrive at a proof.

Example 3. Let us construct a $\overrightarrow{\mathsf{Sel}}$ vector for all the s_i's in $\mathsf{PRESENT}$ that are 0 mod 3. The transpositions are $(51,15)$, $(48,12)$, $(54,45)$, $(39,30)$, $(18,9)$, $(33,24)$. We already have a $\overrightarrow{\mathsf{Sel}}$ vector for $(51,15) \circ (48,12)$ in the previous example.

1. To start, we have $\sigma = (51,15) \circ (48,12)$, $\mathbb{B}_{\pi_0} = \{15,18,21,24,27,30,33,36,39,42,45,48\}$, $\mathbb{B}_{\pi_1} = \{12,15,18,21,24,27,30,33,36,39,42,45\}$, and $\mathbb{B}_{\pi_j} = \{48-3j, 45-3j\}$ for all $2 \le j \le 11$. Let $\sigma' = (54,45)$. Now $(\pi[1 \to 0](54), \pi[1 \to 0](45)) = (54,51)$. If $p = (54,51)$ then $\mathbb{B}_{\gamma_0} = \{9\}$. which is disjoint with $\mathbb{B}_{\pi_2} = \{42,39\}$. Since $\mathbb{B}_{\gamma_0}$ has only one element it is sufficient to generate p. So we have $\mathbb{B}_{\eta_2} = \mathbb{B}_{\pi_2} \cup \mathbb{B}_{\gamma_0} = \{42,39,9\}$. For all other j, we have $\mathbb{B}_{\eta_j} = \mathbb{B}_{\pi_j}$. This will give us $\sigma \circ \sigma'$.

2. This time $\sigma = (51,15) \circ (48,12) \circ (54,45)$. Shifting notations, we have $\mathbb{B}_{\pi_0} = \{15,18,21,24,27,30,33,36,39,42,45,48\}$, $\mathbb{B}_{\pi_1} = \{12,15,18,21,24,27,30,33,36,39,42,45\}$, $\mathbb{B}_{\pi_2} = \{42,39,9\}$, and $\mathbb{B}_{\pi_j} = \{48-3j, 45-3j\}$ for all $3 \le j \le 11$. Let $\sigma' = (33,24)$.

Now $(\pi[1 \to 0](33), \pi[1 \to 0](24)) = (39, 30)$. If $p = (39, 30)$ then $\mathbb{B}_{\gamma_0} = \{24, 27, 30\}$. which is disjoint with $\mathbb{B}_{\pi_2} = \{42, 39, 9\}$. We have $\mathbb{B}_{\gamma_1} = \{27\}$ and $\mathbb{B}_{\gamma_2} = \{24\}$. So we have $\mathbb{B}_{\eta_2} = \mathbb{B}_{\pi_2} \cup \mathbb{B}_{\gamma_0} = \{42, 39, 30, 27, 24, 9\}$, $\mathbb{B}_{\eta_3} = \mathbb{B}_{\pi_3} \cup \mathbb{B}_{\gamma_1} = \{39, 36, 27\}$, $\mathbb{B}_{\eta_4} = \mathbb{B}_{\pi_4} \cup \mathbb{B}_{\gamma_2} = \{36, 33, 24\}$. For all other j, we have $\mathbb{B}_{\eta_j} = \mathbb{B}_{\pi_j}$.

3. Now $\sigma = (51, 15) \circ (48, 12) \circ (54, 45) \circ (33, 24)$. Since we are using shifting notations, all $\mathbb{B}_{\pi_j}$'s are the corresponding $\mathbb{B}_{\eta_j}$'s of the previous iteration. Let $\sigma' = (39, 30)$. Now $(\pi[2 \to 0](39), \pi[2 \to 0](30)) = (45, 39)$. If $p = (45, 39)$ then $\mathbb{B}_{\gamma_0} = \{18, 21\}$. which is disjoint with $\mathbb{B}_{\pi_3} = \{39, 36, 27\}$. We have $\mathbb{B}_{\gamma_1} = \{18\}$. So we have $\mathbb{B}_{\eta_3} = \mathbb{B}_{\pi_3} \cup \mathbb{B}_{\gamma_0} = \{39, 36, 27, 21, 18\}$ and $\mathbb{B}_{\eta_4} = \mathbb{B}_{\pi_4} \cup \mathbb{B}_{\gamma_1} = \{36, 33, 24, 18\}$. For all other j, we have $\mathbb{B}_{\eta_j} = \mathbb{B}_{\pi_j}$.

4. Now $\sigma = (51, 15) \circ (48, 12) \circ (54, 45) \circ (33, 24) \circ (39, 30)$. All $\mathbb{B}_{\pi_j}$'s are the corresponding $\mathbb{B}_{\eta_j}$'s of the previous iteration. Let $\sigma' = (18, 9)$. Now $(\pi[3 \to 0](18), \pi[3 \to 0](9)) = (18, 9)$. If $p = (18, 9)$ then $\mathbb{B}_{\gamma_0} = \{45, 48, 51\}$. which is disjoint with $\mathbb{B}_{\pi_4} = \{36, 33, 24, 18\}$. We have $\mathbb{B}_{\gamma_1} = \{48\}$ and $\mathbb{B}_{\gamma_2} = \{45\}$. So we have $\mathbb{B}_{\eta_4} = \mathbb{B}_{\pi_4} \cup \mathbb{B}_{\gamma_0} = \{51, 48, 45, 36, 33, 24, 18\}$, $\mathbb{B}_{\eta_5} = \mathbb{B}_{\pi_5} \cup \mathbb{B}_{\gamma_1} = \{48, 33, 30\}$ and $\mathbb{B}_{\eta_6} = \mathbb{B}_{\pi_6} \cup \mathbb{B}_{\gamma_2} = \{45, 30, 27\}$. For all other j, we have $\mathbb{B}_{\eta_j} = \mathbb{B}_{\pi_j}$. This completes the construction for all the s_i's of the form $0 \bmod 3$. Let us enumerate the sets explicitly

$$\begin{aligned}
\mathbb{B}_{\eta_0} &= \{15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48\} \\
\mathbb{B}_{\eta_1} &= \{12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45\} \\
\mathbb{B}_{\eta_2} &= \{42, 39, 30, 27, 24, 9\} \\
\mathbb{B}_{\eta_3} &= \{39, 36, 27, 21, 18\} \\
\mathbb{B}_{\eta_4} &= \{51, 48, 45, 36, 33, 24, 18\} \\
\mathbb{B}_{\eta_5} &= \{48, 33, 30\} \\
\mathbb{B}_{\eta_6} &= \{45, 30, 27\} \\
\mathbb{B}_{\eta_j} &= \{48 - 3j, 45 - 3j\}, \ \forall\ 7 \le j \le 11
\end{aligned}$$

This therefore constructs all the s_i's of the PRESENT permutation that are $0 \bmod 3$ in $12 \cdot 64 = 768$ cycles. We can use a similar approach to construct $\overrightarrow{\mathsf{Sel}}$ vectors for the s_i's that are $1, 2 \bmod 3$, that can operate in $8 \cdot 64 = 512$ cycles each. After padding of each of them to 768 cycles, a simple application of lemma 8 and corollary 3, allows us to construct the $\overrightarrow{\mathsf{Sel}}$ vector for the composition of all the s_i's by computing $\overrightarrow{\mathsf{Sel}}_{0 \bmod 3} \hat{}\ \overrightarrow{\mathsf{Sel}}_{1 \bmod 3} \hat{}\ \overrightarrow{\mathsf{Sel}}_{2 \bmod 3}$ that operates in just 768 cycles. Table 5 lists the $\mathbb{B}$ vectors for all the s_i transpositions that are $1, 2 \bmod 3$. Using a similar approach one can construct a similar $\overrightarrow{\mathsf{Sel}}$ vector for the composition of all t_i's that operate in $11 \cdot 64 = 704$ cycles. The $\mathbb{B}$ vectors for the s_i transpositions are also listed in Table 5. Since our strategy is to execute the composition of the t_i's followed by the s_i's this approach takes $704 + 768 = \mathbf{1472}$ cycles, which is the best we could manage.

4 The PRESENT circuit

Using the mathematical background presented in the previous section, we present our construction of the PRESENT circuit. Note that the sequence of operation in PRESENT are as follows:

Group	mod3	j	$\mathbb{B}_{\eta_j}$	#Cycles
s_i	1	0	$\{29, 32, 35, 38, 41, 44, 47, 50\}$	512
		1	$\{8, 11, 14, 17, 20, 23, 26, 47\}$	
		2	$\{5, 8, 11, 23, 44\}$	
		3	$\{8, 20, 26, 29, 32, 41\}$	
		4	$\{5, 17, 29, 38, 47, 50, 53, 56\}$	
		5	$\{14, 26, 35, 44, 53\}$	
		6	$\{11, 32, 50\}$	
		7	$\{8, 29, 47\}$	
	2	0	$\{31, 34, 37, 40, 43, 46, 49, 52\}$	512
		1	$\{10, 13, 16, 19, 22, 25, 28, 49\}$	
		2	$\{25, 28, 31, 34, 37, 40, 43, 46\}$	
		3	$\{4, 7, 10, 22, 40, 43\}$	
		4	$\{7, 19, 25, 28, 37, 40, 46, 49, 52, 55\}$	
		5	$\{4, 10, 13, 16, 25, 34, 37, 52\}$	
		6	$\{10, 13, 31, 34, 49\}$	
		7	$\{10, 28, 31, 46\}$	
t_i	0	0	$\{18, 21, 24, 27, 30, 33, 39, 42, 45, 48, 51, 54\}$	576
		1	$\{6, 9, 12, 15, 18, 30, 51\}$	
		2	$\{15, 27, 48\}$	
		3	$\{12, 24, 45\}$	
		4	$\{9, 21, 27, 30, 33, 36, 39, 42\}$	
		5	$\{6, 18, 36, 39\}$	
		6	$\{3, 6, 9, 33, 51, 54, 57\}$	
		7	$\{6, 30, 54\}$	
		8	$\{3, 27, 51\}$	
	1	0	$\{11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41\}$	704
		1	$\{38\}$	
		2	$\{35\}$	
		3	$\{32\}$	
		4	$\{29, 35, 38, 41, 44, 47, 50, 53\}$	
		5	$\{2, 5, 20, 23, 26, 50\}$	
		6	$\{2, 14, 20, 23, 35, 59\}$	
		7	$\{20, 44\}$	
		8	$\{17, 41\}$	
		9	$\{14, 38\}$	
		10	$\{11, 35\}$	
	2	0	$\{19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49\}$	704
		1	$\{7, 10, 13, 16, 19, 22, 46\}$	
		2	$\{19, 31, 34, 43, 55, 58\}$	
		3	$\{1, 16, 31, 40, 55\}$	
		4	$\{13, 19, 37, 43\}$	
		5	$\{10, 34\}$	
		6	$\{7, 31\}$	
		7	$\{28\}$	
		8	$\{25\}$	
		9	$\{22\}$	
		10	$\{19\}$	

Table 5: Constructed $\mathbb{B}$ sets for the t_i's and s_i's in the PRESENT permutation

PRESENT **Datapath**

```
1. For $i = 1 \rightarrow 31$ do
       addRoundkey(STATE,$K_i$)
       sBoxLayer(STATE)
       pLayer(STATE)
2. addRoundkey(STATE,$K_{32}$)
```

PRESENT **Keypath**

```
1. For $i = 1 \rightarrow 32$ do
       $K_i = [k_{79}, k_{78}, \ldots, k_{16}]$
       $[k_{79}, k_{78}, \ldots, k_1, k_0] \leftarrow [k_{18}, k_{17}, \ldots, k_{20}, k_{19}]$
       $[k_{79}, k_{78}, k_{77}, k_{76}] \leftarrow S[k_{79}, k_{78}, k_{77}, k_{76}]$
       $[k_{19}, k_{18}, k_{17}, k_{16}, k_{15}] \leftarrow [k_{19}, k_{18}, k_{17}, k_{16}, k_{15}] \oplus i$
```

The circuit for the datapath and the keypath are described in Figures 4 and 5 respec-

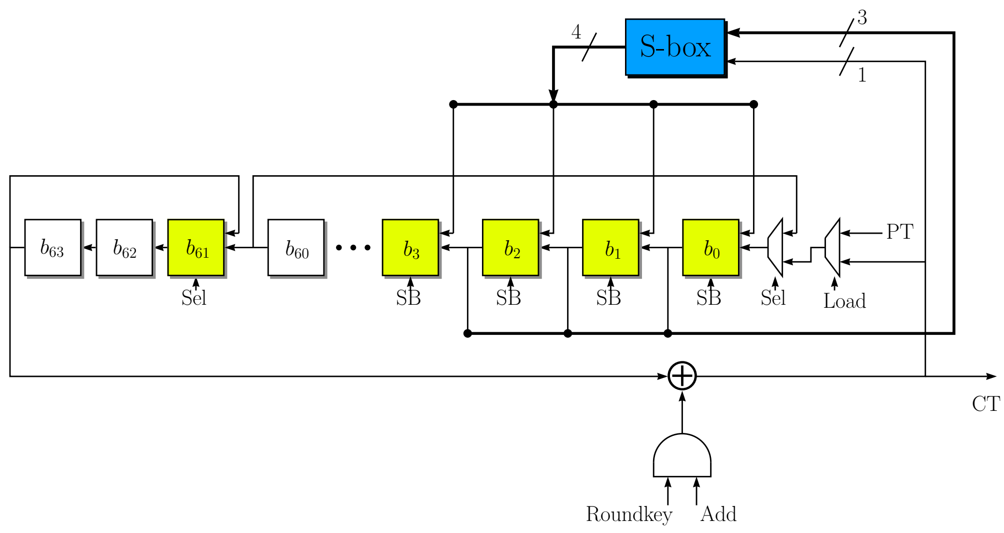


Figure 4: The PRESENT datapath

tively. Note that although it appears that the circuit employs two s-boxes, one for the data and keypaths each, in fact the there is only one s-box circuit with a multiplexer in front which accepts inputs from the state and key registers at different periods in the encryption cycle. In order to explain the circuit operations, it is most instructive to give a cycle by cycle explanation of the flow of data in the registers.

First 80 cycles: In this period the plaintext and key are loaded onto the state and key registers bit by bit. We initiate a register **Cycle** which is reset to zero at the end of the key and plaintext loading.

Cycle 0 to 63: This period is used for adding the roundkey to the state bits and then a subsequent S-box operation. Although key addition and the subsequent register updates are done bitwise, it is possible to execute the 4-bit S-box operation by using the idea introduced in [JMPS17]. In Figure 4, we can see that the last 4 flip-flops in the circuit are in fact scan flip-flops which will help in the S-box operation. In the first 3 cycles of every 4-cycle period, the SB signal that controls these flip-flops are kept at zero so that in these 3 cycles the updated value is the addition of the corresponding state and keybits without the S-box operation. In the 4th cycle of this 4-cycle period, the SB signal is changed to 1 so that 4 bit output of the S-box is updated en-masse in this cycle.

Cycle 64 to 1535: The next 1472 cycles are used to implement the permutation layer as explained in the previous sub-section. The Sel port that controls the 61^{st} flip-flop is fed the signals from the $\overrightarrow{\mathsf{Sel}}$ vector constructed in the previous section. The **Cycle** register is reset to zero at the end of this period.

The above procedure is repeated 31 times. In the 32^{nd} iteration the first 64 cycles are used for the final roundkey addition operation and the ciphertext is available at the output of the xor gate that does the key addition. The keypath operations are slightly more involved. We need to perform the key update operations correctly, and at the same time ensure that the correct roundkey bit is available during the roundkey addition operation. The key update operation rotates the 80-bit key towards the left by 61 bits, then applies the s-box to a fixed nibble and then adds the round-constant to another fixed 5 bit chunk. The main concern therefore is to ensure that after the completion of a round, which in this case consists of

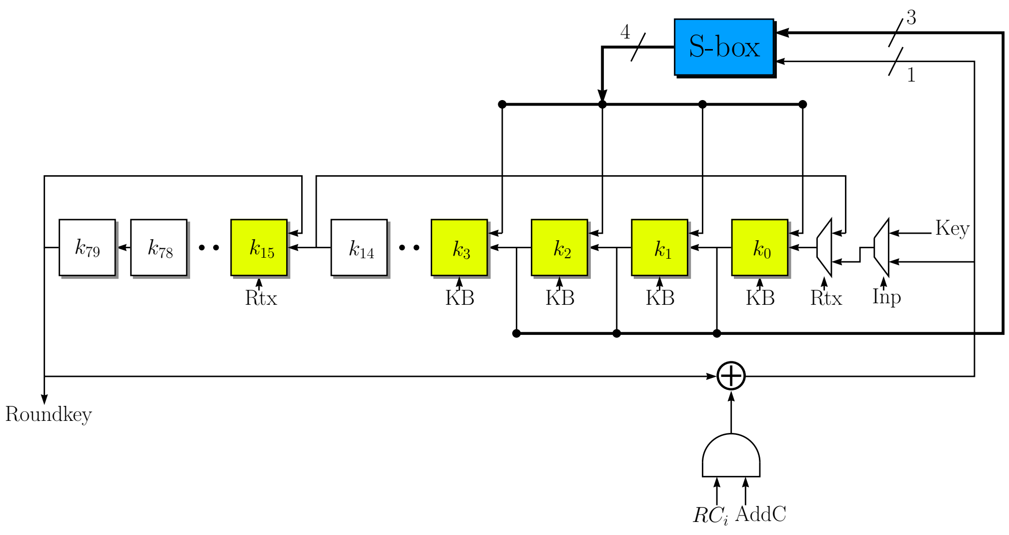
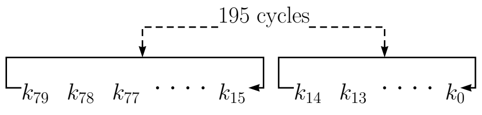


Figure 5: The PRESENT Keypath

1536 cycles, the key is rotated by exactly 61 bits. It may have been possible to achieve this using a gated clock in the key registers that freezes the update operations for certain period of time. But clock gating requires some logic of its own and our intention was to see if we could achieve the required functionality without resorting to gating. Note that if we were to let the key register rotate uninterrupted for 1536 cycles, we would achieve a left key rotation of 1536 mod 80 = 16 bits. However if we rotate the register for β cycles such that $\beta \equiv 61 \bmod 80$ and somehow freeze the rotation for the remaining $1536 - \beta$ cycles, we would achieve the required functionality. We chose $\beta = 1341$, which would require freezing the rotation operation for 195 cycles. To achieve this we use a scan flip-flop in the 15th location, controlled by a Rtx signal. When the Rtx signal is 1, the key register performs internal rotation between the first 65 and the next 15 bit chunks as shown in the following figure.



Since 195 is a multiple of both 15 and 65, such internal rotation when performed for 195 clock cycles, results in the identity function, and so we achieve our end objective of arresting rotation for exactly 195 cycles. For this purpose, one can choose any 195 of the 1536 cycles used in every round, except of course for the first 64 when the key addition is being performed. There are additional control signals KB that like SB in the case of of the state path controls the S-box operation in the key registers. And the AddC signal controls addition with round constants. These signals are set to 1 at appropriate cycles to ensure the respective functionalities.

4.1 Area and throughput results

The number of cycles taken to compute the encryption is therefore $80+1536\cdot 31+64 = 47760$. This is around 24 times slower than the implementation in [JMPS17]. Assuming that the $\overrightarrow{\mathsf{Sel}}$ vector is stored as look-up-table within the circuit, then after synthesizing the circuit using the STM 90nm standard cell library, the synthesized circuit occupies logic area of around 915 GE which is also 65 GE more than the implementation in [JMPS17]. Let us call this circuit: configuration **A**. However it may be possible have a configuration **B** in which the $\overrightarrow{\mathsf{Sel}}$ vector in stored in some local memory of the processor communicating with the circuit. The processor could then input the this signal bit-by-bit via an extra input Sel port of the circuit. Around 200 B of storage is required which could be accommodated in a EEPROM or any similar memory. In this event, the encryption circuit would occupy only **727** GE which is the smallest area reported for this block cipher in literature. The results are summarized in Table 9.

4.2 Circuit for combined encryption and decryption

The approach outlined in the previous section is surprisingly effective when we try to implement a PRESENT circuit that can offer the combined functionalities of encryption and decryption. As already pointed out in [BBR16, BBR17a, JMPS17], such circuits are useful in implementing modes of operation like ELmD, CBC that require access to both the block cipher and its inverse operation. Our strategy to execute the PRESENT permutation P was to first execute the transpositions t_i in any order and then the transpositions s_i again in any order. In order to execute the inverse permutation P^{-1} it is enough to reverse the order: first execute the s_i's followed by the t_i's as explained below.

In order to understand why this is so, let us denote the composition of all s_i's as $\mathcal{S}$ and the composition of all t_i's as $\mathcal{T}$, so that we have $P = \mathcal{S} \circ \mathcal{T}$. Both $\mathcal{S}$ and $\mathcal{T}$ are compositions of disjoint transpositions. Transpositions are involutary functions, which is to say they are self-inverses. Since all the t_i's in $\mathcal{T}$ are disjoint, $\mathcal{T}^{-1}$ is again the composition of all the t_i's therefore equal to $\mathcal{T}$. The same is true for $\mathcal{S}$. So we have $P^{-1} = \mathcal{T}^{-1} \circ \mathcal{S}^{-1} = \mathcal{T} \circ \mathcal{S}$. Thus the inverse permutation can be executed on the same circuit by shuffling around the $\overrightarrow{\mathsf{Sel}}$ vector. There are however a few additions to the combined circuit that are listed below:

- There is an additional circuit for the PRESENT inverse S-box.
- The order of operations in the decryption process is listed as follows:

```
PRESENT Datapath
1. addRoundkey(STATE,$K_{32}$)
2. Inv-pLayer(STATE)
3. For $i = 31 \to 2$ do
     Inv-sBoxLayer(STATE)
     addRoundkey(STATE,$K_i$)
     Inv-pLayer(STATE)
4. Inv-sBoxLayer(STATE)
5. addRoundkey(STATE,$K_1$)
```

```
PRESENT Keypath
1. $K_{32} = [k_{79}, k_{78}, \dots, k_{16}]$
2. For $i = 31 \to 1$ do
     $[k_{19}, k_{18}, k_{17}, k_{16}, k_{15}] \leftarrow [k_{19}, k_{18}, k_{17}, k_{16}, k_{15}] \oplus i$
     $[k_{79}, k_{78}, k_{77}, k_{76}] \leftarrow S^{-1}[k_{79}, k_{78}, k_{77}, k_{76}]$
     $[k_{79}, k_{78}, \dots, k_1, k_0] \leftarrow [k_{60}, k_{59}, \dots, k_{62}, k_{61}]$
```

The sequence of operations during decryption is slightly different. So let us look at the sequence of operations in each cycle:

First 80 cycles: As usual the ciphertext and key are loaded onto the respective registers.

Cycle 0 to 63: In the round immediately after ciphertext loading, we perform only bitwise round key addition in this period. However in all the subsequent rounds, we need to do an Inverse s-box operation before roundkey addition. This would require some incremental additions to the circuit. First of all we need a 4-bit xor to do the key addition instead of just a single bit xor in the encryption path. A four bit multiplexer is additionally required to select between the 4-bit updates during encryption and decryption. The logic circuit is explained diagrammatically in figure 6.

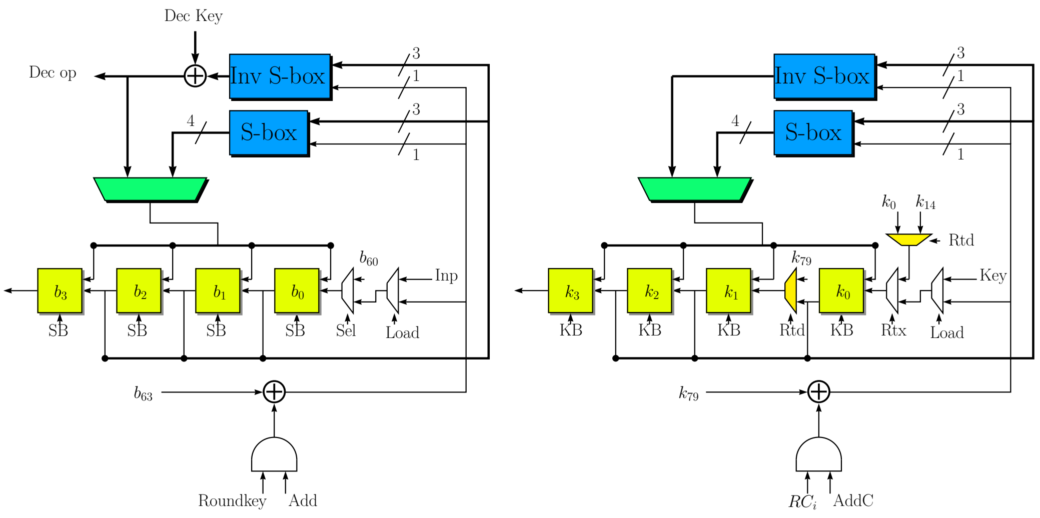


Figure 6: Modified logic around the last 4 flip-flops to accommodate decryption

Cycle 64 to 1535: The next 1472 cycles are used to implement the inverse permutation layer, with the $\mathcal{S}$ executed ahead of $\mathcal{T}$.

- The keyschedule involves addition by round constant followed by application of inverse s-box on a fixed nibble followed by rotation by 19 bits to the left. As before we need to try to rotate the key register for $\beta \equiv 19 \bmod 80$ cycles and somehow arrest the rotation for the remaining $1536 - \beta$ cycles. We choose $\beta = 1399$. which requires stopping the rotation for 237 cycles. Again, we try to achieve this by breaking up the key into chunks of 79 and 1 bits and doing internal rotation within the key-chunks for 237 cycles. Since 237 is a multiple of 79 and 1, internal rotation for 237 cycles again gives the identity transformation which satisfies our end objective. In terms of hardware, this requires two extra multiplexers to do the internal rotation as shown in Figure 6.

In configuration **A**, the area occupied by the circuit when synthesized with the standard cell library of the STM 90nm CMOS process, is around 1040 GE. This is around 200 GE less than the previous best reported implementation of the combined circuit for PRESENT in [BBR17b], which occupies around 1240 GE. In configuration **B**, the circuit occupies only around 809 GE.

5 Application to GIFT

GIFT was a block cipher designed by Banik et al. [BPP+17] and presented at CHES 2017, with a view to strengthen the cryptographic properties of PRESENT by redesigning the permutation layer and keyschedule. It is a block cipher with an SPN round function in which the linear layer is a bit permutation similar to PRESENT. The permutation function G is listed in table 6.

The following can be said about the function G:

1. It is a special permutation of type $\kappa = 4$.

2. It can be decomposed into fourteen 4-cycles and two 2-cycles all of which are pairwise disjoint. Additionally it has 4 fixed points.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$G(i)$	0	17	34	51	48	1	18	35	32	49	2	19	16	33	50	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$G(i)$	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$G(i)$	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$G(i)$	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

Table 6: Specifications of GIFT bit-permutation layer.

3. Each 4-cycle can be decomposed into three transpositions $s_i \circ t_i \circ u_i$. The decomposition is shown in the following table.

i	c_i	$s_i \circ t_i \circ u_i$	i	c_i	$s_i \circ t_i \circ u_i$
0	$(1,17,21,5)$	$(5,17) \circ (17,21) \circ (1,5)$	8	$(11,19,55,47)$	$(19,47) \circ (19,55) \circ (11,47)$
1	$(2,34,32,10)$	$(10,34) \circ (34,42) \circ (2,10)$	9	$(13,33,25,53)$	$(13,25) \circ (25,53) \circ (13,33)$
2	$(3,51,63,15)$	$(15,51) \circ (51,63) \circ (3,15)$	10	$(14,50,46,58)$	$(14,46) \circ (46,58) \circ (14,50)$
3	$(4,48,12,16)$	$(12,16) \circ (16,48) \circ (4,16)$	11	$(20,52,60,28)$	$(28,52) \circ (52,60) \circ (20,28)$
4	$(6,18,38,26)$	$(18,26) \circ (18,38) \circ (6,26)$	12	$(23,39,43,27)$	$(27,39) \circ (39,43) \circ (23,27)$
5	$(7,35,59,31)$	$(31,35) \circ (35,59) \circ (7,31)$	13	$(24,36,56,44)$	$(36,44) \circ (36,56) \circ (24,44)$
6	$(8,32)$	$(8,32) \circ \quad \circ$	14	$(30,54)$	$(30,54) \circ \quad \circ$
7	$(9,49,29,37)$	$(29,37) \circ (37,49) \circ (9,37)$	15	$(41,57,61,45)$	$(45,57) \circ (57,61) \circ (41,45)$

Table 7: Decomposition of the c_i's in the GIFT permutation

As per the strategies outlined in the case of PRESENT, we try to implement all the u_i's first, followed by the t_i's and s_i's, since as per lemma 3, we would have then constructed G. Furthermore for each of the u_i's, t_i's and s_i' we construct composite $\overrightarrow{\mathsf{Sel}}$ vectors by finding $\overrightarrow{\mathsf{Sel}}$ vectors for each equivalence class modulo 4 and then doing a bitwise OR. Each of the transposition sets can be implemented in $9 \times 64 = 576$ cycles. This implies that G can be executed in $3 \times 576 = 1728$ cycles. The results are tabulated in table 8.

5.1 Circuit details

Since the structure of GIFT is similar to PRESENT, the circuit for the datapath is exactly the same as in Figure 4, with the obvious exception that the scan flip-flop is used in the 60^{th} instead of the 61^{st} location. The sequence of operations in GIFT is only slightly different from PRESENT:

Group	mod4	j	$\mathbb{B}_{\eta_j}$	mod4	j	$\mathbb{B}_{\eta_j}$
u_i	0	0	$\{19, 23, 27, 31, 35\}$	1	0	$\{26, 30, 34, 38, 42, 46, 50\}$
		1	$\{31, 47, 51, 55\}$		1	$\{26, 30, 34, 38, 42, 46\}$
		2	$\{27, 35, 39, 51\}$		2	$\{18, 38, 42\}$
		3	$\{23, 35, 47\}$		3	$\{34, 38, 58\}$
		4	$\{19\}$		4	$\{30, 34\}$
					5	$\{26, 30\}$
					6	$\{26\}$
	2	0	$\{13, 17, 21, 25, 29, 33, 37, 41, 45\}$	3	0	$\{16, 20, 24, 28, 32, 36, 40, 44, 48\}$
		1	$\{41, 53, 57\}$		1	$\{28, 32, 36, 40, 44, 48, 52\}$
		2	$\{37, 53\}$		2	$\{40, 44, 48, 52, 56\}$
		3	$\{33\}$		3	$\{28, 36, 44, 52\}$
		4	$\{29, 37, 41, 45, 49, 53\}$		4	$\{32, 40, 48\}$
		5	$\{25, 49\}$		5	$\{28, 36\}$
		6	$\{21, 45\}$		6	$\{24, 32\}$
		7	$\{17, 41\}$		7	$\{20\}$
		8	$\{13, 37\}$		8	$\{16\}$
t_i	0	0	$\{15, 19, 23, 27, 31, 35, 39, 43\}$	1	0	$\{10, 14, 18, 22, 26, 30, 34\}$
		1	$\{7, 11, 15, 19, 39\}$		1	$\{10, 14, 18, 30\}$
		2	$\{3, 15, 35\}$		2	$\{14, 26, 42\}$
		3	$\{11, 31\}$		3	$\{2, 10, 22\}$
		4	$\{7, 27\}$		4	$\{18\}$
		5	$\{23\}$		5	$\{14\}$
		6	$\{19\}$		6	$\{10\}$
		7	$\{15\}$			
	2	0	$\{25, 29, 33, 37, 41\}$	3	0	$\{8, 12, 16, 20, 24, 28, 32, 36, 40\}$
		1	$\{5, 9, 13, 37\}$		1	$\{4, 8, 12, 16, 20, 36\}$
		2	$\{9, 21, 33\}$		2	$\{0, 16, 32\}$
		3	$\{5, 29\}$		3	$\{12, 16, 28\}$
		4	$\{25\}$		4	$\{8, 24\}$
					5	$\{4, 20\}$
					6	$\{16\}$
					7	$\{12\}$
					8	$\{8\}$
s_i	0	0	$\{31, 35, 39, 43, 47, 51\}$	1	0	$\{6, 10, 14, 38, 42, 46\}$
		1	$\{11, 15, 19, 23, 27, 57\}$		1	$\{10, 26, 30, 42\}$
		2	$\{15, 19, 23, 43\}$		2	$\{6, 26, 38, 46, 50, 54\}$
		3	$\{15, 19, 39, 47\}$		3	$\{50\}$
		4	$\{15, 35\}$		4	$\{54\}$
		5	$\{11, 31\}$			
	2	0	$\{17, 21, 25, 29, 33, 37, 41, 45\}$	3	0	$\{12, 16, 20, 24, 28, 32, 36, 40, 44\}$
		1	$\{9, 13, 17, 21, 25, 41\}$		1	$\{12, 16, 20, 24, 28, 32, 36, 40\}$
		2	$\{21, 25, 29, 33, 37, 41, 45, 49\}$		2	$\{16, 20, 24, 32, 36\}$
		3	$\{17, 29, 33, 37, 45\}$		3	$\{16, 20, 28, 32\}$
		4	$\{13, 29, 33, 41\}$		4	$\{16, 24, 28\}$
		5	$\{9, 25, 37\}$		5	$\{20, 24\}$
		6	$\{21, 33\}$		6	$\{16, 20\}$
		7	$\{17, 29\}$		7	$\{12, 16\}$
					8	$\{12\}$

Table 8: Constructed $\mathbb{B}$ sets for the u_i's, t_i's and s_i's in the GIFT permutation

```
GIFT Datapath

1. For $i = 1 \rightarrow 28$ do
       sBoxLayer(STATE)
       pLayer(STATE)
       addRoundkey(STATE,$RK_i$)
```

```
GIFT Keypath

1. For $i = 1 \rightarrow 28$ do
       $K_i = [k_{127}, k_{78}, \ldots, k_0]$
       For $j = 0 \rightarrow 7$: $L_j \leftarrow [k_{16j+15}, k_{16j+15}, \ldots, k_{16j}]$
       $RK_i = L_1 || L_0$
       $L_7 || L_6 || \cdots || L_0 \leftarrow L_1 \ggg 2 || L_0 \ggg 12 || L_7 || \cdots || L_2$
```

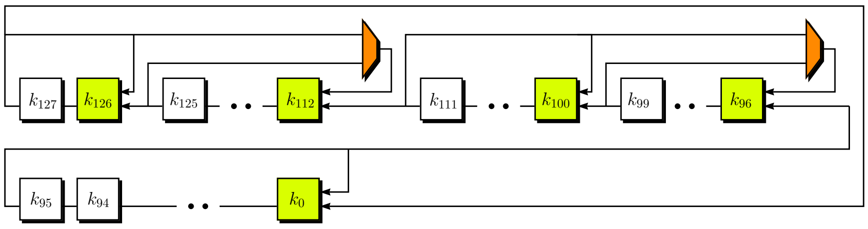


Figure 7: The GIFT key register

So the sequence of operations in the datapath is as follows:

First 128 cycles: In this period the key is loaded onto the state and key registers bit by bit. In cycles 64 to 127, the plaintext is loaded onto the state register after performing the s-box operation. Thereafter we have 28 iterations of the following operations.

Cycle 0 to 1727: Used to compute the permutation layer.

Cycle 1728 to 1791: The next 64 cycles are used to compute add roundkeys and then perform the s-box operation of the next round.

Thus the total number of cycles taken for the encryption routine is $128 + 28 \times 1792 = 50304$. The keyschedule is slightly more complicated: it breaks up the current key into eight 16 bit words L_7 to L_0. L_1 and L_0 are internally right rotated by 2, 12 bits respectively and the whole key is then right rotated by 32 bits. In other words this means internal left rotation of L_1 and L_0 by 14, 4 bits and overall left rotation by 96 bits. So we do the following

- Let the key register rotate left for 96 cycles. After this L_1 and L_0 occupy the the most significant 32 bits of the register(k_{127} to k_{96}).

- At this point of time we will partition the key register in to chunks of 16 (k_{127} to k_{112}), 16 (k_{111} to k_{96}) and 96 bits (k_{95} to k_0) and do an internal rotation for 288 cycles. Since $1792 - 288 = 1504 \equiv 96 \bmod 128$ this achieves our first objective of 96 bit left rotation.

- To achieve left rotation of L_1 by 14 bits, we partition the 1st 16 MSBs into chunks of 2 (k_{127} to k_{126}) and 14 (k_{125} to k_{112}) bits and do an internal rotation in these 2 groups for 98 cycles, and do a normal rotation over (k_{127} to k_{112}) over the remaining 288-98=190 cycles (see figure 7). Since 2 and 14 both divide 98, the rotation results in identity transformation. So the effective rotation is for $190 \equiv 14 \bmod 16$ cycles.

- Similarly rotating L_0 by 4 bits, we partition 2nd 16 MSBs into chunks of 12 (k_{111} to k_{100}) and 4 (k_{99} to k_{96}) bits. Internal rotation is carried out for in these smaller chunks 12 cycles. So effectively we rotate the 2nd chunk by $276 \equiv 4 \bmod 16$ bits.

However the key addition in GIFT is quite complicated: neighboring key bits do not xor with neighboring state bits as in PRESENT. In fact, the designers recommend that $\forall i \in [0, 31]$ the i^{th} bit of L_1 be xored with the $(4i+2)^{nd}$ state bit and the the i^{th} bit of L_0 be xored with the $(4i+1)^{st}$ state bit. Thus, the circuit also requires a filter to extract the correct roundkey bit in every cycle, which increases the total area slightly. In configuration **A**, the circuit occupies 1132 GE. In configuration **B**, the circuit occupies 925 GE, which is only around 5 GE smaller than the figure reported in [BPP+17] for the same standard cell library.

5.2 Combined Circuit for encryption and decryption

The GIFT decryption circuit suffers from the same issues as the corresponding PRESENT circuit, and therefore the circuit for the combined decryption is same as the one outlined in figure 6. The only differences are in the order in which the functions are carried out. The following is the sequence of operations:

GIFT **Datapath**

```
1. addRoundkey(STATE,$RK_{28}$)
2. inv-pLayer(STATE)
3. For $i = 27 \to 1$ do
     inv-sBoxLayer(STATE)
     addRoundkey(STATE,$RK_i$)
     inv-pLayer(STATE)
4. inv-sBoxLayer(STATE)
```

GIFT **Keypath**

```
1. For $i = 1 \to 28$ do
     $K_i = [k_{127}, k_{78}, \dots, k_0]$
     For $j = 0 \to 7$: $L_j \leftarrow [k_{16j+15}, k_{16j+15}, \dots, k_{16j}]$
     $RK_i = L_1 || L_0$
     $L_7 || L_6 || \cdots || L_0 \leftarrow L_5 || \cdots || L_0 || L_7 \lll 2 || L_6 \lll 12$
```

As expected, the inverse permutation layer is constructed by executing the s_i transpositions first, followed by the t_i's and then the u_i's. The cycle by cycle execution of operations is as follows:

First 128 cycles: In this period the key is loaded onto the state and key registers bit by bit. In cycles 64 to 127, the plaintext is loaded onto the state register without performing the inverse s-box operation. Thereafter the following operations are executed 28 times.

Cycle 0 to 63: Used for executing the inverse s-box operations followed by roundkey addition as shown in figure 6. As in PRESENT only in the first round, the inverse s-box operation is omitted.

Cycle 64 to 1791: Used for executing the inverse p-layer.

After this, the GIFT decryption process requires one more inverse s-box operation. Hence the decryption operation requires an additional 64 cycles to complete. The key schedule for decryption can be carried out using the same circuit as in figure 7. We need left rotation of L_7, L_6 by 2, 12 bits followed by a left rotation by 32 bits. At the beginning of the round cycle when L_7, L_6 still occupy the 32 msbs in the key register we do internal rotation for 96 cycles. It is easy to see to verify that this will achieve left rotation by 32 bits. In these 96 cycles, we do further internal rotation between the 2 and 14 bit chunks (k_{127} to k_{126} and k_{125} to k_{112}) for 14 cycles, and for 36 cycles between the 12 and 4 bits chunks (k_{111} to k_{100} and k_{99} to k_{96}) for 36 cycles. This is sufficient to achieve the required functionalities in the inverse keyschedule.

The circuit in configuration **A** occupies 1290 GE and the area is around 1050 GE in configuration **B**. This is the first reported synthesis results for the combined circuit for this block cipher.

6 Application to FLIP (how to do Knuth shuffles in constrained hardware)

FLIP is a family of stream ciphers proposed by Méaux et al. at Eurocrypt 2016 for FHE based applications. In [MJSC16] the authors suggested a stream cipher based solution to implement the above. The FLIP family stream ciphers have the lowest multiplicative depth compared with previous ciphers. Several versions are provided including 80-bit and 128-bit

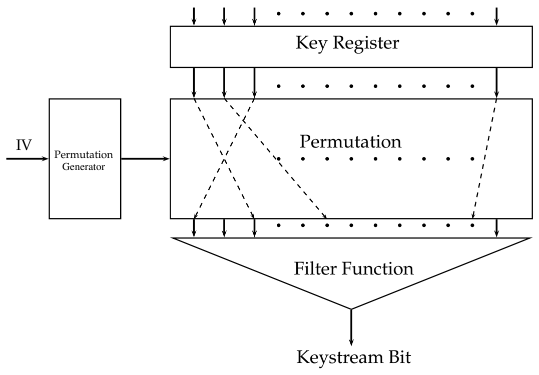


Figure 8: The FLIP stream cipher

security instantiations. The main design principle is to filter a constant key register with a time-varying public bit permutation. For 80 bit security the authors suggest the use of the instance FLIP$(42, 128,^8 \Delta^9)$ which uses a 530 bit secret key with hamming weight 265. The internal state $State_i$ of the cipher is a permutation of the original secret key sk. The cipher works as follows:

- Let $sk_S \in \{0,1\}^{530}$ with $HW(sk) = 265$.
- For $i = 1 \to n$ do
  1. Choose a random permutation P_i from the symmetric group S_{530}. (P_i may be a function of IV)
  2. Let $State_i = P_i(sk)$.
  3. Compute $z_i = F(State_i)$.

In the above definition, F is a $\{0,1\}^{530} \to \{0,1\}$ boolean function of multiplicative depth 4. It consists of a linear function of 42 variables, a quadratic bent function of 128 variables and the remaining 360 variables are used to construct 8 triangular functions of algebraic degree 9 each. For example a degree 3 triangular function is given as $x_1 + x_2x_3 + x_4x_5x_6$ (a degree n function thus has $n(n+1)/2$ variables).

Although the designers stop short of providing detailed design specifications, they do however mention that the permutations are generated by employing a combination of the IV and a PRP (possibly in the counter mode) to generate a sequence of pseudo-random bits, which are then used as random inputs to a Knuth shuffle module which generate the permutation. The question is therefore how to efficiently do a Knuth shuffle in a lightweight setting. In this let us make two observations

Observation 1: If P_1 and P_2 are random permutations over any symmetric group then $P_1 \circ P_2$ is also a random permutation. This means that we can modify the the sequence of operations in FLIP to the following:

- Let $sk \in \{0,1\}^{530}$ with $HW(sk) = 265$.
- $State_0 = sk$
- **for** $i = 1 \to n$ do

```
1: $P_i \xleftarrow{\$} S_{530}$.
2: Let $State_i = P_i(State_{i-1})$.
3: Compute $z_i = F(State_i)$.
```

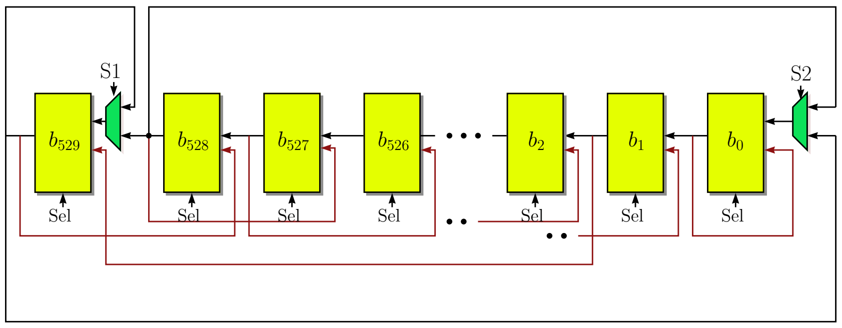


Figure 9: 2nd Circuit for Knuth Shuffle

This makes the i^{th} state $P_i \circ P_{i-1} \circ \cdots \circ P_1(sk)$ in place of $P_i(sk)$ but since P_i and $P_i \circ P_{i-1} \circ \cdots \circ P_1$ are both random permutations, this does not differ from the ideology of FLIP.

Observation 2: The above allows us to simply place the secret key in a register of equal length and do state updates by implementing the permutations P_i via some circuit. Since the authors recommend Knuth Shuffle, let us look at the algorithm. Let the state be denoted by the bits $b_{529}, b_{528}, \ldots, b_0$.

```
• for $i$ from 529 downto 1 do
    1: $j \leftarrow$ random integer such that $0 \le j \le i$.
    2: swap $b_j$ and $b_i$
```

Each of these swaps could be implemented with a circuit as shown in Fig 1.

6.1 First Attempt

The first idea is therefore to use the circuit in Fig 1 to implement a swap. As explained in Lemma 4, any swap of the form $b_i \leftrightarrow b_j$ $(j < i)$ is implemented by the sequence of functions:

$$[r^i \circ v \circ r^{529-i}] \circ [r^{i-1} \circ v \circ r^{530-i}] \circ \cdots \circ [r^{j+2} \circ v \circ r^{527-j}] \circ [r^{j+1} \circ v^{i-j} \circ r^{529-i}]$$

and by the identity function when $i = j$. This transpositions would take $530(i-j)$ cycles to complete and we could certainly use this circuit to implement one swap. The average value of $i - j$ is around $\frac{530}{4}$ and so implementing 529 swaps one after the other would take around $\frac{530^3}{4} \approx 2^{25}$ cycles which is a high price to pay for one keystream bit.

6.2 Second Attempt

We try to investigate if we can affect any speedup by increasing the circuit size. One of the reasons that a swap takes $530(i-j)$ cycles is that data can be transferred in only one direction. This is true because as per Lemma 4, the above sequence of transitions may also be written as

$$\begin{aligned} & r^{1+i} \circ w \circ (r^{529} \circ w)^{i-j-1} \circ (r \circ w)^{i-j-1} \circ r^{529-i} \\ = & (r^{-1})^{530-i} \circ (r^{-1} \circ w')^{i-j-1} \circ (r \circ w)^{i-j} \circ r^{529-i} \end{aligned}$$

In the above equation w' is a permutation that swaps the 1st and 0th bits i.e. $(1, 0)$. The above is not difficult to deduce once we use the fact $r^{-1} = r^{529}$ and $r^{-1} \circ w' = r \circ w \circ r^{-2}$.

Denote $u = r^{-1} \circ w'$, then the above is written as $(r^{-1})^{530-i} \circ u^{i-j-1} \circ v^{i-j} \circ r^{529-i}$. It can be seen that the circuit of Figure 9, it is possible to realize the functions r, u, v by appropriately adjusting the Sel, S1 and S2 signals. If we assume that in Figure 9, the signal at the top is filtered when the corresponding select signal is 0 and the one at the bottom when select is 1, it is easy to deduce that Sel, S1, S2 = (0,0,0) achieves v, Sel, S1, S2 = (0,1,1) achieves r and Sel, S1, S2 = (1,*,*) achieves u (* denotes any signal 0 or 1).

The algorithm for the Knuth shuffle basically consists of applying the following transpositions

$$\pi_K = (1, j_1) \circ (2, j_2) \circ (3, j_3) \circ \cdots \circ (528, j_{528}) \circ (529, j_{529}), \text{ with } (j_i \leq i, \ \forall\ i)$$

Denote $\Delta_i = i - j_i$, we have

$$(i, j_i) = \begin{cases} (r^{-1})^{530-i} \circ u^{\Delta_i - 1} \circ v^{\Delta_i} \circ r^{529-i}, & \text{if } \Delta_i > 0, \\ (r^{-1})^{529-i} \circ r^{529-i}, & \text{if } \Delta_i = 0. \end{cases}$$

This results in the following expression for π_K:

$$\pi_k = r \circ [u^{\Delta_1 - 1} \circ v^{\Delta_1}] \circ [u^{\Delta_2 - 1} \circ v^{\Delta_2}] \circ \cdots \circ \underset{\text{when } \Delta_i = 0}{[r]} \circ \cdots \circ [u^{\Delta_{529} - 1} \circ v^{\Delta_{529}}]$$

The above expression is solely in terms of r, u, v and thus can be executed on the circuit in figure 9. Unless $\Delta_i = 0$ which requires a single rotation, each of the expressions in the square braces takes $2\Delta_i - 1$ cycles. Since Δ_i has an average value of $\frac{530}{4}$, each shuffle takes around $530 * (2 * \frac{530}{4} - 1) \approx 2^{17}$ cycles. A synthesis of the above circuit using the standard cell library of the STM 90nm logic process, yielded a circuit of 3581 GE. The circuit is certainly an improvement on the previous circuit but still takes a lot of cycles to produce one keystream bit.

6.3 Third Attempt

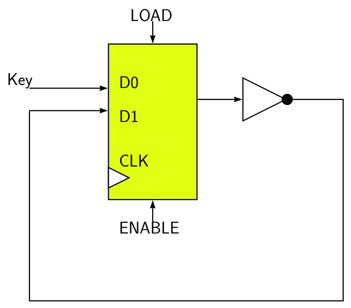
The previous circuits took time proportional to N^3 and N^2 clock cycles respectively to produce one keystream bit where N is the size of the key. In this part, we will try to construct a circuit in which the number of clock cycles taken to produce a keystream bit is at least linear in N. In order to achieve this, let us look at a few facts:

1: In order to achieve a shuffle in linear time, each individual swap has to be executed in constant time.

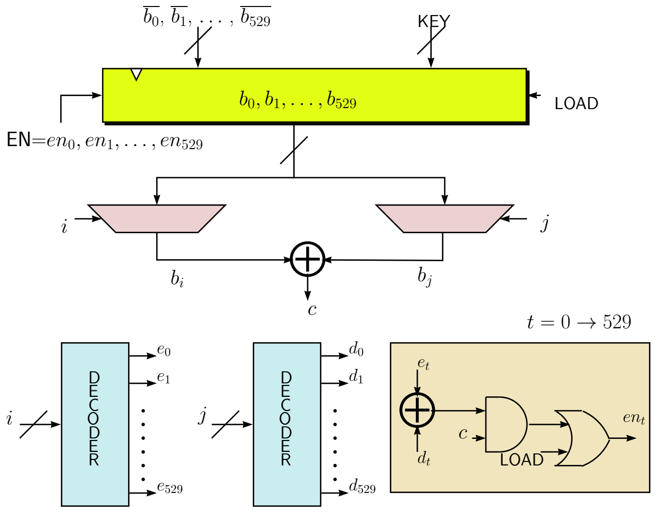
2: Observe that logically, a swap $b_i \leftrightarrow b_j$ needs to be executed only when b_i and b_j are of opposite parities. No swap operation is really necessary if b_i and b_j are logically equal. Furthermore, when b_i and b_j are logically unequal, a swap is essentially executed by toggling the polarities of both b_i and b_j, i.e. **swap**(b_i, b_j) is same as $b_i \leftarrow$ **not** b_i and $b_j \leftarrow$ **not** b_j.

3: In order to design this circuit, we note that a memory element must be able to accommodate **a)** the secret key during the initial loading cycle, **b)** hold the current value stored in the flip-flop for the next cycle, if no swap is required and finally **c)** toggle the current logic state if a swap is required at the particular location. In order to do this we use a scan flip-flop with an additional ENABLE pin that allows transitions at the positive clock edge only if it is HIGH, as shown in Figure 10a.

Thus we propose the circuit in Figure 10b. The circuit comprises of the following elements:



(a) Memory Element



(b) Circuit Details

Figure 10: 3rd Circuit for FLIP

Multiplexer: We employ two banks of multiplexers to filter out the bits b_i and b_j from the current state. We compute $c = b_i \oplus b_j$ to determine the difference in the logic values of b_i and b_j.

Decoder: We employ 2 decoder circuits that convert the 10 bit values i and j into a corresponding set of 530 bit signals e_t, d_t (for $t = 0 \rightarrow 529$), such that $e_t = 1$ iff $t = i$, and $d_t = 1$ iff $t = j$, and all signals are 0 otherwise. These signals are employed to feed the the ENABLE ports of the register bank. We argue that the logic value en_t driving the $t - th$ flip-flop is given as $[(e_t \oplus d_t) \cdot c]$ OR LOAD. The logic behind this is as follows. The signal LOAD is high only in the first cycle when in loads the key on to the register and is low thereafter. Thus en_t is forced to be high when LOAD is high. In the subsequent cycles en_t evaluates as $(e_t \oplus d_t) \cdot c$. If $c = 0$, i.e. b_i and b_j are of same parity then en_t evaluates to 0 which means that the flip-flop holds its previous value as no swap is required. If $c = 1$, then $en_t = e_t \oplus d_t$. Now $e_t = d_t = 0$ implies that no swap is scheduled at location t in that particular clock cycle and in this event en_t is 0. Now $e_t = d_t = 1$ occurs when $i = j$ in some iteration of the Knuth Shuffle. Here too, no swap operation is required and en_t evaluates to 0. When $e_t \neq d_t$, en_t evaluates to 1, and it is then that b_i and b_j are both toggled to effect a swap.

From the above description of the circuit elements and operational details, it is clear that each swap can be performed in one clock cycle, and so the shuffle takes exactly $529 + 1 = 530$ (1 extra cycle for key loading) clock cycles to execute a shuffle and hence produce one keystream bit. This circuit when synthesized with standard cell library of the STM 90nm logic process occupies around 8605 GE.

7 Results and Conclusion

In this paper we looked at a few circuit constructions aimed at achieving minimalism in block cipher and stream cipher circuits. The final results are presented in Table 9. More specifically, we tried to answer the question if bit-permutations like the one used

	Design	Config.	Area (GE)	Power (μW)	Latency	Ref
1	PRESENT (E)	**A**	935	40.0	1472 per round	Section 4
		B	727	35.8	1472 per round	Section 4
			847[2]	0.43[3]	68 per round	[JMPS17]
2	PRESENT (ED)	**A**	1039	41.4	1472 per round	Section 4
		B	809	37.7	1472 per round	Section 4
			1238	56.0	17 per round	[BBR17b]
3	GIFT (E)	**A**	1132	49.8	1728 per round	Section 5
		B	925	45.8	1728 per round	Section 5
			930	35.9	96 per round	[BPP$^+$17]
4	GIFT (ED)	**A**	1290	52.6	1728 per round	Section 5
		B	1050	44.8	1728 per round	Section 5
5	FLIP	2nd ckt	3581	164.9	$\approx 2^{17}$ per bit	Section 6
		3rd ckt	8605	171.9	530 per bit	Section 6

Table 9: Tabulation of Results (Unless other wise stated, power reported at 10 MHz)

in the linear layers of block ciphers PRESENT and GIFT can be executed in a flip-flop array using only two scan flip-flops. While it was already known [Con] that the answer to the above question was yes, a straightforward application of the ideas [Con] would take a lot of clock cycles, and thus affect the throughput of the resulting circuit drastically. Much of the paper is then dedicated to reducing the number of operations required to execute the bit permutation in this setting. As an outcome, we construct extremely lightweight implementations of the PRESENT and GIFT circuits for both encryption (E) and combined encryption+decryption (ED) functionalities. In configuration B, the circuits of both PRESENT and GIFT are the smallest reported so far for both the (E) and (ED) variants. We extend these ideas to construct a circuit for the stream cipher FLIP. The first circuit we investigate is due to a straightforward application of the results [Con], but takes around 2^{25} cycles to produce one keystream bit. This is deemed too impractical to be of any use. The second circuit we construct takes time quadratic in the size of the secret key to produce a keystream bit and occupies only 3581 GE. We then observe that a third circuit that uses slightly different ideas for bit swapping can achieve the FLIP functionality in linear time but occupies around 8605 GE. these are the first reported hardware implementations of FLIP.

References

[BBR16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES: A Compact Implementation of the AES Encryption/Decryption Core. In *Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, pages 173–190, 2016.

[2]Synthesized using IBM 130nm CMOS process
[3]Power reported at 100 KHz

[BBR17a] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Compact circuits for combined AES encryption/decryption. *Journal of Cryptographic Engineering*, pages 1–15, 2017.

[BBR17b] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Efficient configurations for block ciphers with unified ENC/DEC paths. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2017, McLean, VA, USA, May 1-5, 2017*, pages 41–46, 2017.

[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 *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153, 2016.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BPP$^+$17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 321–345, 2017.

[BSS$^+$] Ray Beaulieu, Douglas Shors, Jason Smith, Treatman-Clark Stefan, Bryan Weeks, and Louis Wingers. Simon and Speck: Block Ciphers for the Internet of Things. Available at `https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/papers/session1-shors-paper.pdf`.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 272–288, 2009.

[Con] Keith Conrad. Generating Sets. Available at `http://www.math.uconn.edu/~kconrad/blurbs/grouptheory/genset.pdf`.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[JMPS17] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives - Applications to AES, PRESENT and SKINNY. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 687–707, 2017.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International*

Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I, pages 311–343, 2016.

[RPLP08] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents. In *Smart Card Research and Advanced Applications, 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, pages 89–103, 2008.