Meet-in-the-Middle attack using Correlated Sequences and its Applications to Simon-like Ciphers†

Raghvendra Rohit and Guang Gong

Department of Electrical and Computer Engineering, University of Waterloo,
Waterloo, Ontario, N2L 3G1, CANADA.
{rsrohit, ggong}@uwaterloo.ca

Abstract. This paper presents a generic meet-in-the-middle attack on block ciphers by introducing a novel property of block ciphers called *correlated sequences*. Such sequences exploit the properties of given key dependent sequences of length t to obtain other keyed sequences of same length with σ $(0 \leq \sigma < t)$ computations of the nonlinear function. We call these sequences (σ, t)-*correlated sequences*, utlilize them for $2t$ rounds in a meet-in-the-middle attack, and thereby propose a $2t - 3 + l$ (resp. $2t + l$)-round key recovery attack on Feistel (resp. Substitution Permutation Network) ciphers, where l is the number of rounds of partial encryption.
We apply this technique on Simon-32/64 and Simeck-32/64 by exploiting the *linear segment set* property of the nonlinear function, and show the construction of $(1, 8)$-correlated sequences, i.e., length 8 sequences can be obtained by computing the nonlinear function only once. Using these sequences, we present the first 24, 25 (out of 32)-round attacks on both ciphers which is independent of their respective key scheduling algorithms. The attacks require 3 known plaintext-ciphertext pairs and has a time complexity less than that of average exhaustive search. Next, we show that the attack can be pushed to 27 rounds by considering the key scheduling algorithms and one round differential properties without change in the attack complexities. Our attack improves the previous results on Simon-32/64 and Simeck-32/64 by 4 rounds and has a success probability 1. The attack has been experimentally verified with implementation of attack on small state ciphers.

Keywords: Correlated sequences · Simon · Simeck · Meet-in-the-middle attack · Substitution Permutation Network · Feistel

1 Introduction

Block ciphers are one of the most and well studied symmetric key primitives. It takes n bit plaintext and m bit $(m \geq n)$ master key as a input and outputs a random looking n bit ciphertext after iterating a round function multiple times. Such a $m + n$ to n bit mapping can be defined using two well known constructions namely Feistel structure and Substitution Permutation Network (SPN). A typical example of Feistel cipher[1] is DES [DES77] while AES [DR98] is the most widely deployed SPN block cipher.

A common approach while designing a block cipher is to evaluate its security against the known generic attacks. Accordingly, the number of rounds are determined together

† This article is the revised version of the paper titled **Correlated Sequence Attack on Reduced-Round Simon-32/64 and Simeck-32/64**. https://eprint.iacr.org/2018/699.pdf

[1] We only consider a Feistel network with 2 branches in this work.

with an appropriate security margin[2]. Examples of such attacks include differential and linear cryptanalysis [BS91, Mat93], impossible differential and zero-correlation attacks [BBS99, BR11], integral attacks [KW02] and meet-in-the-middle (MitM) attack [DH77] to name a few.

Over the past few years, the above cryptanalytic techniques in concurrent with automated tools such as SAT/SMT solver and Mixed Integer Linear Programming (MILP) have been extensively used in the design and analysis of block ciphers. As a result, several lightweight block ciphers have emerged specifically to target the challenges posed by resource constrained environments such as RFID (EPC tags and NFC), IoT devices and sensor networks. Examples of lightweight block ciphers encompass PRESENT [BKL+07], LED [GPPR11], HIGHT [HSH+06], PICCOLO [SIH+11], SIMON and SPECK [BTCS+13], SIMECK [YZS+15], SKINNY [BJK+16] and GIFT[BPP+17].

Among most of the aforementioned block ciphers, Simon [BTCS+13] designed by the US National Security Agency (NSA) in 2013 acheive overwhelming performance in hardware due to its simple quadratic round function which consists of only bitwise XORS and ANDs. Later in CHES 2015, Yang *et al.* [YZS+15] proposed Simeck that has a smaller hardware footprint than Simon, by combining the good design components of both Simon and Speck. However, the attacks on Simon are directly applicable to Simeck because of a similar structure. Accordingly, Simon-like ciphers, i.e., Feistel ciphers with a similar nonlinear function[3] as Simon attracted a lot of attention from the cryptographic community.

In this work, we propose a generic meet-in-the-middle attack on block ciphers by introducing a novel property called correlated sequences. As an application of our attack, we break up to 85% (27/32) rounds of Simon-32/64 and Simeck-32/64 while the previous attacks can reach only 23 rounds. Although, our attack has time complexity close to average exhaustive search, it has a very low data complexity and success probability 1. In addition, it depict weaknesses in Simon-like ciphers which were not investigated before. Table 1 presents a summary of state-of-the-art cryptanalytic results on Simon-32/64 and Simeck-32/64.

Related work. Several papers have analyzed the security and investigated the parameter choices of the round function to get a deeper understanding of design rationale of Simon-like ciphers [ALLW14, CW16, CCW+18, FSW16, KLT15, KR16, KSI16, LLW17, QHS16, QCW16, SHMS14, TM16, WLV+14, WWJZ14, ZGHL16]. Currently, the best cryptanalytic results on Simon and Simeck are reduced-round differential/linear and integral attacks. The smaller versions, namely Simon-32/64 and Simeck-32/64, with blocksize and key length, 32 and 64-bit, respectively, have security margin of 28% with MitM attack [SHMS14] covering only 18 rounds. Up to our knowledge, the only 24 round attack on Simon-32/64 was proposed in [CCW+18]. However, it requires full codebook and has time complexity 2^{63} (see Table 1).

Our contributions. In this paper, we propose a generic meet-in-the-middle attack on block ciphers by introducing a novel property called *correlated sequences*. We apply this attack on reduced-round Simon-32/64 and Simeck-32/64, and present the first 24, 25, 26 and 27-round key recovery attacks on both ciphers. Thus, our main contributions are summarized as follows.

1. We introduce a novel property of block ciphers called correlated sequences. For a fixed key k, we consider t rounds of cipher as a keyed sequence of length t, i.e, $S_{(k,t)} = (s_0, s_1, \ldots, s_{t-1})$, where s_i is the state at i-th round. Our technique exploit the properties of $S_{(k,t)}$ to obtain $S_{(k',t)}$ with σ $(0 \leq \sigma < t)$ computations of the nonlinear function. We call these sequences (σ, t)-*correlated sequences*. For a

[2] $\left(1 - \frac{\#\text{ attacked rounds}}{\#\text{ full rounds}}\right) \times 100$

[3] Different shift parameters, for e.g., the shift parameters of Simon and Simeck are (8,1,2) and (5,0,1), respectively.

Table 1: Summary of attacks on Simon-32/64 and Simeck-32/64

Attack	Cipher	# attacked rounds / 32	Data	Memory (Bytes)	Time	Success rate
Differential	Simon-32/64 [WWJZ14]	21	2^{31}	-	$2^{55.25}$	0.51
	Simon-32/64 [QHS16]	22	2^{32}	-	$2^{58.76}$	0.315
	Simeck-32/64 [KR16]	19	2^{31}	2^{33}	2^{40}	-
	Simeck-32/64 [QHS16]	22	2^{32}	-	$2^{57.9}$	0.417
Linear	Simon-32/64 [CW16]	23	$2^{31.19}$	-	$2^{61.84}$	0.277
	Simeck-32/64 [QCW16]	23	$2^{31.91}$	-	$2^{61.78}$	0.456
Integral	Simon-32/64 [WLV+14]	21	2^{31}	2^{54}	2^{63}	1
	Simon-32/64 [FSW16]	22	2^{31}	$2^{55.8}$	2^{63}	1
	Simon-32/64 [CCW+18]	24	2^{32}	$2^{33.64}$	2^{63}	1
	Simeck-32/64 [ZGHL16]	21	2^{31}	$2^{46.22}$	2^{63}	1
Impossible Differential	Simon-32/64 [DF16]	20	2^{32}	$2^{45.5}$	$2^{62.8}$	-
	Simeck-32/64 [YZS+15]	20	2^{32}	2^{58}	$2^{62.5}$	-
Zero correlation	Simon-32/64 [SFW15]	21	2^{32}	2^{31}	$2^{59.4}$	-
	Simeck-32/64 [ZGHL17]	21	2^{32}	$2^{47.67}$	$2^{58.78}$	-
Meet-in-the-middle	Simon-32/64 [SHMS14]	18	8	2^{52}	$2^{62.57}$	1
Correlated sequences Sections 5 and 6	Simon-32/64	**24**	3	2^{49}	$2^{62.87}$	1
		25	3	2^{49}	$2^{62.94}$	1
		26	3	2^{49}	$2^{62.88}$	1
		27	3	2^{49}	$2^{62.94}$	1
	Simeck-32/64	**24**	3	2^{49}	$2^{62.87}$	1
		25	3	2^{49}	$2^{62.94}$	1
		26	3	2^{49}	$2^{62.88}$	1
		27	3	2^{49}	$2^{62.94}$	1

r-round block cipher with (σ, t)-correlated sequences, we propose a generic $r_1 + r_2$ rounds MitM attack where $r_1 + r_2 \leq r$, $r_1 = 2t - 3$ (resp. $2t$) for Feistel (resp. SPN) ciphers, and r_2 is the number of rounds of partial encryption. Our attack has a time complexity strictly less than that of trivial exhaustive search and requires only $\left\lceil \frac{\text{keysize}}{\text{blocksize}} \right\rceil + 1$ known plaintext-ciphertext pairs.

2. We apply the method of correlated sequences to Simon-like ciphers where the key length is twice the blocksize. We investigate the *linear segment set* properties of the quadratic nonlinear function and show that all $(1, 8)$-correlated sequences can be obtained by computing the nonlinear function only once. Using these sequences, we present the first 24, 25-round key recovery attack on Simon-32/64 and Simeck-32/64 which is independent of the key scheduling algorithms. The attack requires 3 known plaintext-ciphertext pairs and has a memory complexity of 2^{49} bytes. The time complexities are $2^{62.87}$(resp. $2^{62.94}$) for 24 (resp. 25)-round attacks.

3. By incorporating the properties of correlated sequences with key scheduling algorithms and one round differentials, we show that the attack can be pushed to 2 more rounds. Accordingly, we present a key recovery attack on 26 (27)-round Simon-32/64 and Simeck-32/64 which has same complexities as of the 24 (25)-round attack.

4. We provide an implementation of our attack in Python for the smaller cases, i.e., Simon-8/16 (Simeck-8/16) and Simon-16/32 (Simeck-16/32). Our experimental results have a success probability 1 and hence proves the attack's soundness for Simon-32/64 and Simeck-32/64.

Outline. The rest of the paper is organized as follows. In Section 2, we review the

meet-in-the-middle attack and specification of Simon-like block ciphers. Section 3 presents the definitions and basic properties of the correlated sequences, and applications of these sequences to MitM attack on Feistel and SPN ciphers. In Section 4, we provide the theoretical construction of correlated sequences for Simon-like ciphers. Section 5 presents the application of correlated sequences to 25-round key recovery attack on Simon-32/64 and Simeck-32/64. In Section 6, we show that the attack can be improved by 2 rounds leading to 27 round key recovery attack. Finally, the paper is concluded in Section 7.

2 Preliminaries

In this section, we briefly review the meet-in-the-middle attack and give a brief description of Simon-like ciphers. The notations used throughout the paper are defined in Table 2.

Table 2: Notations

Notation	Description
$+$	bitwise XOR
$\&$	bitwise AND
$\|\|$	concatenation operator
n	wordsize
$\mathcal{K}$	key space
$\mathbb{F}_2$	{0,1}
$\mathbb{F}_2^n$	n dimensional vector space over $\mathbb{F}_2$
L^i	left cyclic shift operator, i.e., for $x \in \mathbb{F}_2^n$, $L^i(x) = (x_i, x_{i+1}, \ldots, x_{n-1}, x_0, x_1, \ldots, x_{i-1})$
C_s	coset modulo $2^n - 1$, i.e., $C_s = \{s, 2s, \ldots, 2^{n_s-1}s\}$ where n_s is the smallest number such that $s \equiv 2^{n_s}s \bmod 2^n - 1$, and s is the smallest number in C_s and denotes the coset leader
$\|S\|$	cardinality of set S
$A[i]$	i-th element of A
$Img(f)$	Image set of f

2.1 Meet-in-the-Middle attack

Consider an encryption algorithm Enc that takes plaintext P and secret key k as a input and gives the ciphertext $C = \mathsf{Enc}(k, P)$. The basic idea is to decompose Enc as a composition of two subciphers (see Figure 1) such that $C = \mathsf{Enc}(k, P) = \mathsf{Enc}_b(k_b, \mathsf{Enc}_f(k_f, P))$.

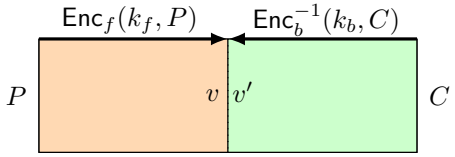

Figure 1: MitM attack

The steps of standard MitM attack is divided into two phases.

1. **MitM phase.** For all possible values of k_f, compute $v = \mathsf{Enc}_f(k_f, P)$ and store (k_f, v) in data structure DS. We then compute $v' = \mathsf{Enc}_b^{-1}(k_b, C)$ and check if $v' \in \mathsf{DS}$. If so, the value of k corresponding to the pair (k_f, k_b) is one of the key candidate.

2. **Brute force phase.** If the number of keys obtained from MitM phase is more than one, we perform an exhaustive search on additional plaintext-ciphertext pairs to get the correct key.

2.2 Specification of Simon-like block ciphers

Simon-$2n/mn$, where $2n$ and mn denote the blocksize and key length, respectively, is a family of block ciphers proposed by NSA in 2013 [BTCS+13]. A generic diagram of Simon-like block cipher is depicted in Figure 2. It adopts a Non-Linear Feedback Shift Register (NLFSR) based structure where the nonlinearity comes from the quadratic function $f_{(a,b,c)}(x) = L^a(x)\&L^b(x) + L^c(x)$. We refer to $f_{(a,b,c)}$ as Simon-like nonlinear function unless the parameter set (a, b, c) is explicitly mentioned.

For r-round cipher, the $(i+2)$-th element of NLFSR sequence is given by $s_{i+2} = f_{(a,b,c)}(s_{i+1}) + s_i + k_i$ where $k_i \in \mathbb{F}_2^n$ is the i-th round subkey[4] and $0 \leq i < r$. Finally, the ciphertext is the r-th state of NLFSR, i.e., (s_{r+1}, s_r). The shift parameters of Simon are given by $(a, b, c) = (8, 1, 2)$.

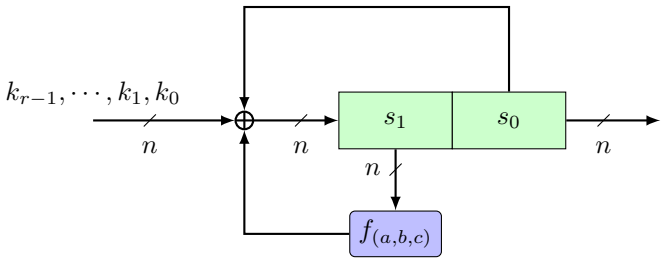

Figure 2: Simon-like block cipher

Simeck-$2n/mn$ was proposed in CHES 2015 by Yang *et al.* [YZS+15] and adopts a Simon-like structure with the shift parameters given by $(5, 0, 1)$. However, it has more efficient and compact hardware implementation because of reuse of the round function in the key scheduling algorithm.

Key scheduling algorithms. For $n = 16$ and $m = 4$, $r = 32$ and the subkeys are calculated as follows.

$$\begin{aligned} \text{Simon-32/64}: &\quad k_{i+4} = Z_i + k_i + k_{i+1} + L^{15}(k_{i+1}) + L^{13}(k_{i+3}) + L^{12}(k_{i+3}), \\ \text{Simeck-32/64}: &\quad k_{i+4} = Z_i + f_{(5,0,1)}(k_{i+1}) + k_i. \end{aligned}$$

The attack presented in this paper is not affected by the constants Z_i and the reader may refer to [BTCS+13, YZS+15] for more details of their respective key scheduling algorithms.

3 Correlated Sequences of Block Ciphers

In this section, we formally introduce the correlated sequences and then show how to use them in a meet-in-the-middle attack. Consider an n-bit block cipher with r rounds and mn-bit master key $k = (k_0, k_1, \ldots, k_{m-1})$ as depicted in Figure 3. Let s_i denote the state

[4] $k_0, k_1, \ldots, k_{m-1}$ are first m n-bit words of key.

at i-th round. Then, for $0 \leq i < r$, $s_{i+1} = \mathsf{rf}(s_i, k_i)$ where rf denotes the round function, and is generally a composition of two functions, namely i) a linear function χ and ii) a nonlinear function ρ.

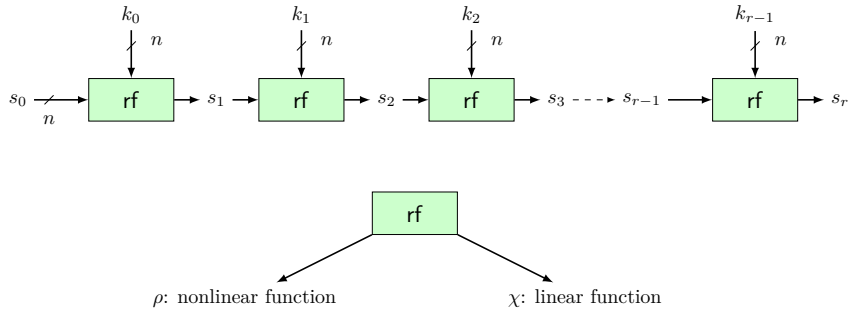

Figure 3: Generic diagram of a block cipher

3.1 General idea

Definition 1 (Keyed sequence)**.** Given $k \in \mathcal{K}$ and $1 \leq t < r$. We say $S_{(k,t)} = (s_0, s_1, \ldots, s_{t-1})$ is a keyed sequence of length t, if $s_{i+1} = \mathsf{rf}(s_i, k_i)$ for $0 \leq i < t-1$.

From Definition 1, it is clear that we need to compute rf t times to obtain $S_{(k,t)}$. This implies that ρ is computed t times in total. Thus, to obtain another sequence $S_{(k',t)}$ of same length t, the worst case is to compute ρ exactly t times. The idea of correlated sequences is "Given $S_{(k,t)}$ and $k' \neq k$, obtain the sequence $S_{(k',t)}$ by computing the nonlinear function ρ at most t times."

We now present a toy example to illustrate this notion before providing the formal definition.

Example 1. Consider a 4-bit toy Simon-like block cipher with 8-bit blocksize and 16-bit key as shown in Figure 4. Let the non-linear function is given by $\rho(x) = x\&L(x) + L^2(x)$ where $x \in \mathbb{F}_2^4$. The length seven keyed sequences are given in Table 3. We note the following observations from Table 3.

1. For all $k = (k_0, k_1, k_2, k_3)$, $s_4 = k_2$, $s_5 = 0$ and $s_6 = k_2 + k_4$.
2. For all $k' = (k'_0, k'_1, k'_2.k'_3)$, $s'_4 = k'_2$, $s'_5 = 1$ and $s'_6 = k'_2 + k'_4 + \rho(1)$.
3. For each row, $k'_3 = k_3 + 1$ and $s'_6 = s_6 + k_4 + k'_4 + \rho(1)$.

We now define the correlated sequences in Definition 2.

Definition 2 ((σ, t)-correlated sequences)**.** Given $S_{(k,t)}$ and $0 \leq \sigma < t$. We say $S_{(k,t)}$ and $S_{(k',t)}$ are (σ, t)-correlated sequences if $S_{(k',t)}$ can be obtained from $S_{(k,t)}$ by computing the nonlinear function ρ exactly σ times.

Remark 1. $\sigma = 0 \implies S_{(k,t)}$ and $S_{(k',t)}$ are linearly related.

Definition 3 (Zero correlated keys)**.** Given $S_{(k,t)}$. We define zero correlated keys as the set $\mathsf{CK}(k) = \{k' \mid S_{(k,t)}$ and $S_{(k',t)}$ are $(0,t)$-correlated sequences$\}$.

For example, in Table 3, for each row $S_{(k,t)}$ and $S_{(k',t)}$ are $(1,7)$ correlated sequences, $|\mathsf{CK}((0,0,0,0))| = 15$ (gray colored rows) and $|\mathsf{CK}((0,0,0,1))| = 15$ (light gray rows). Thus, to obtain all 32 sequences, we only need to compute $\rho(1)$ once.

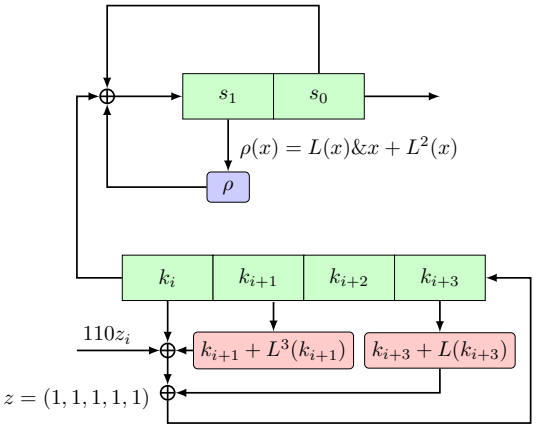

Figure 4: 4-bit toy Simon-like cipher

Table 3: Keyed sequences

k_0	k_1	k_2	k_3	k_4	s_0	s_1	s_2	s_3	s_4	s_5	s_6	k'_0	k'_1	k'_2	k'_3	k'_4	s'_0	s'_1	s'_2	s'_3	s'_4	s'_5	s'_6
0	**0**	**0**	**0**	**13**	**0**	**0**	**0**	**0**	**0**	**0**	**13**	**0**	**0**	**0**	**1**	**14**	**0**	**0**	**0**	**0**	**0**	**1**	**10**
0	0	1	4	1	0	0	0	0	1	0	0	0	0	1	5	2	0	0	0	0	1	1	7
0	0	2	8	4	0	0	0	0	2	0	6	0	0	2	9	7	0	0	0	0	2	1	1
0	0	3	14	14	0	0	0	0	3	0	13	0	0	3	15	13	0	0	0	0	3	1	10
0	0	4	1	14	0	0	0	0	4	0	10	0	0	4	0	13	0	0	0	0	4	1	13
0	0	5	5	2	0	0	0	0	5	0	7	0	0	5	4	1	0	0	0	0	5	1	0
0	0	6	13	11	0	0	0	0	6	0	13	0	0	6	12	8	0	0	0	0	6	1	10
0	0	7	11	1	0	0	0	0	7	0	6	0	0	7	10	2	0	0	0	0	7	1	1
0	0	8	2	11	0	0	0	0	8	0	3	0	0	8	3	8	0	0	0	0	8	1	4
0	0	9	7	4	0	0	0	0	9	0	13	0	0	9	6	7	0	0	0	0	9	1	10
0	0	10	10	2	0	0	0	0	10	0	8	0	0	10	11	1	0	0	0	0	10	1	15
0	0	11	13	11	0	0	0	0	11	0	0	0	0	11	12	8	0	0	0	0	11	1	7
0	0	12	11	1	0	0	0	0	12	0	13	0	0	12	10	2	0	0	0	0	12	1	10
0	0	13	14	14	0	0	0	0	13	0	3	0	0	13	15	13	0	0	0	0	13	1	4
0	0	14	7	4	0	0	0	0	14	0	10	0	0	14	6	7	0	0	0	0	14	1	13
0	0	15	0	13	0	0	0	0	15	0	2	0	0	15	1	14	0	0	0	0	15	1	5

3.2 Application of correlated sequences to MitM attack

Let (s_0, s_r) denote the plaintext and ciphertext pair encrypted with the mn-bit master key k. As depicted in Figure 5, we first use s_0 to construct (σ, t)-correlated sequences and their corresponding $\mathsf{CK}(\cdot)$ for t rounds. Next, starting with s_r, we follow the same approach. We then do partial encryption for r_2 rounds starting from t-th round and match the state values at $(t+r_2)$-th round. Thus, for a SPN cipher the number of attacked rounds is $2t + r_2$. Considering a Feistel cipher, the state at i-th round equals (s_{i+1}, s_i) where $s_i = \mathsf{rf}(s_{i-1}, k_{i-2}) + s_{i-2}$ and $i \geq 2$. Accordingly, the encryption of (s_1, s_0) gives (s_{r+1}, s_r) and the number of attacked rounds is $2t - 4 + r_2$, and equals $2t - 3 + r_2$ if matching is done on half state.

Time complexity. Let T^e (resp. T^d) denote the number of computations of ρ to construct (σ, t)-correlated sequences and their corresponding $\mathsf{CK}(\cdot)$ in encryption (resp. decryption) direction. Then, the time complexity in terms of the number of computations of ρ is given by $T = T^e + T^d + 2^{|\mathcal{K}|} \times \frac{r_2}{r}$. Clearly, $T < 2^{|\mathcal{K}|}$.

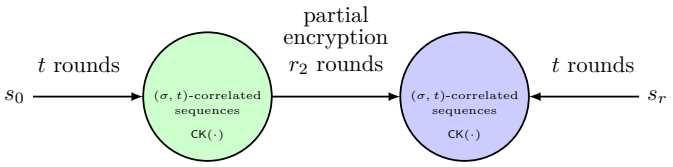

Figure 5: MitM attack using correlated sequences

Data complexity. The above attack filters $2^{n(m-1)}$ keys that map s_0 to s_r. The correct key can then be found out by performing an exhaustive search on the remaining known $\lceil \frac{mn}{n} \rceil - 1$ plaintext-ciphertext pairs. Note that for a Feistel cipher, an additional plaintext-ciphertext pair is needed if matching is done on half state.

4 Correlated Sequences of Simon-like Ciphers

In this section, we show the construction of correlated sequences of Simon-like ciphers where the key length is twice the block size. We first look at the theoretical properties of non-linear function $f_{(a,b,c)}$. Next, we use these properties to construct $(1, 8)$-correlated sequences. We assume that $a \neq b \neq c$.

4.1 Properties of Simon-like nonlinear function

Property 1. Let s be the coset leader corresponding to the coset C_s, then for $0 \leq i < |C_s|$, the following hold.

1. $f_{(a,b,c)}(L^i(s)) = L^i(f_{(a,b,c)}(s))$
2. $f_{(a,b,c)}(s) = L^{a-1}(s) + L^{b-1}(s) + L^{c-1}(s)$, if $s = \underbrace{011\ldots1}_{n}$.

Property 2. Let $s = \underbrace{0101\ldots01}_{n}$ and a, b are not both simultaneously even or odd, then

$$f_{(a,b,c)}(s) = \begin{cases} s & \text{if } c \equiv 0 \bmod 2 \\ L(s) & \text{otherwise} \end{cases}$$

Properties 1 and 2 imply that it is enough to compute the values of $f_{(a,b,c)}$ for $\frac{2^n-1}{n}$ coset leaders only. However, as $f_{(a,b,c)}$ is quadratic and the only linear term involved in it is $L^c(.)$, hence $f_{(a,b,c)}(x) = L^c(x) + z$ for all $x \in \mathbb{F}_2^n$ and some constant $z \in \mathbb{F}_2^n$. As a result, we partition the coset leaders based on the values of z. Since, $f_{(a,b,c)}$ is linear on each partition, we call such partition as *z-linear segment set* and formally define it in Definition 4 as follows.

Definition 4 (z-linear segment set)**.** The z-linear segment set of $f_{(a,b,c)}$ is the set of coset leaders CL_z given by $CL_z = \{s \mid f_{(a,b,c)}(s) + L^c(s) = z\}$.

Table 4 lists the z-linear segment sets for $n = 8$ and $(a, b, c) = (8, 1, 2)$, while the number of z-linear segments (denoted by N_z) for varying n are presented in Table 5. (Note that since $n = 8$, the shifts $(8, 1, 2)$ is equivalent to $(0, 1, 2)$.)

Example 2. In Table 4, consider $z = 2$ and $3 \in CL_2$. Then, for all

$$x \in C_3 = \{3, 6, 12, 24, 48, 96, 192, 129\},$$

$f_{(8,1,2)}$ is computed as follows.

x	$f_{(8,1,2)}(x)$
3	$L^2(3) + 2 = 14$
6	$L^2(6) + L(2) = 28$
12	$L^2(12) + L^2(2) = 56$
24	$L^2(24) + L^3(2) = 112$
48	$L^2(28) + L^4(2) = 224$
96	$L^2(96) + L^5(2) = 193$
192	$L^2(192) + L^6(2) = 131$
129	$L^2(129) + L^7(2) = 7$

Table 4: z-linear segment sets for $n = 8$ and $(a, b, c) = (8, 1, 2)$

z	CL_z	z	CL_z
0	{0, 1, 5, 9, 17, 21, 37, 85}	2	{3, 11, 19, 43}
6	{7, 23, 39, 87 }	8	{13, 45}
14	{15, 47}	16	{25}
18	{27, 91 }	24	{29}
30	{31, 95}	32	{53}
34	{51}	38	{55}
50	{59}	56	{61}
62	{63}	78	{111}
102	{119}	126	{127}
255	{255}	-	-

4.2 Construction of $(1, 8)$-correlated sequences

Let (s_0, s_1) be any random $2n$-bit value and $\mathrm{K}_{(k_0,k_1)} = \{(k_0, k_1, k_2, k_3) \mid (k_2, k_3) \in \mathbb{F}_2^n \times \mathbb{F}_2^n\}$ be the set of 2^{2n} keys with k_0 and k_1 fixed to some constant value. For $t \geq 6$ and $0 \leq i < 2^n$, define

$$\mathrm{P}(i, t, \mathrm{K}_{(k_0,k_1)}) = \{(k, S_{(k,t)}) \mid k \in \mathsf{K}_{(k_0,k_1)} \text{ and } s_5 = i\}$$

as the set of keys and their corresponding sequences which maps s_5 to i.

We start with the simpler case, i.e., $s_5 = 0$. First, we construct $\mathsf{P}(0, 8, \mathsf{K}_{(k_0,k_1)})$ and then show how to construct $\mathsf{P}(i, 8, \mathsf{K}_{(k_0,k_1)})$ from the knowledge of $\mathsf{P}(0, 8, \mathsf{K}_{(k_0,k_1)})$.

4.2.1 Construction of $\mathsf{P}(0, 8, \mathsf{K}_{(k_0,k_1)})$

We divide the construction of $\mathsf{P}(0, 8, \mathsf{K}_{(k_0,k_1)})$ into 3 steps, namely i) Finding $\mathsf{P}(0, 6, \mathsf{K}_{(k_0,k_1)})$, ii) Obtaining $\mathsf{P}(0, 7, \mathsf{K}_{(k_0,k_1)})$ from $\mathsf{P}(0, 6, \mathsf{K}_{(k_0,k_1)})$, and iii) Obtaining $\mathsf{P}(0, 8, \mathsf{K}_{(k_0,k_1)})$ from $\mathsf{P}(0, 7, \mathsf{K}_{(k_0,k_1)})$. For each step, we denote the number of computations of $f_{(a,b,c)}$ by T_{step}. We now present the details of each step as follows.

Step 1: Finding $\mathsf{P}(0, 6, \mathsf{K}_{(k_0,k_1)})$. We note that $\forall k \in \mathsf{K}_{(k_0,k_1)}$, $S_{(k,4)}$ is a constant sequence and requires only 2 computations of $f_{(a,b,c)}$. Hence, finding the keys for which $s_5 = 0$ is equivalent to solve $f_{(a,b,c)}(X + k_2) = k_3 + s_3$ where $X = f_{(a,b,c)}(s_3) + s_2$. We use z-linear segments (see Definition 4) to solve this equation. As a result, $T_{step_1} = 3 + N_z$. Note that $|\mathsf{P}(0, 6, \mathsf{K}_{(k_0,k_1)})| = 2^n$, as $s_4 = X + k_2$ can take all 2^n distinct values.

Table 5: Number of z-linear segment sets for varying n

n	# coset leaders	N_z	
		(a,b,c)	
		$(8,1,2)$	$(5,0,1)$
8	36	20	17
10	108	42	14
12	352	119	119
14	1182	50	287
16	4116	909	798

Thus, for all $(k, S_{(k,6)}) \in \mathsf{P}(0,6,\mathsf{K}_{(k_0,k_1)})$, the pair (k_2, k_3) is unique. Accordingly, let $\mathsf{I}_{(k_0,k_1)} = \{k_3 \mid (k, S_{(k,6)}) \in \mathsf{P}(0,6,\mathsf{K}_{(k_0,k_1)})\}$, then $|\mathsf{I}_{(k_0,k_1)}| = 2^n$.

Step 2: Obtaining $\mathsf{P}(0,7,\mathsf{K}_{(k_0,k_1)})$ from $\mathsf{P}(0,6,\mathsf{K}_{(k_0,k_1)})$. Let $(k, S_{(k,6)}) \in \mathsf{P}(0,6,\mathsf{K}_{(k_0,k_1)})$ and consider the following relation $s_4 + s_6$. We have $s_4 + s_6 = s_4 + f_{(a,b,c)}(s_5) + s_4 + k_4 = s_4 + 0 + s_4 + k_4 \implies s_6 = s_4 + k_4$. Thus, $T_{step_2} = 0$.

Step 3: Obtaining $\mathsf{P}(0,8,\mathsf{K}_{(k_0,k_1)})$ from $\mathsf{P}(0,7,\mathsf{K}_{(k_0,k_1)})$. Let $(k, S_{(k,7)}) \in \mathsf{P}(0,7,\mathsf{K}_{(k_0,k_1)})$. We compute s_7 as follows.

$$\begin{aligned} s_7 &= f_{(a,b,c)}(s_6) + s_5 + k_5 = f_{(a,b,c)}(s_6) + k_5 \\ &= f_{(a,b,c)}(s'_4) + k_5 \text{ (By step 1)} \\ &= f_{(a,b,c)}(X + k'_2) + k_5 = s'_3 + k'_3 + k_5 \\ &= s_3 + \mathsf{I}_{(k_0,k_1)}[k'_2] + k_5 \text{ (as } s'_3 = s_3) \\ &= s_3 + \mathsf{I}_{(k_0,k_1)}[k_2 + k_4] + k_5. \end{aligned}$$

Note that since $s_6 = X + k'_2 \implies k'_2 = s_6 + X = s_4 + k_4 + X = k_2 + k_4$. Furthermore $T_{step_3} = 0$.

Corollary 1. *Given $\mathsf{P}(0,8,\mathsf{K}_{(k_0,k_1)})$, $\mathsf{I}_{(k_0,k_1)}$ and $k = (k_0, k_1, 0, \mathbf{I}_{(k_0,k_1)}[0])$. Then*

$$|\mathsf{CK}(k)| = 2^n - 1.$$

We could use the similar construction shown above to get $\mathsf{P}(i,8,\mathsf{K}_{(k_0,k_1)})$ for $1 \leq i < 2^n$. However, this would require $2^n(3 + N_z)$ computations of $f_{(a,b,c)}$ in total. Next, we show how to reduce this number to $(3 + 2N_z)$.

4.2.2 Computing $\mathsf{P}(i,8,\mathsf{K}_{(k_0,k_1)})$ from $\mathsf{P}(0,8,\mathsf{K}_{(k_0,k_1)})$

Theorem 1. *Given $\mathsf{I}_{(k_0,k_1)}$, $k = (k_0, k_1, 0, \mathsf{I}_{(k_0,k_1)}[0])$, $(k, S_{(k,8)}) \in \mathsf{P}(0,8,\mathsf{K}_{(k_0,k_1)})$ and $X = f_{(a,b,c)}(s_3) + s_2$. Let $1 \leq i < 2^n$ and $\tilde{k} = (k_0, k_1, 0, \mathsf{I}_{(k_0,k_1)}[0] + i)$. Then the following hold.*

1. $S_{(k,5)} = S_{(\tilde{k},5)}$
2. $(\tilde{k}, S_{(\tilde{k},6)}) \in \mathsf{P}(i,6,\mathsf{K}_{(k_0,k_1)})$
3. $\tilde{s_6} = s_3 + \mathsf{I}_{(k_0,k_1)}[X + i] + X + \tilde{k_2} + \tilde{k_4}$
4. $\tilde{s_7} = s_3 + i + \tilde{k_5} + \mathsf{I}_{(k_0,k_1)}[\tilde{s_6} + X]$

5. $|CK(\bar{k})| = 2^n - 1$

Proof. 1. Since $k_2 = \tilde{k_2} = 0 \implies s_4 = \tilde{s_4} = X \implies S_{(k,5)} = S_{(\tilde{k},5)}$.

2. It is enough to show that $\tilde{s_5} = i$. We have

$$\begin{aligned} \tilde{s_5} &= f_{(a,b,c)}(\tilde{s_4}) + \tilde{s_3} + \tilde{k_3} = f_{(a,b,c)}(s_4) + s_3 + \tilde{k_3} \\ &= \mathsf{l}_{(k_0,k_1)}[0] + s_3 + s_3 + \mathsf{l}_{(k_0,k_1)}[0] + i = i. \end{aligned}$$

3.

$$\begin{aligned} \tilde{s_6} &= f_{(a,b,c)}(\tilde{s_5}) + \tilde{s_4} + \tilde{k_4} = f_{(a,b,c)}(i) + \tilde{s_4} + \tilde{k_4} \\ &= s_3 + \mathsf{l}_{(k_0,k_1)}[X + i] + X + \tilde{k_2} + \tilde{k_4}. \end{aligned}$$

4. The proof is similar to previous part.

5. Note that for $1 \leq j < 2^n$, $(k_0, k_1, j, \mathsf{l}_{(k_0,k_1)}[j]) \in \mathsf{CK}(k) \iff (k_0, k_1, j, \mathsf{l}_{(k_0,k_1)}[j] + i) \in \mathsf{CK}(\bar{k})$. This follows because $s_5 + \bar{s_5} = k_3 + \bar{k_3} \implies k_3 + \bar{k_3} = i$. Thus, $|\mathsf{CK}(\bar{k})| = 2^n - 1$. □

We use Theorem 1 with z-linear segment sets to compute all partitions. A brief comparison of different approaches with the number of computations of $f_{(a,b,c)}$ to obtain $\mathsf{P}(i, 8, \mathsf{K}_{(k_0,k_1)})$ is provided in Table 6.

Table 6: Comparison of different approaches with the number of computations of $f_{(a,b,c)}$ for 6 out of r rounds

Approach	# computations of $f_{(a,b,c)}$	
	(a,b,c)	
	$(8,1,2)$	$(5,0,1)$
Naive	$2^{64} \times \frac{6}{r}$	$2^{64} \times \frac{6}{r}$
Theorem 1 and z-linear segment sets	$2^{32} \times \frac{(3+1818)}{r}$	$2^{32} \times \frac{(3+1596)}{r}$

5 Key Recovery Attack on 25 rounds Simon-32/64 and Simeck-32/64

In this section, we show the key recovery attack procedure on 25-round Simon-32/64 and Simeck-32/64. We note that construction of $(1, 8)$-correlated sequences as shown in Section 4 is independent of the key scheduling algorithms. Thus, we simply utilize these sequences for 6 encryption and 6 decryption rounds in a MitM attack (see Figure 6). As a result, we do partial encryption for 12 rounds, starting from round 6 and match the left half of state, i.e., s_{19} at 19-th round.

From now, we denote s_i^e/s_i^d, k_i^e/k_i^d, X^e/X^d, $\mathsf{l}_{(k_0^e,k_1^e)}/\mathsf{l}_{(k_0^d,k_1^d)}$ and $\mathsf{DS^e}/\mathsf{DS^d}$ as the i-th element of keyed sequence, i-th subkey, value of $f_{(a,b,c)}(s_3^e)+s_2^e$ / $f_{(a,b,c)}(s_3^d)+s_2^d$, indexing set and stored data structure from encryption/decryption side, respectively. For example, $s_0^e = s_0$, $s_1^e = s_1$, $s_0^d = s_{26}$, $s_1^d = s_{25}$, $k_{24}^e = k_0^d$ and so on.

In Algorithm 1, we present a generic procedure for recovering the secret key. It takes input as 3 known plaintext-ciphertext pairs encrypted either by Simon-32/64 or Simeck-32/64 and returns the secret key. The attack procedure is divided into two phases, namely i) Offline phase and ii) Online phase. The time complexities of both phases are given by $T^{offline}$ and T^{online}, where a subscript (for e.g., T_i^{online}) denotes the time complexity of i-th step of the corresponding phase. In what follows, we present the details of both phases.

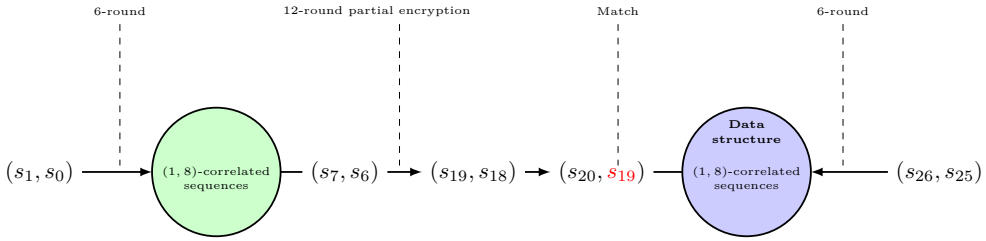

Figure 6: 25-round key recovery procedure

5.1 Offline phase

In this phase, we first compute z-linear segment sets using Defintion 4. Next, we construct a data structure $\mathsf{DS^d}$ that is used in the online phase to compute the value of s_7^d without doing any nonlinear operation. Note that in order to compute s_7^d for any key $k = (k_0^d, k_1^d, k_2^d, k_3^d)$, we only need the values of s_3^d, X^d and $\mathsf{I}_{(k_0^d, k_1^d)}$ (see Theorem 1). Hence, we store the array $[s_3^d, X^d, \mathsf{I}_{(k_0^d, k_1^d)}]$ as the $(L^{16}(k_0^d)||k_1^d)$-th row of $\mathsf{DS^d}$ for a fixed (k_0^d, k_1^d) pair.

The procedures compute_zs and construct_ds in Algorithm 1 constitute the offline phase. In Appendix A, we provide an example of $\mathsf{DS^d}$ for a toy Simon cipher.

Memory complexity. The memory required to store z-linear segment sets is $(N_z +$ # coset leaders$) \times 16$ bit. Furthermore, to store a single row of $\mathsf{DS^d}$, $(1 + 1 + 2^{16}) \times 16$ bit space is needed. Thus, the total memory (Mem) is given by:

$$\begin{aligned} \mathsf{Mem}_{\mathsf{Simon\text{-}32/64}} &= (N_z + \#\text{ coset leaders}) \times 16 + 2^{32} \times (1 + 1 + 2^{16}) \times 16 \\ &= (909 + 4116) \times 16 + 2^{32} \times (2 + 2^{16}) \times 16 \approx 2^{49} \text{ bytes.} \end{aligned}$$

Similarly, $\mathsf{Mem}_{\mathsf{Simeck\text{-}32/64}} \approx 2^{49}$ bytes as $N_z = 798$ (see Table 5).

Time complexity. The time complexity in terms of the number of computations of $f_{(a,b,c)}$ is given by:

$$\begin{aligned} T^{offline} &= T_0^{offline} + T_1^{offline} \\ &= \#\text{ coset leaders} + 2^{32} \times \underbrace{\left(\frac{3 + N_z}{25}\right)}_{\#\text{ computations of } f_{(a,b,c)} \text{ to get } \mathsf{I}_{(k_0^d, k_1^d)}} \end{aligned}$$

Thus, $T^{offline} \approx 2^{37.18}$ and 2^{37} for Simon-32/64 and Simeck-32/64, respectively.

5.2 Online phase

In this phase, we recover the secret key that maps the plaintext $(s_0^{e,i}, s_1^{e,i})$ to ciphertext $(s_0^{d,i}, s_1^{d,i})$ for $i = 0, 1, 2$. We first find the key set K that maps $(s_0^{e,0}, s_1^{e,0})$ to $(s_0^{d,0}, s_1^{d,0})$ using filter_keys procedure in Algorithm 2. Note that $|K| = 2^{48}$ as partial matching is done at 19-th round (see step 16 of Algorithm 2). Next, we perform an exhaustive search on remaining 2 plaintext-ciphertext pairs to get the correct key (steps 31-41 of Algorithm 1). We now present the details of filter_keys procedure.

Procedure filter_keys. For a fixed (k_0^e, k_1^e) pair, we first compute s_3^e, $X^e = f_{(a,b,c)}(s_3^e) + s_2^e$ and the indexing set $\mathsf{I}_{(k_0^e, k_1^e)}$. Then, we use Theorem 1 and z-linear segment sets to compute partitions $\mathsf{P}(i, 8, \mathsf{K}_{(k_0^e, k_1^e)})$ (see steps 7-14 of Algorithm 2). Next, we do encryption for 12 rounds and check if s_{19}^e matches with s_7^d or not. If so, the corresponding key is one of the

Algorithm 1 Key recovery algorithm

Input : $\{(s_0^{e,0}, s_1^{e,0}), (s_0^{d,0}, s_1^{d,0})\}, \{(s_0^{e,1}, s_1^{e,1}), (s_0^{d,1}, s_1^{d,1})\}, \{(s_0^{e,2}, s_1^{e,2}), (s_0^{d,2}, s_1^{d,2})\}$
Output : secret key k
Procedure main :
    // Offline phase ▷ $T^{offline}$
    call procedure compute_zs
    call procedure construct_ds
    // Online phase ▷ T^{online}
    call procedure recover_sk

Procedure compute_zs : ▷ $T_0^{offline}$
    // Compute z-linear segment sets using Definition 4
    $n = 16$, $(a, b, c) = (8, 1, 2)$ / $(5, 0, 1)$
    return(Z, CL_z)

Procedure construct_ds : ▷ $T_1^{offline}$
    // Construct data structure for 6 decryption rounds
    $\mathsf{DS^d} = [\ [\ ]\ ]$
    $s_0^d = s_0^{d,0}$, $s_1^d = s_1^{d,0}$
    for $k_0^d = 0$ to $2^{16} - 1$ **do**
        for $k_1^d = 0$ to $2^{16} - 1$ **do**
            Compute $s_3^d, X^d, \mathsf{I}_{(k_0^d, k_1^d)}$
            DS.append($[\ s_3^d, X^d, \mathsf{I}_{(k_0^d, k_1^d)}\ ]$)
        end for
    end for
    return$(\mathsf{DS^d})$

Procedure recover_sk :
    // Filtering keys with Algorithm 2
    $K =$ filter_keys ▷ T_0^{online}
    // Exhaustive search on K using second plaintext-ciphertext pair
    for $\gamma \in K$ **do** ▷ T_1^{online}
        if encryption of $(s_0^{e,1}, s_1^{e,1})$ with γ equals $(s_0^{d,1}, s_1^{d,1})$ **do**
            K_1.append(γ)
        end if
    end for
    // Exhaustive search on K_1 using third plaintext-ciphertext pair
    for $\gamma \in K_1$ **do** ▷ T_2^{online}
        if encryption of $(s_0^{e,2}, s_1^{e,2})$ with γ equals $(s_0^{d,2}, s_1^{d,2})$ **do**
            K_2.append(γ)
        end if
    end for
    return(K_2) ▷ $K_2 = \{k\}$

correct key candidate. The number of computations of $f_{(a,b,c)}$ is then calculated as follows:

$$\underbrace{\frac{3 + N_z}{25}}_{\#\text{ computations of } f_{(a,b,c)} \text{ to get } \mathsf{I}_{(k_0^e, k_1^e)}} + \underbrace{\frac{N_z}{25}}_{\#\text{ computations of } f_{(a,b,c)} \text{ to get } \mathsf{P}(i, 8, \mathsf{K}_{(k_0^e, k_1^e)})} + 2^{32} \times \underbrace{\frac{12}{25}}_{\text{12-round encryption}}$$

$$= \frac{3+2N_z}{25} + 2^{32} \times \frac{12}{25}.$$

The time complexity (T_0^{online}) of filter_keys then equals $2^{32} \times (\frac{3+2N_z}{25} + 2^{32} \times \frac{12}{25}) \approx 2^{64} \times \frac{12}{25}$.

In Appendix B, we provide an example of computation of s_7^d from DS^{d}.

Algorithm 2 Extracting keys that maps (s_0^e, s_1^e) to (s_0^d, s_1^d)

$s_0^e = s_0^{e,0}$, $s_1^e = s_1^{e,0}$
$\mathsf{K} = []$
Procedure filter_keys :
  for $k_0^e = 0$ to $2^{16} - 1$ **do**
    for $k_1^e = 0$ to $2^{16} - 1$ **do**
      Compute s_3^e, $X^e = f_{(a,b,c)}(s_3^e) + s_2^e$ and $\mathsf{I}_{(k_0^e, k_1^e)}$
      for z in z-linear segment sets **do**
        for $x \in CL_z$ **do**
          for $i = 0$ to $|C_x|-1$
            $f = L^c(C_x[i]) + L^i(z)$
            for $j = 0$ to $2^{16} - 1$ **do**
              $k = (k_0^e, k_1^e, j, \mathsf{I}_{(k_0^e, k_1^e)}[j] + C_x[i])$
              $s_6^e = f + X^e + k_2^e + k_4^e$
              $s_7^e = s_3^e + k_5^e + C_x[i] + \mathsf{I}_{(k_0^e, k_1^e)}[s_6^e + X^e]$
              Encrypt (s_7^e, s_6^e) for 12 rounds and get s_{19}^e
              if s_{19}^e==compute_$s_7^d(k, \mathsf{DS}^{\mathsf{d}})$ **do**
                K.append(k)
              end if
            end for
          end for
        end for
      end for
    end for
  end for
  return(K)

Procedure compute_$s_7^d(k, \mathsf{DS}^{\mathsf{d}})$:
  // Compute 8-th element of sequence from decryption side (Theorem 1)
  $s_3^d = \mathsf{DS}^{\mathsf{d}}[L^{16}(k_0^d) + k_1^d][0]$
  $X^d = \mathsf{DS}^{\mathsf{d}}[L^{16}(k_0^d) + k_1^d][1]$
  $\mathsf{I}_{(k_0^d, k_1^d)} = \mathsf{DS}^{\mathsf{d}}[L^{16}(k_0^d) + k_1^d][2]$
  $p = \mathsf{I}_{(k_0^d, k_1^d)}[k_2^d + k_3^d]$
  $s_6^d = \mathsf{I}_{(k_0^d, k_1^d)}[p + X^d] + s_3^d + X^d + k_2^d + k_4^d$
  $s_7^d = s_3^d + k_5^d + p + \mathsf{I}_{(k_0^d, k_1^d)}[s_6^d + X^d]$
  return(s_7^d)

Time complexity of online phase. The time complexity of complete attack is dominated by T^{online} which is given by:

$$\begin{aligned} T^{online} &= T_0^{online} + T_1^{online} + T_2^{online} \\ &= 2^{64} \times \frac{12}{25} + 2^{48} + 2^{16} \approx 2^{62.94}. \end{aligned}$$

Remark 2. For the 24-round attack, the data and memory complexities are the same. However, the time complexity is $2^{64} \times \frac{11}{24} \approx 2^{62.87}$.

5.3 Experimental verification of attack

We have implemented the complete attack in Python and the source code is provided in Appendix C. We ran experiments for toy versions of both ciphers, i.e., with block-size/keysize, 8/16 and 16/32-bit. We find that the attack works for all keys, and for any 3 distinct random plaintext encrypted either by Simon-8/16 (Simon-16/32) or Simeck-8/16 (Simeck-16/32). Hence, a success probability of 1 proves that the similar results hold for Simon-32/64 and Simeck-32/64.

6 Key Recovery Attack on 27 Rounds

In this section, we show how to improve the key recovery attack presented in previous section by 2 rounds with the same complexities as of the 25-round attack. For a fixed partition $\mathsf{P}(i, 8, \mathsf{K}_{(k_0^e, k_1^e)})$, we incorporate the properties of key scheduling algorithms (KSA) and one round differentials and show that $\mathsf{P}(i, 9, \mathsf{K}_{(k_0^e, k_1^e)})$ can be computed from $\mathsf{P}(i, 8, \mathsf{K}_{(k_0^e, k_1^e)})$ by computing $f_{(a,b,c)}$ at most 2^{15} times. As a result, both forward and middle rounds can be extended by one round each, i.e., partial encryption starts from round 7 and matching is done at 20-th round. The results of the following two properties can be obtained directly by the definition of $\mathsf{P}(i, 8, \mathsf{K}_{(k_0^e, k_1^e)})$ and key scheduling algorithms. We present the main result of this section in Lemma 1.

Property 3 (Simon KSA and $\mathsf{P}(i, 8, \mathsf{K}_{(k_0^e, k_1^e)})$)**.** Let $n = 16$, $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be such that $F(x) = f_{(8,1,2)}(x + \Delta_y) + x + L^{n-1}(x) + L^{n-6}(y) + L^{n-8}(y)$, where $y = \mathsf{l}_{(k_0^e, k_1^e)}[x]$ and $\Delta_y = L^{n-3}(y) + L^{n-4}(y)$. Then $|Img(F(x))| \leq 2^{n-1}$.

Property 4 (Simeck KSA)**.** Let $n \geq 4$, $k_{i+4}^e = f_{(5,0,1)}(k_{i+1}^e) + k_i^e$ and $i \geq 0$. Then for a fixed (k_0^e, k_1^e) pair, k_4^e is constant for all $2^n \times 2^n$ values of k_2^e and k_3^e.

Property 5 (Differential [KLT15])**.** Let $n \geq 4$, $\Delta \in \mathbb{F}_2^n$ be fixed. Then

$$|Img(f_{(a,b,c)}(x) + f_{(a,b,c)}(x + \Delta))| \leq 2^{n-1}.$$

Lemma 1. *Given $n = 16$ and $(a, b, c) = (8, 1, 2)/(5, 0, 1)$. Then for all $(k, S_{(k,8)}) \in \mathsf{P}(i, 8, \mathsf{K}_{(k_0^e, k_1^e)})$, s_7^e can take at most 2^{n-1} values.*

Proof. Consider the value of s_7^e in the following cases:

- Case 1 : $(a, b, c) = (8, 1, 2)$

$$\begin{aligned} s_7^e &= f_{(8,1,2)}(s_6^e) + s_5^e + k_5^e = f_{(8,1,2)}(s_4^e + k_4^e + f_{(8,1,2)}(i)) + i + k_5^e \\ &= f_{(8,1,2)}(X^e + k_2^e + k_4^e + f_{(8,1,2)}(i)) + i + k_5^e,\ X^e = f_{(8,1,2)}(s_3^e) + s_2^e \\ &= f_{(8,1,2)}(C_0 + k_2^e + (L^{13}(k_3^e) + L^{12}(k_3^e)) + \\ &\quad C_1 + k_2^e + L^{15}(k_2^e) + L^{10}(k_3^e) + L^8(k_3^e) \text{ (Simon KSA)} \end{aligned}$$

  Here C_0 and C_1 are constants and given by:

$$\begin{aligned} C_0 &= X^e + f_{(8,1,2)}(i) + k_0^e + k_1^e + L^{15}(k_1^e) + Z_0 \\ C_1 &= i + Z_1 + L^{13}(Z_0) + L^{12}(Z_0) + L^{13}(k_0^e) + L^{12}(k_0^e) + \\ &\quad k_1^e + L^{13}(k_1^e) + L^{11}(k_1^e) \end{aligned}$$

  By Property 3, s_7^e can take at most 2^{n-1} values.

- Case 2 : $(a, b, c) = (5, 0, 1)$

$$\begin{aligned} s_7^e &= f_{(5,0,1)}(s_6^e) + s_5^e + k_5^e = f_{(5,0,1)}(s_4^e + k_4^e + f_{(5,0,1)}(i)) + i + k_5^e \\ &= f_{(5,0,1)}(X^e + k_2^e + k_4^e + f_{(5,0,1)}(i)) + i + k_5^e,\ X^e = f_{(5,0,1)}(s_3^e) + s_2^e \\ &= f_{(5,0,1)}(\Delta + k_2^e) + C_1 + f_{(5,0,1)}(k_2^e) \text{ (Property 4)} \end{aligned}$$

Similar to previous case, Δ and C_1 are constants and given by:

$$\begin{aligned} \Delta &= X^e + f_{(5,0,1)}(i) + k_0^e + f_{(5,0,1)}(k_1^e) + Z_0 \\ C_1 &= i + Z_1 + k_1^e \end{aligned}$$

The proof then follows from Property 5. □

From Lemma 1, we note that for each partition $\mathsf{P}(i, 8, \mathsf{K}_{(k_0^e, k_1^e)})$, s_7^e can take at most 2^{15} values. Accordingly, we only modify steps 11-18 of Algorithm 2. The partial encryption starts from (s_8^e, s_7^e) and s_{21}^e is then used for the matching. The modification is presented in Algorithm 3.

Algorithm 3 Modified algorithm for 27-round key recovery attack

```
TEMP_S6 = [ ]
TEMP_S7= [ ]
TEMP_Uniq_S7 = [ ]
for j = 0 to 2^16 − 1 do
        k = (k_0^e, k_1^e, j, l_(k_0^e,k_1^e)[j] + C_x[i])
        s_6^e = f + X^e + k_2^e + k_4^e
        s_7^e = s_3^e + k_5^e + C_x[i] + l_(k_0^e,k_1^e)[s_6^e + X^e]
        TEMP_S6.append(s_6^e)
        TEMP_S7.append(s_7^e)
end for

// Unique value of TEMP_S7
TEMP_Uniq_S7 = unique(TEMP_S7)

for u in TEMP_Uniq_S7 do
        t^e = f_(a,b,c)(u)
        // get_index finds indexes l such that TEMP_S7[l] = u
        Indices = get_index(TEMP_S7)
            for ind in Indices do
                k = (k_0^e, k_1^e, ind, l_(k_0^e,k_1^e)[ind] + C_x[i])
                s_8^e = t^e + k_6^e+TEMP_S6[ind]
                Encrypt (s_8^e, u) for 13 rounds and get s_21^e
                if s_21^e==compute_s_7^d(k, DS^d) do
                    K.append(k)
                end if
            end for
end for
```

Attack complexities. The data and memory complexities are same as 25-round attack.

The time complexity is given by:

$$\begin{aligned} T^{online} &= 2^{32} \times T_0^{online} + T_1^{online} + T_2^{online} \\ &\approx 2^{32}(\frac{3+2N_z}{27} + 2^{31} \times \frac{1}{27} + 2^{32} \times \frac{13}{27}) + 2^{48} + 2^{16} \\ &\approx 2^{64} \times \frac{13}{27} \approx 2^{62.94}. \end{aligned}$$

Remark 3. The complexities of 26-round attack are calculated accordingly.

7 Concluding Remarks

In this work, we have introduced a new property of block ciphers called correlated sequences and demonstrated its application in a meet-in-the-middle attack. As a result, we presented a $2t-3+l$ (resp. $2t+l$)-round attack for Feistel (resp. SPN) ciphers with t length correlated sequences and l rounds of partial encryption. We have applied our technique on two lightweight block ciphers Simon-32/64 and Simeck-32/64 and presented the first 24, 25, 26, 27-round attack on these ciphers with data and memory complexities of 3 and 2^{49} bytes, respectively. The time complexities are $2^{62.87}$(resp. $2^{62.94}$) for 24, 26 (resp. 25, 27)-round attacks.

Future work. We observe that number of attacked rounds depends on the length of correlated sequences. Thus, an immediate question is how to extend the length of correlated sequences of Simon-32/64 and Simeck-32/64 and improve the presented attack. It should be noted that correlated sequences has similar applications to other variants of Simon and Simeck as well. In addition, investigating the generic techniques or the underlying ciphers' structure to construct correlated sequences is an interesting future problem. Furthermore, in our analysis of Simon-like ciphers, we exploited the linear segment set properties of the nonlinear round function. Thus, our attack has similar applications to constructions (for e.g., Keccak [BDPVA09] 5-bit Sbox and Rasta's [DEG+18] substitution layer) where the nonlinear function preserves such properties. The recently proposed lightweight permutations sLiSCP and sLiSCP-light [ARH+17, ARH+18] use Simeck's round function as their underlying nonlinear component. Hence, our attack could be used as a starting point to analyze these permutations further.

References

[ALLW14] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential cryptanalysis of round-reduced simon and speck. In: Cid C., Rechberger C. (eds.), FSE 2014. LNCS, vol. 8540, pp. 525–545. Springer, Heidelberg (2014)

[ARH+17] Riham AlTawy, Raghvendra Rohit, Morgan He, Kalikinkar Mandal, Gangqiang Yang, and Guang Gong. sliscp: Simeck-based permutations for lightweight sponge cryptographic primitives. In: Adams C., Camenisch J. (eds.), SAC 2017. LNCS, vol. 10719, pp. 129–150. Springer, Cham (2017)

[ARH+18] Riham Altawy, Raghvendra Rohit, Morgan He, Kalikinkar Mandal, Gangqiang Yang, and Guang Gong. sliscp-light: Towards hardware optimized sponge-specific cryptographic permutations. ACM Transactions on Embedded Computing Systems (TECS), 17(4):81, 2018.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Stern J. (eds.), Eurocrypt 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)

[BDPVA09] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Keccak specifications. Submission to NIST (Round 2), 2009.

[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: M. Robshaw and J. Katz (eds.), CRYPTO 2016. LNCS, vol. 9815, pp. 123–153. Springer, Heidelberg (2016)

[BKL+07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In: P. Paillier and I. Verbauwhede (eds.), CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift: A small present. In: Fischer W., Homma N. (eds), CHES 2017. LNCS, vol. 10529, pp. 321–345. Springer, Cham (2017)

[BR11] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Cryptology ePrint Archive, Report 2011/123, 2011. https://eprint.iacr.org/2011/123.pdf.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. Journal of CRYPTOLOGY, 4(1), pp. 3–72, 1991.

[BTCS+13] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.

[CCW+18] Zhihui Chu, Huaifeng Chen, Xiaoyun Wang, Xiaoyang Dong, and Lu Li. Improved integral attacks on simon32 and simon48 with dynamic key-guessing techniques. Security and Communication Networks, 2018. https://doi.org/10.1155/2018/5160237.

[CW16] Huaifeng Chen and Xiaoyun Wang. Improved linear hull attack on round-reduced simon with dynamic key-guessing techniques. In: Peyrin T. (eds), FSE 2016. LNCS, vol. 9783, pp. 428–449. Springer, Heidelberg (2016)

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low and depth and few ands per bit. In: hacham H., Boldyreva A. (eds), Crypto 2018. LNCS, vol. 10991, pp. 662–692. Springer, Cham (2018)

[DES77] FIPS 46, Data Encryption Standard, US Department of Commerce/National Bureau of Standards, National Technical Information Service, Springerfield, Virginia, 1977

[DF16] Patrick Derbez and Pierre-Alain Fouque. Automatic search of meet-in-the-middle and impossible differential attacks. In: Robshaw M., Katz J. (eds), CRYPTO 2016. LNCS, vol. 9815, pp. 157–184. Springer, Heidelberg (2016)

[DH77] Whitfield Diffie and Martin E Hellman. Special feature exhaustive cryptanalysis of the nbs data encryption standard. Computer 10(6), pp. 74–84. (1977)

[DR98] Joan Daemen and Vincent Rijmen. The design of Rijndael: AES - The advanced encryption standard. 1998. Available at https://www.nist.gov/publications/advanced-encryption-standard-aes

[FSW16] Kai Fu, Ling Sun, and Meiqin Wang. New integral attacks on simon. IET Information Security 11(5), pp. 277–286. (2016)

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The led block cipher. In: B. Preneel and T. Takagi (eds.), CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)

[HSH+06] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bon-Seok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, et al. Hight: A new block cipher suitable for low-resource device. In: Goubin L., Matsui M (eds.), CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the simon block cipher family. In: R. Gennaro and M. Robshaw (eds.), CRYPTO 2015. LNCS, vol. 9215, pp. 258–269. Springer, Heidelberg (2015)

[KR16] Stefan Kölbl and Arnab Roy. A brief comparison of simon and simeck. In: Bogdanov A. (eds), LightSec 2016. LNCS, vol. 10098, pp. 69–88. Springer, Cham (2016)

[KSI16] Kota Kondo, Yu Sasaki, and Tetsu Iwata. On the design rationale of simon block cipher: Integral attacks and impossible differential attacks against simon variants. In: Manulis M., Sadeghi AR., Schneider S. (eds), ACNS 2016. LNCS, vol. 9696, pp. 518–536. Springer, Cham (2016)

[KW02] Lars Knudsen and David Wagner. Integral cryptanalysis. In: Daemen J., Rijmen V. (eds), FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002)

[LLW17] Zhengbin Liu, Yongqiang Li, and Mingsheng Wang. Optimal differential trails in simon-like ciphers. IACR Transactions on Symmetric Cryptology 2017 1, pp. 358–379. (2017) http://dx.doi.org/10.13154/tosc.v2017.i1.358-379

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In: Helleseth T. (eds), Eurocrypt 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1993)

[QCW16] Lingyue Qin, Huaifeng Chen, and Xiaoyun Wang. Linear hull attack on round-reduced simeck with dynamic key-guessing techniques. In: Liu J., Steinfeld R. (eds) Information Security and Privacy, ACISP 2016. LNCS, vol. 9723, pp. 409–424. Springer, Cham (2016)

[QHS16] Kexin Qiao, Lei Hu, and Siwei Sun. Differential analysis on simeck and simon with dynamic key-guessing techniques. In: Camp O., Furnell S., Mori P. (eds), ICISSP 2016. LNCS, vol. 691, pp. 64–85. Springer, Cham (2016)

[SFW15] Ling Sun, Kai Fu, and Meiqin Wang. Improved zero-correlation cryptanalysis on simon. In: Lin D., Wang X., Yung M. (eds), Inscrypt 2015. LNCS, vol. 9589, pp. 125–143. Springer, Cham (2014)

[SHMS14] Ling Song, Lei Hu, Bingke Ma, and Danping Shi. Match box meet-in-the-middle attacks on the simon family of block ciphers. In: Eisenbarth T., Öztürk E. (eds), LightSec 2014. LNCS, vol. 8898, pp. 140–151. Springer, Cham (2014)

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: an ultra-lightweight blockcipher. In: Preneel B., Takagi T. (eds), CHES 2011. LNCS, vol. 6917, pp. 342–357. Springer, Heidelberg (2011)

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In: Peyrin T. (eds), FSE 2016. LNCS, vol. 9783, pp. 357-377. Springer, Heidelberg (2016)

[WLV+14] Qingju Wang, Zhiqiang Liu, Kerem Varıcı, Yu Sasaki, Vincent Rijmen, and Yosuke Todo. Cryptanalysis of reduced-round simon32 and simon48. In: Meier W., Mukhopadhyay D. (eds), Indocrypt 2014. LNCS, vol. 8885, pp. 143–160. Springer, Cham (2014)

[WWJZ14] Ning Wang, Xiaoyun Wang, Keting Jia, and Jingyuan Zhao. Differential attacks on reduced simon versions with dynamic key-guessing techniques. Cryptology ePrint Archive, Report 2014/448, 2014. https://eprint.iacr.org/2014/448.pdf.

[YZS+15] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D Aagaard, and Guang Gong. The simeck family of lightweight block ciphers. In: T. Güneysu and H. Handschuh (Eds.), CHES 2015. LNCS, vol. 9293, pp. 307–329. Springer, Heidelberg (2015)

[ZGHL16] Kai Zhang, Jie Guan, Bin Hu, and Dongdai Lin. Integral cryptanalysis on simeck. ICIST 2016. IEEE, pp. 216–222. (2016)

[ZGHL17] Kai Zhang, Jie Guan, Bin Hu, and Dongdai Lin. Security evaluation on simeck against zero-correlation linear cryptanalysis. IET Information Security 12(1), pp. 87–93. (2018)

A Example: Data Structure

Consider a toy Simon-8/16 cipher as given in Example 1. Let $k = (1, 2, 3, 4)$ and $s_0^e = 15$, $s_1^e = 14$, then $s_0^d = 5$ and $s_1^d = 11$. In Table 7, we provide the data structure DS^d that is used for 6 decryption rounds.

Table 7: Data structure for toy Simon

i	$\mathsf{DS}^d[i]$
0	[9, 15, [9, 14, 7, 2, 4, 3, 14, 11, 2, 4, 12, 8, 7, 1, 13, 9]]
1	[8, 10, [2, 5, 10, 15, 15, 8, 3, 6, 0, 6, 8, 12, 5, 3, 9, 13]]
2	[11, 5, [14, 10, 0, 6, 15, 11, 5, 3, 5, 0, 11, 12, 12, 9, 6, 1]]
3	[10, 2, [2, 4, 10, 14, 7, 1, 11, 15, 0, 7, 8, 13, 13, 10, 1, 4]]
4	[13, 6, [0, 6, 12, 8, 5, 3, 13, 9, 10, 13, 6, 3, 7, 0, 15, 10]]
5	[12, 3, [2, 4, 8, 12, 7, 1, 9, 13, 1, 6, 11, 14, 12, 11, 2, 7]]
6	[15, 8, [13, 8, 5, 2, 4, 1, 8, 15, 15, 11, 7, 1, 14, 10, 2, 4]]
7	[14, 15, [14, 9, 0, 5, 3, 4, 9, 12, 5, 3, 11, 15, 0, 6, 10, 14]]
8	[1, 12, [10, 15, 6, 1, 3, 6, 11, 12, 0, 4, 12, 10, 1, 5, 9, 15]]
9	[0, 8, [2, 7, 10, 13, 11, 14, 7, 0, 0, 4, 8, 14, 1, 5, 13, 11]]
10	[3, 6, [14, 8, 2, 6, 11, 13, 3, 7, 4, 3, 8, 13, 9, 14, 1, 4]]
11	[2, 0, [2, 6, 10, 12, 3, 7, 15, 9, 0, 5, 8, 15, 9, 12, 5, 2]]
12	[5, 13, [11, 14, 5, 2, 2, 7, 8, 15, 0, 4, 14, 8, 1, 5, 11, 13]]
13	[4, 9, [3, 6, 9, 14, 10, 15, 4, 3, 0, 4, 10, 12, 1, 5, 15, 9]]
14	[7, 3, [9, 15, 3, 7, 12, 10, 2, 6, 10, 13, 0, 5, 7, 0, 9, 12]]
15	[6, 5, [3, 7, 13, 11, 2, 6, 8, 14, 8, 13, 6, 1, 1, 4, 11, 12]]
16	[12, 2, [4, 2, 12, 8, 1, 7, 13, 9, 6, 1, 14, 11, 11, 12, 7, 2]]
17	[13, 7, [6, 0, 8, 12, 3, 5, 9, 13, 13, 10, 3, 6, 0, 7, 10, 15]]
18	[14, 14, [9, 14, 5, 0, 4, 3, 12, 9, 3, 5, 15, 11, 6, 0, 14, 10]]
19	[15, 9, [8, 13, 2, 5, 1, 4, 15, 8, 11, 15, 1, 7, 10, 14, 4, 2]]
20	[8, 11, [5, 2, 15, 10, 8, 15, 6, 3, 6, 0, 12, 8, 3, 5, 13, 9]]
21	[9, 14, [14, 9, 2, 7, 3, 4, 11, 14, 4, 2, 8, 12, 1, 7, 9, 13]]
22	[10, 3, [4, 2, 14, 10, 1, 7, 15, 11, 7, 0, 13, 8, 10, 13, 4, 1]]
23	[11, 4, [10, 14, 6, 0, 11, 15, 3, 5, 0, 5, 12, 11, 9, 12, 1, 6]]
24	[4, 8, [6, 3, 14, 9, 15, 10, 3, 4, 4, 0, 12, 10, 5, 1, 9, 15]]
25	[5, 12, [14, 11, 2, 5, 7, 2, 15, 8, 4, 0, 8, 14, 5, 1, 13, 11]]
26	[6, 4, [7, 3, 11, 13, 6, 2, 14, 8, 13, 8, 1, 6, 4, 1, 12, 11]]
27	[7, 2, [15, 9, 7, 3, 10, 12, 6, 2, 13, 10, 5, 0, 0, 7, 12, 9]]
28	[0, 9, [7, 2, 13, 10, 14, 11, 0, 7, 4, 0, 14, 8, 5, 1, 11, 13]]
29	[1, 13, [15, 10, 1, 6, 6, 3, 12, 11, 4, 0, 10, 12, 5, 1, 15, 9]]
30	[2, 1, [6, 2, 12, 10, 7, 3, 9, 15, 5, 0, 15, 8, 12, 9, 2, 5]]
31	[3, 7, [8, 14, 6, 2, 13, 11, 7, 3, 3, 4, 13, 8, 14, 9, 4, 1]]
32	[1, 14, [6, 1, 10, 15, 11, 12, 3, 6, 12, 10, 0, 4, 9, 15, 1, 5]]
33	[0, 10, [10, 13, 2, 7, 7, 0, 11, 14, 8, 14, 0, 4, 13, 11, 1, 5]]
34	[3, 4, [2, 6, 14, 8, 3, 7, 11, 13, 8, 13, 4, 3, 1, 4, 9, 14]]
35	[2, 2, [10, 12, 2, 6, 15, 9, 3, 7, 8, 15, 0, 5, 5, 2, 9, 12]]
36	[5, 15, [5, 2, 11, 14, 8, 15, 2, 7, 14, 8, 0, 4, 11, 13, 1, 5]]
37	[4, 11, [9, 14, 3, 6, 4, 3, 10, 15, 10, 12, 0, 4, 15, 9, 1, 5]]
38	[7, 1, [3, 7, 9, 15, 2, 6, 12, 10, 0, 5, 10, 13, 9, 12, 7, 0]]
39	[6, 7, [13, 11, 3, 7, 8, 14, 2, 6, 6, 1, 8, 13, 11, 12, 1, 4]]
40	[9, 13, [7, 2, 9, 14, 14, 11, 4, 3, 12, 8, 2, 4, 13, 9, 7, 1]]
41	[8, 8, [10, 15, 2, 5, 3, 6, 15, 8, 8, 12, 0, 6, 9, 13, 5, 3]]
42	[11, 7, [0, 6, 14, 10, 5, 3, 15, 11, 11, 12, 5, 0, 6, 1, 12, 9]]
43	[10, 0, [10, 14, 2, 4, 11, 15, 7, 1, 8, 13, 0, 7, 1, 4, 13, 10]]
44	[13, 4, [12, 8, 0, 6, 13, 9, 5, 3, 6, 3, 10, 13, 15, 10, 7, 0]]
45	[12, 1, [8, 12, 2, 4, 9, 13, 7, 1, 11, 14, 1, 6, 2, 7, 12, 11]]
46	[15, 10, [5, 2, 13, 8, 8, 15, 4, 1, 7, 1, 15, 11, 2, 4, 14, 10]]
47	[14, 13, [0, 5, 14, 9, 9, 12, 3, 4, 11, 15, 5, 3, 10, 14, 0, 6]]
48	[6, 6, [11, 13, 7, 3, 14, 8, 6, 2, 1, 6, 13, 8, 12, 11, 4, 1]]
49	[7, 0, [7, 3, 15, 9, 6, 2, 10, 12, 5, 0, 13, 10, 12, 9, 0, 7]]
50	[4, 10, [14, 9, 6, 3, 3, 4, 15, 10, 12, 10, 4, 0, 9, 15, 5, 1]]
51	[5, 14, [2, 5, 14, 11, 15, 8, 7, 2, 8, 14, 4, 0, 13, 11, 5, 1]]
52	[2, 3, [12, 10, 6, 2, 9, 15, 7, 3, 15, 8, 5, 0, 2, 5, 12, 9]]
53	[3, 5, [6, 2, 8, 14, 7, 3, 13, 11, 13, 8, 3, 4, 4, 1, 14, 9]]
54	[0, 11, [13, 10, 7, 2, 0, 7, 14, 11, 14, 8, 4, 0, 11, 13, 5, 1]]
55	[1, 15, [1, 6, 15, 10, 12, 11, 6, 3, 10, 12, 4, 0, 15, 9, 5, 1]]
56	[14, 12, [5, 0, 9, 14, 12, 9, 4, 3, 15, 11, 3, 5, 14, 10, 6, 0]]
57	[15, 11, [2, 5, 8, 13, 15, 8, 1, 4, 1, 7, 11, 15, 4, 2, 10, 14]]
58	[12, 0, [12, 8, 4, 2, 13, 9, 1, 7, 14, 11, 6, 1, 7, 2, 11, 12]]
59	[13, 5, [8, 12, 6, 0, 9, 13, 3, 5, 3, 6, 13, 10, 10, 15, 0, 7]]
60	[10, 1, [14, 10, 4, 2, 15, 11, 1, 7, 13, 8, 7, 0, 4, 1, 10, 13]]
61	[11, 6, [6, 0, 10, 14, 3, 5, 11, 15, 12, 11, 0, 5, 1, 6, 9, 12]]
62	[8, 9, [15, 10, 5, 2, 6, 3, 8, 15, 12, 8, 6, 0, 13, 9, 3, 5]]
63	[9, 12, [2, 7, 14, 9, 11, 14, 3, 4, 8, 12, 4, 2, 9, 13, 1, 7]]
64	[0, 12, [11, 14, 7, 0, 2, 7, 10, 13, 1, 5, 13, 11, 0, 4, 8, 14]]
65	[1, 8, [3, 6, 11, 12, 10, 15, 6, 1, 1, 5, 9, 15, 0, 4, 12, 10]]
66	[2, 4, [3, 7, 15, 9, 2, 6, 10, 12, 9, 12, 5, 2, 0, 5, 8, 15]]
67	[3, 2, [11, 13, 3, 7, 14, 8, 2, 6, 9, 14, 1, 4, 4, 3, 8, 13]]
68	[4, 13, [10, 15, 4, 3, 3, 6, 9, 14, 1, 5, 15, 9, 0, 4, 10, 12]]
69	[5, 9, [2, 7, 8, 15, 11, 14, 5, 2, 1, 5, 11, 13, 0, 4, 14, 8]]
70	[6, 1, [2, 6, 8, 14, 3, 7, 13, 11, 1, 4, 11, 12, 8, 13, 6, 1]]
71	[7, 7, [12, 10, 2, 6, 9, 15, 3, 7, 7, 0, 9, 12, 10, 13, 0, 5]]
72	[8, 14, [15, 8, 3, 6, 2, 5, 10, 15, 5, 3, 9, 13, 0, 6, 8, 12]]
73	[9, 11, [4, 3, 14, 11, 9, 14, 7, 2, 7, 1, 13, 9, 2, 4, 12, 8]]
74	[10, 6, [7, 1, 11, 15, 2, 4, 10, 14, 13, 10, 1, 4, 0, 7, 8, 13]]
75	[11, 1, [15, 11, 5, 3, 14, 10, 0, 6, 12, 9, 6, 1, 5, 0, 11, 12]]
76	[12, 7, [7, 1, 9, 13, 2, 4, 8, 12, 12, 11, 2, 7, 1, 6, 11, 14]]
77	[13, 2, [5, 3, 13, 9, 0, 6, 12, 8, 7, 0, 15, 10, 10, 13, 6, 3]]
78	[14, 11, [3, 4, 9, 12, 14, 9, 0, 5, 0, 6, 10, 14, 5, 3, 11, 15]]
79	[15, 12, [4, 1, 8, 15, 13, 8, 5, 2, 14, 10, 2, 4, 15, 11, 7, 1]]
80	[5, 8, [7, 2, 15, 8, 14, 11, 2, 5, 5, 1, 13, 11, 4, 0, 8, 14]]
81	[4, 12, [15, 10, 3, 4, 6, 3, 14, 9, 5, 1, 9, 15, 4, 0, 12, 10]]
82	[7, 6, [10, 12, 6, 2, 15, 9, 7, 3, 0, 7, 12, 9, 13, 10, 5, 0]]
83	[6, 0, [6, 2, 14, 8, 7, 3, 11, 13, 4, 1, 12, 11, 13, 8, 1, 6]]
84	[1, 9, [6, 3, 12, 11, 15, 10, 1, 6, 5, 1, 15, 9, 4, 0, 10, 12]]
85	[0, 13, [14, 11, 0, 7, 7, 2, 13, 10, 5, 1, 11, 13, 4, 0, 14, 8]]
86	[3, 3, [13, 11, 7, 3, 8, 14, 6, 2, 14, 9, 4, 1, 3, 4, 13, 8]]
87	[2, 5, [7, 3, 9, 15, 6, 2, 12, 10, 12, 9, 2, 5, 5, 0, 15, 8]]
88	[13, 3, [3, 5, 9, 13, 6, 0, 8, 12, 0, 7, 10, 15, 13, 10, 3, 6]]
89	[12, 6, [1, 7, 13, 9, 4, 2, 12, 8, 11, 12, 7, 2, 6, 1, 14, 11]]
90	[15, 13, [1, 4, 15, 8, 8, 13, 2, 5, 10, 14, 4, 2, 11, 15, 1, 7]]
91	[14, 10, [4, 3, 12, 9, 9, 14, 5, 0, 6, 0, 14, 10, 3, 5, 15, 11]]
92	[9, 10, [3, 4, 11, 14, 14, 9, 2, 7, 1, 7, 9, 13, 4, 2, 8, 12]]
93	[8, 15, [8, 15, 6, 3, 5, 2, 15, 10, 3, 5, 13, 9, 6, 0, 12, 8]]
94	[11, 0, [11, 15, 3, 5, 10, 14, 6, 0, 9, 12, 1, 6, 0, 5, 12, 11]]
95	[10, 7, [1, 7, 15, 11, 4, 2, 14, 10, 10, 13, 4, 1, 7, 0, 13, 8]]
96	[12, 5, [9, 13, 7, 1, 8, 12, 2, 4, 2, 7, 12, 11, 11, 14, 1, 6]]
97	[13, 0, [13, 9, 5, 3, 12, 8, 0, 6, 15, 10, 7, 0, 6, 3, 10, 13]]
98	[14, 9, [9, 12, 3, 4, 0, 5, 14, 9, 10, 14, 0, 6, 11, 15, 5, 3]]
99	[15, 14, [8, 15, 4, 1, 5, 2, 13, 8, 2, 4, 14, 10, 7, 1, 15, 11]]
100	[8, 12, [3, 6, 15, 8, 10, 15, 2, 5, 9, 13, 5, 3, 8, 12, 0, 6]]
101	[9, 9, [14, 11, 4, 3, 7, 2, 9, 14, 13, 9, 7, 1, 12, 8, 2, 4]]
102	[10, 4, [11, 15, 7, 1, 10, 14, 2, 4, 1, 4, 13, 10, 8, 13, 0, 7]]
103	[11, 3, [5, 3, 15, 11, 0, 6, 14, 10, 6, 1, 12, 9, 11, 12, 5, 0]]
104	[4, 15, [4, 3, 10, 15, 9, 14, 3, 6, 15, 9, 1, 5, 10, 12, 0, 4]]
105	[5, 11, [8, 15, 2, 7, 5, 2, 11, 14, 11, 13, 1, 5, 14, 8, 0, 4]]
106	[6, 3, [8, 14, 2, 6, 13, 11, 3, 7, 11, 12, 1, 4, 6, 1, 8, 13]]
107	[7, 5, [2, 6, 12, 10, 3, 7, 9, 15, 9, 12, 7, 0, 0, 5, 10, 13]]
108	[0, 14, [7, 0, 11, 14, 10, 13, 2, 7, 13, 11, 1, 5, 8, 14, 0, 4]]
109	[1, 10, [11, 12, 3, 6, 6, 1, 10, 15, 9, 15, 1, 5, 12, 10, 0, 4]]
110	[2, 6, [15, 9, 3, 7, 10, 12, 2, 6, 5, 2, 9, 12, 8, 15, 0, 5]]
111	[3, 0, [3, 7, 11, 13, 2, 6, 14, 8, 1, 4, 9, 14, 8, 13, 4, 3]]
112	[11, 2, [3, 5, 11, 15, 6, 0, 10, 14, 1, 6, 9, 12, 12, 11, 0, 5]]
113	[10, 5, [15, 11, 1, 7, 14, 10, 4, 2, 4, 1, 10, 13, 13, 8, 7, 0]]
114	[9, 8, [11, 14, 3, 4, 2, 7, 14, 9, 9, 13, 1, 7, 8, 12, 4, 2]]
115	[8, 13, [6, 3, 8, 15, 15, 10, 5, 2, 13, 9, 3, 5, 12, 8, 6, 0]]
116	[15, 15, [15, 8, 1, 4, 2, 5, 8, 13, 4, 2, 10, 14, 1, 7, 11, 15]]
117	[14, 8, [12, 9, 4, 3, 5, 0, 9, 14, 14, 10, 6, 0, 15, 11, 3, 5]]
118	[13, 1, [9, 13, 3, 5, 8, 12, 6, 0, 10, 15, 0, 7, 3, 6, 13, 10]]
119	[12, 4, [13, 9, 1, 7, 12, 8, 4, 2, 7, 2, 11, 12, 14, 11, 6, 1]]
120	[3, 1, [7, 3, 13, 11, 6, 2, 8, 14, 4, 1, 14, 9, 13, 8, 3, 4]]
121	[2, 7, [9, 15, 7, 3, 12, 10, 6, 2, 2, 5, 12, 9, 15, 8, 5, 0]]
122	[1, 11, [12, 11, 6, 3, 1, 6, 15, 10, 15, 9, 5, 1, 10, 12, 4, 0]]
123	[0, 15, [0, 7, 14, 11, 13, 10, 7, 2, 11, 13, 5, 1, 14, 8, 4, 0]]
124	[7, 4, [6, 2, 10, 12, 7, 3, 15, 9, 12, 9, 0, 7, 5, 0, 13, 10]]
125	[6, 2, [14, 8, 6, 2, 11, 13, 7, 3, 12, 11, 4, 1, 1, 6, 13, 8]]
126	[5, 10, [15, 8, 7, 2, 2, 5, 14, 11, 13, 11, 5, 1, 8, 14, 4, 0]]
127	[4, 14, [3, 4, 15, 10, 14, 9, 6, 3, 9, 15, 5, 1, 12, 10, 4, 0]]
128	[11, 13, [5, 0, 11, 12, 12, 9, 6, 1, 14, 10, 0, 6, 15, 11, 5, 3]]

i	$\mathsf{DS}^d[i]$
129	[10, 10, [0, 7, 8, 13, 13, 10, 1, 4, 2, 4, 10, 14, 7, 1, 11, 15]]
130	[9, 7, [2, 4, 12, 8, 7, 1, 13, 9, 9, 14, 7, 2, 4, 3, 14, 11]]
131	[8, 2, [0, 6, 8, 12, 5, 3, 9, 13, 2, 5, 10, 15, 15, 8, 3, 6]]
132	[15, 0, [15, 11, 7, 1, 14, 10, 2, 4, 13, 8, 5, 2, 4, 1, 8, 15]]
133	[14, 7, [5, 3, 11, 15, 0, 6, 10, 14, 14, 9, 0, 5, 3, 4, 9, 12]]
134	[13, 14, [10, 13, 6, 3, 7, 0, 15, 10, 0, 6, 12, 8, 5, 3, 13, 9]]
135	[12, 11, [1, 6, 11, 14, 12, 11, 2, 7, 2, 4, 8, 12, 7, 1, 9, 13]]
136	[3, 14, [4, 3, 8, 13, 9, 14, 1, 4, 14, 8, 2, 6, 11, 13, 3, 7]]
137	[2, 8, [0, 5, 8, 15, 9, 12, 5, 2, 2, 6, 10, 12, 3, 7, 15, 9]]
138	[1, 4, [0, 4, 12, 10, 1, 5, 9, 15, 10, 15, 6, 1, 3, 6, 11, 12]]
139	[0, 0, [0, 4, 8, 14, 1, 5, 13, 11, 2, 7, 10, 13, 11, 14, 7, 0]]
140	[7, 11, [10, 13, 0, 5, 7, 0, 9, 12, 9, 15, 3, 7, 12, 10, 2, 6]]
141	[6, 13, [8, 13, 6, 1, 1, 4, 11, 12, 3, 7, 13, 11, 2, 6, 8, 14]]
142	[5, 5, [0, 4, 14, 8, 1, 5, 11, 13, 11, 14, 5, 2, 2, 7, 8, 15]]
143	[4, 1, [0, 4, 10, 12, 1, 5, 15, 9, 3, 6, 9, 14, 10, 15, 4, 3]]
144	[15, 1, [11, 15, 1, 7, 10, 14, 4, 2, 8, 13, 2, 5, 1, 4, 15, 8]]
145	[14, 6, [3, 5, 15, 11, 6, 0, 14, 10, 9, 14, 5, 0, 4, 3, 12, 9]]
146	[13, 15, [13, 10, 3, 6, 0, 7, 10, 15, 6, 0, 8, 12, 3, 5, 9, 13]]
147	[12, 10, [6, 1, 14, 11, 11, 12, 7, 2, 4, 2, 12, 8, 1, 7, 13, 9]]
148	[11, 12, [0, 5, 12, 11, 9, 12, 1, 6, 10, 14, 6, 0, 11, 15, 3, 5]]
149	[10, 11, [7, 0, 13, 8, 10, 13, 4, 1, 4, 2, 14, 10, 1, 7, 15, 11]]
150	[9, 6, [4, 2, 8, 12, 1, 7, 9, 13, 14, 9, 2, 7, 3, 4, 11, 14]]
151	[8, 3, [6, 0, 12, 8, 3, 5, 13, 9, 5, 2, 15, 10, 8, 15, 6, 3]]
152	[7, 10, [13, 10, 5, 0, 0, 7, 12, 9, 15, 9, 7, 3, 10, 12, 6, 2]]
153	[6, 12, [13, 8, 1, 6, 4, 1, 12, 11, 7, 3, 11, 13, 6, 2, 14, 8]]
154	[5, 4, [4, 0, 8, 14, 5, 1, 13, 11, 14, 11, 2, 5, 7, 2, 15, 8]]
155	[4, 0, [4, 0, 12, 10, 5, 1, 9, 15, 6, 3, 14, 9, 15, 10, 3, 4]]
156	[3, 15, [3, 4, 13, 8, 14, 9, 4, 1, 8, 14, 6, 2, 13, 11, 7, 3]]
157	[2, 9, [5, 0, 15, 8, 12, 9, 2, 5, 6, 2, 12, 10, 7, 3, 9, 15]]
158	[1, 5, [4, 0, 10, 12, 5, 1, 15, 9, 15, 10, 1, 6, 6, 3, 12, 11]]
159	[0, 1, [4, 0, 14, 8, 5, 1, 11, 13, 7, 2, 13, 10, 14, 11, 0, 7]]
160	[3, 12, [8, 13, 4, 3, 1, 4, 9, 14, 2, 6, 14, 8, 3, 7, 11, 13]]
161	[2, 10, [8, 15, 0, 5, 5, 2, 9, 12, 10, 12, 2, 6, 15, 9, 3, 7]]
162	[1, 6, [12, 10, 0, 4, 9, 15, 1, 5, 6, 1, 10, 15, 11, 12, 3, 6]]
163	[0, 2, [8, 14, 0, 4, 13, 11, 1, 5, 10, 13, 2, 7, 7, 0, 11, 14]]
164	[7, 9, [0, 5, 10, 13, 9, 12, 7, 0, 3, 7, 9, 15, 2, 6, 12, 10]]
165	[6, 15, [6, 1, 8, 13, 11, 12, 1, 4, 13, 11, 3, 7, 8, 14, 2, 6]]
166	[5, 7, [14, 8, 0, 4, 11, 13, 1, 5, 5, 2, 11, 14, 8, 15, 2, 7]]
167	[4, 3, [10, 12, 0, 4, 15, 9, 1, 5, 9, 14, 3, 6, 4, 3, 10, 15]]
168	[11, 15, [11, 12, 5, 0, 6, 1, 12, 9, 0, 6, 14, 10, 5, 3, 15, 11]]
169	[10, 8, [8, 13, 0, 7, 1, 4, 13, 10, 10, 14, 2, 4, 11, 15, 7, 1]]
170	[9, 5, [12, 8, 2, 4, 13, 9, 7, 1, 7, 2, 9, 14, 14, 11, 4, 3]]
171	[8, 0, [8, 12, 0, 6, 9, 13, 5, 3, 10, 15, 2, 5, 3, 6, 15, 8]]
172	[15, 2, [7, 1, 15, 11, 2, 4, 14, 10, 5, 2, 13, 8, 8, 15, 4, 1]]
173	[14, 5, [11, 15, 5, 3, 10, 14, 0, 6, 0, 5, 14, 9, 9, 12, 3, 4]]
174	[13, 12, [6, 3, 10, 13, 15, 10, 7, 0, 12, 8, 0, 6, 13, 9, 5, 3]]
175	[12, 9, [11, 14, 1, 6, 2, 7, 12, 11, 8, 12, 2, 4, 9, 13, 7, 1]]
176	[5, 6, [8, 14, 4, 0, 13, 11, 5, 1, 2, 5, 14, 11, 15, 8, 7, 2]]
177	[4, 2, [12, 10, 4, 0, 9, 15, 5, 1, 14, 9, 6, 3, 3, 4, 15, 10]]
178	[7, 8, [5, 0, 13, 10, 12, 9, 0, 7, 7, 3, 15, 9, 6, 2, 10, 12]]
179	[6, 14, [1, 6, 13, 8, 12, 11, 4, 1, 11, 13, 7, 3, 14, 8, 6, 2]]
180	[1, 7, [10, 12, 4, 0, 15, 9, 5, 1, 1, 6, 15, 10, 12, 11, 6, 3]]
181	[0, 3, [14, 8, 4, 0, 11, 13, 5, 1, 13, 10, 7, 2, 0, 7, 14, 11]]
182	[3, 13, [13, 8, 3, 4, 4, 1, 14, 9, 6, 2, 8, 14, 7, 3, 13, 11]]
183	[2, 11, [15, 8, 5, 0, 2, 5, 12, 9, 12, 10, 6, 2, 9, 15, 7, 3]]
184	[13, 13, [3, 6, 13, 10, 10, 15, 0, 7, 8, 12, 6, 0, 9, 13, 3, 5]]
185	[12, 8, [14, 11, 6, 1, 7, 2, 11, 12, 12, 8, 4, 2, 13, 9, 1, 7]]
186	[15, 3, [1, 7, 11, 15, 4, 2, 10, 14, 2, 5, 8, 13, 15, 8, 1, 4]]
187	[14, 4, [15, 11, 3, 5, 14, 10, 6, 0, 5, 0, 9, 14, 12, 9, 4, 3]]
188	[9, 4, [8, 12, 4, 2, 9, 13, 1, 7, 2, 7, 14, 9, 11, 14, 3, 4]]
189	[8, 1, [12, 8, 6, 0, 13, 9, 3, 5, 15, 10, 5, 2, 6, 3, 8, 15]]
190	[11, 14, [12, 11, 0, 5, 1, 6, 9, 12, 6, 0, 10, 14, 3, 5, 11, 15]]
191	[10, 9, [13, 8, 7, 0, 4, 1, 10, 13, 14, 10, 4, 2, 15, 11, 1, 7]]
192	[10, 14, [13, 10, 1, 4, 0, 7, 8, 13, 7, 1, 11, 15, 2, 4, 10, 14]]

i	$\mathsf{DS}^d[i]$
193	[11, 9, [12, 9, 6, 1, 5, 0, 11, 12, 15, 11, 5, 3, 14, 10, 0, 6]]
194	[8, 6, [5, 3, 9, 13, 0, 6, 8, 12, 15, 8, 3, 6, 2, 5, 10, 15]]
195	[9, 3, [7, 1, 13, 9, 2, 4, 12, 8, 4, 3, 14, 11, 9, 14, 7, 2]]
196	[14, 3, [0, 6, 10, 14, 5, 3, 11, 15, 3, 4, 9, 12, 14, 9, 0, 5]]
197	[15, 4, [14, 10, 2, 4, 15, 11, 7, 1, 4, 1, 8, 15, 13, 8, 5, 2]]
198	[12, 15, [12, 11, 2, 7, 1, 6, 11, 14, 7, 1, 9, 13, 2, 4, 8, 12]]
199	[13, 10, [7, 0, 15, 10, 10, 13, 6, 3, 5, 3, 13, 9, 0, 6, 12, 8]]
200	[2, 12, [9, 12, 5, 2, 0, 5, 8, 15, 3, 7, 15, 9, 2, 6, 10, 12]]
201	[3, 10, [9, 14, 1, 4, 4, 3, 8, 13, 11, 13, 3, 7, 14, 8, 2, 6]]
202	[0, 4, [1, 5, 13, 11, 0, 4, 8, 14, 11, 14, 7, 0, 2, 7, 10, 13]]
203	[1, 0, [1, 5, 9, 15, 0, 4, 12, 10, 3, 6, 11, 12, 10, 15, 6, 1]]
204	[6, 9, [1, 4, 11, 12, 8, 13, 6, 1, 2, 6, 8, 14, 3, 7, 13, 11]]
205	[7, 15, [7, 0, 9, 12, 10, 13, 0, 5, 12, 10, 2, 6, 9, 15, 3, 7]]
206	[4, 5, [1, 5, 15, 9, 0, 4, 10, 12, 10, 15, 4, 3, 3, 6, 9, 14]]
207	[5, 1, [1, 5, 11, 13, 0, 4, 14, 8, 2, 7, 8, 15, 11, 14, 5, 2]]
208	[14, 2, [6, 0, 14, 10, 3, 5, 15, 11, 4, 3, 12, 9, 9, 14, 5, 0]]
209	[15, 5, [10, 14, 4, 2, 11, 15, 1, 7, 1, 4, 15, 8, 8, 13, 2, 5]]
210	[12, 14, [11, 12, 7, 2, 6, 1, 14, 11, 1, 7, 13, 9, 4, 2, 12, 8]]
211	[13, 11, [0, 7, 10, 15, 13, 10, 3, 6, 3, 5, 9, 13, 6, 0, 8, 12]]
212	[10, 15, [10, 13, 4, 1, 7, 0, 13, 8, 1, 7, 15, 11, 4, 2, 14, 10]]
213	[11, 8, [9, 12, 1, 6, 0, 5, 12, 11, 11, 15, 3, 5, 10, 14, 6, 0]]
214	[8, 7, [3, 5, 13, 9, 6, 0, 12, 8, 8, 15, 6, 3, 5, 2, 15, 10]]
215	[9, 2, [1, 7, 9, 13, 4, 2, 8, 12, 3, 4, 11, 14, 14, 9, 2, 7]]
216	[6, 8, [4, 1, 12, 11, 13, 8, 1, 6, 6, 2, 14, 8, 7, 3, 11, 13]]
217	[7, 14, [0, 7, 12, 9, 13, 10, 5, 0, 10, 12, 6, 2, 15, 9, 7, 3]]
218	[4, 4, [5, 1, 9, 15, 4, 0, 12, 10, 15, 10, 3, 4, 6, 3, 14, 9]]
219	[5, 0, [5, 1, 13, 11, 4, 0, 8, 14, 7, 2, 15, 8, 14, 11, 2, 5]]
220	[2, 13, [12, 9, 2, 5, 5, 0, 15, 8, 7, 3, 9, 15, 6, 2, 12, 10]]
221	[3, 11, [14, 9, 4, 1, 3, 4, 13, 8, 13, 11, 7, 3, 8, 14, 6, 2]]
222	[0, 5, [5, 1, 11, 13, 4, 0, 14, 8, 14, 11, 0, 7, 7, 2, 13, 10]]
223	[1, 1, [5, 1, 15, 9, 4, 0, 10, 12, 6, 3, 12, 11, 15, 10, 1, 6]]
224	[6, 11, [11, 12, 1, 4, 6, 1, 8, 13, 8, 14, 2, 6, 13, 11, 3, 7]]
225	[7, 13, [9, 12, 7, 0, 0, 5, 10, 13, 2, 6, 12, 10, 3, 7, 9, 15]]
226	[4, 7, [15, 9, 1, 5, 10, 12, 0, 4, 4, 3, 10, 15, 9, 14, 3, 6]]
227	[5, 3, [11, 13, 1, 5, 14, 8, 0, 4, 8, 15, 2, 7, 5, 2, 11, 14]]
228	[2, 14, [5, 2, 9, 12, 8, 15, 0, 5, 15, 9, 3, 7, 10, 12, 2, 6]]
229	[3, 8, [1, 4, 9, 14, 8, 13, 4, 3, 3, 7, 11, 13, 2, 6, 14, 8]]
230	[0, 6, [13, 11, 1, 5, 8, 14, 0, 4, 7, 0, 11, 14, 10, 13, 2, 7]]
231	[1, 2, [9, 15, 1, 5, 12, 10, 0, 4, 11, 12, 3, 6, 6, 1, 10, 15]]
232	[14, 1, [10, 14, 0, 6, 11, 15, 5, 3, 9, 12, 3, 4, 0, 5, 14, 9]]
233	[15, 6, [2, 4, 14, 10, 7, 1, 15, 11, 8, 15, 4, 1, 5, 2, 13, 8]]
234	[12, 13, [2, 7, 12, 11, 11, 14, 1, 6, 9, 13, 7, 1, 8, 12, 2, 4]]
235	[13, 8, [15, 10, 7, 0, 6, 3, 10, 13, 13, 9, 5, 3, 12, 8, 0, 6]]
236	[10, 12, [1, 4, 13, 10, 8, 13, 0, 7, 11, 15, 7, 1, 10, 14, 2, 4]]
237	[11, 11, [6, 1, 12, 9, 11, 12, 5, 0, 5, 3, 15, 11, 0, 6, 14, 10]]
238	[8, 4, [9, 13, 5, 3, 8, 12, 0, 6, 3, 6, 15, 8, 10, 15, 2, 5]]
239	[9, 1, [13, 9, 7, 1, 12, 8, 2, 4, 14, 11, 4, 3, 7, 2, 9, 14]]
240	[0, 7, [11, 13, 5, 1, 14, 8, 4, 0, 0, 7, 14, 11, 13, 10, 7, 2]]
241	[1, 3, [15, 9, 5, 1, 10, 12, 4, 0, 12, 11, 6, 3, 1, 6, 15, 10]]
242	[2, 15, [2, 5, 12, 9, 15, 8, 5, 0, 9, 15, 7, 3, 12, 10, 6, 2]]
243	[3, 9, [4, 1, 14, 9, 13, 8, 3, 4, 7, 3, 13, 11, 6, 2, 8, 14]]
244	[4, 6, [9, 15, 5, 1, 12, 10, 4, 0, 3, 4, 15, 10, 14, 9, 6, 3]]
245	[5, 2, [13, 11, 5, 1, 8, 14, 4, 0, 15, 8, 7, 2, 2, 5, 14, 11]]
246	[6, 10, [12, 11, 4, 1, 1, 6, 13, 8, 14, 8, 6, 2, 11, 13, 7, 3]]
247	[7, 12, [12, 9, 0, 7, 5, 0, 13, 10, 6, 2, 10, 12, 7, 3, 15, 9]]
248	[8, 5, [13, 9, 3, 5, 12, 8, 6, 0, 6, 3, 8, 15, 15, 10, 5, 2]]
249	[9, 0, [9, 13, 1, 7, 8, 12, 4, 2, 11, 14, 3, 4, 2, 7, 14, 9]]
250	[10, 13, [4, 1, 10, 13, 13, 8, 7, 0, 15, 11, 1, 7, 14, 10, 4, 2]]
251	[11, 10, [1, 6, 9, 12, 12, 11, 0, 5, 3, 5, 11, 15, 6, 0, 10, 14]]
252	[12, 12, [7, 2, 11, 12, 14, 11, 6, 1, 13, 9, 1, 7, 12, 8, 4, 2]]
253	[13, 9, [10, 15, 0, 7, 3, 6, 13, 10, 9, 13, 3, 5, 8, 12, 6, 0]]
254	[14, 0, [14, 10, 6, 0, 15, 11, 3, 5, 12, 9, 4, 3, 5, 0, 9, 14]]
255	[15, 7, [4, 2, 10, 14, 1, 7, 11, 15, 15, 8, 1, 4, 2, 5, 8, 13]]

B Example: Computing Eighth Element of Sequence

Consider the parameters as given in Appendix A. The 25-round sequence is given in Table 8.

Table 8: 25-round sequence

i	0	1	2	3	4	5	6	7	8	9	10	11	12	-
k_i^e	1	2	3	4	3	0	8	10	1	3	15	15	7	-
s_i^e	15	14	9	11	7	4	5	1	9	12	3	1	8	-

i	13	14	15	16	17	18	19	20	21	22	23	24	25	26
k_i^e	6	8	6	4	11	13	11	2	3	3	1	7	-	-
s_i^e	12	4	11	1	9	2	10	5	4	6	10	15	11	5

We compute s_7^d using data structure given in Table 7 in the following 3 steps.

1. Find the row corresponding to (k_0^d, k_1^d) in $\mathsf{DS^d}$. The value of row is given by:

$$\begin{aligned} \mathsf{row} &= L^4(k_0^d)||k_1^d \\ &= 7||1 = 113 \text{ (as } k_0^d = k_{24}^e = 7 \text{ and } k_1^d = k_{23}^e = 1) \end{aligned}$$

2. Compute the partition $p = \mathsf{DS^d}[113][2][k_2^d] + k_3^d = \mathsf{DS^d}[113][2][3] + 3 = \mathsf{DS^d}[113][2][3] + 3 = 7 + 3 = 4$. Note that $\mathsf{DS^d}[113][0][0] = s_3^d = s_{23}^e = 10$ and $X^d = 5$.

3. Compute s_7^d. We have

$$\begin{aligned} s_7^d &= s_3^d + k_5^d + p + \mathsf{DS^d}[113][2][s_6^d + X^d] \\ &= 10 + 11 + 4 + \mathsf{DS^d}[113][2][5 + 5] \\ &= 5 + \mathsf{DS^d}[113][2][0] = 5 + 15 = 10 = s_{19}^e. \end{aligned}$$

C Python Implementation

C.1 Main.py

```
## ******* Classes ******* ##
from segments import *
from indexset import *
from functions import *
from extractkeys import *
from datastr import *

## ******* Input parameters ******* ##
ws = 4          # wordsize
a = 0           # Shift parameter a
b = 1           # Shift parameter b
c = 2           # Shift parameter c
N = 1<<ws
t_round = 25    # Total rounds
pe_round = 12   # Partial encryption rounds
# pe_round = 13 # if t_round = 27


if __name__ == "__main__":
```

```
secret_key = [1, 2, 3, 4]        # random secret key
pt = [[15, 14], [13, 12], [11, 10]] # 3 random plaintexts
ct = []                          # ciphertexts
rk = keyexpansion(secret_key, t_round, ws, a, b, c)
for i in range(len(pt)):
        ct.append(encryption(pt[i], rk, t_round, ws, a, b
            , c))
se0 = pt[0][0]   # s0 from encryption side
se1 = pt[0][1]   # s1 from encryption side
sd0 = ct[0][1]   # s0 from decryption side
sd1 = ct[0][0]   # s1 from decryption side

## ******* START : z-linear segment sets  ******* ##
ZS = zsegments(ws, a, b, c)
[Z,CLZ] = ZS.construct_zsegments()
# print_zset(Z, CLZ)
## ******* END : z-linear segment sets  ******* ##

## **** Data structure for 6 decryption rounds **** ##
X = datastr(ws, a, b, c, Z, CLZ, sd0, sd1)
DS = X.construct_ds()

## ******* START : Recover secret key  ******* ##

E = extract_keys(ws, a, b, c, se0, se1, pe_round, t_round
    , DS, Z, CLZ)
K = E.filter_keys()

K1 = []; K2 = []
for key in K:
        rk = keyexpansion(key, t_round, ws, a, b, c)
        if([ct[1][0], ct[1][1]] == encryption(pt[1], rk,
            t_round, ws, a, b, c) ):
                K1.append(key)

for key in K1:
        rk = keyexpansion(key, t_round, ws, a, b, c)
        if([ct[2][0], ct[2][1]] == encryption(pt[2], rk,
            t_round, ws, a, b, c) ):
                K2.append(key)

if(secret_key == K2[0]):
        print("KEY Found")
        print("Secret key :" + str(K2[0]))

## ******* END : Recover secret key  ******* ##
```

C.2 Class functions.py

```
## *** Rotates a ws-bit word cyclically left by j bits *** ##
def lrotate(word, j, ws):
        return  ((word << j) % (1<<ws) ) | (word >> (ws - j))

## *** Rotates a ws-bit word cyclically right by j bits *** ##
def rrotate(word, j, ws):
        return ((word << (ws - j)) % (1 << ws)) | (word >> j)

## **** Simon-like nonlinear round function **** ##
```

```
def rf(lword, ws, a, b, c):
        return( lrotate(lword, a, ws) & lrotate(lword, b, ws) ^
            lrotate(lword, c, ws) )

## **** Key scheduling algorithms **** ##
z =[1,1,1,1,1,0,1,0,0,0,1,0,0,1,0,1,0,1,1,0,0,0,0, \
    1,1,1,0,0,1,1,0,1,1,1,1,1,0,1,0,0,0,1,0,0,1,0, \
    1,0,1,1,0,0,0,0,1,1,1,0,0,1,1,0]
def keyexpansion(key, rounds, ws, a, b, c):
        con = (1 << ws) - 4; m = 4; round_key = []
        for i in range(m):
                round_key.append(key[i])
        for i in range(m,rounds):
        tmp_simon = round_key[i-4] ^ con ^ z[(i-4)%len(z)] ^ (
            rrotate(round_key[i-1], 3, ws)) ^ (rrotate(round_key[i
            -1], 4, ws)) ^ rrotate(round_key[i-3], 1, ws) ^
            round_key[i-3]
        #tmp_simeck = round_key[i-4] ^ con ^ z[(i-4)%len(z)] ^ ((
             lrotate(round_key[i-3], a, ws) & lrotate(round_key[i
            -3], b, ws) ) ^ lrotate(round_key[i-3], c, ws))
        round_key.append(tmp_simon)

        return round_key

## ******* Encryption ******* ##
def encryption(pt, subkey, rounds, ws, a, b, c):
        s = []; s.append(pt[0]); s.append(pt[1])
        for i in range(2, rounds+2):
                tmp = s[i-2] ^ (( lrotate(s[i-1], a, ws) &
                    lrotate(s[i-1], b, ws) ) ^ lrotate(s[i-1], c,
                    ws)) ^ subkey[i-2]
                s.append(tmp)

        return [s[-2], s[-1]]

## ******* Print z-linear segment sets ******* ##
def print_zset(Z, CLZ):
        j = 0
        for z in Z:
                print(z,CLZ[j])
                j = j + 1
        print("Nz : " + str(len(Z)))

## ******* Compute 8-th element of sequence ******* ##
def compute_s7(D, K, ws):
        s3 = D[(K[0]<<ws) + K[1]][0]
        X  = D[(K[0]<<ws) + K[1]][1]
        I  = D[(K[0]<<ws) + K[1]][2]
        p  = I[K[2]] ^ K[3]
        s7 = s3 ^ K[5] ^ p ^ I[I[p ^ X] ^ K[2] ^ K[4] ^ s3]
        return s7
```

C.3 Class segments.py

```
import numpy as np
from functions import *
CL4 = [0, 1, 3, 5, 7, 15]        # Coset leaders for ws = 4
CL8 = [ 0, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, \
```

```
        21, 23, 25, 27, 29, 31, 37, 39, 43, 45, \
        47, 51, 53, 55, 59, 61, 63, 85, 87, 91, \
        95, 111, 119, 127, 255] # Coset leaders for ws = 8
# Coset leaders for ws = 10, 12, 14, 16 are not included due to
    space.

class zsegments:
        def __init__(self, ws, a, b, c):
        self.ws = ws; self.a = a; self.b = b; self.c = c

        def construct_zsegments(self):
                Z = [] ; TEMP1 = [] ; CL = []
                if(self.ws == 4): TEMP = CL4
                elif(self.ws == 8): TEMP = CL8

                for s in TEMP:
                        TEMP1.append( rf(s, self.ws, self.a, self
                            .b, self.c) ^ lrotate(s, self.c, self.
                            ws) )

                Z = np.unique(TEMP1)

                for z in Z:
                        indices = [i for i, x in enumerate(TEMP1)
                             if x == z]
                        CL.append([TEMP[k] for k in indices] )

                return(Z,CL)
```

C.4 Class indexset.py

```
import numpy as np
from functions import *

class indexset:
        def __init__(self, ws, c, X, s3, Z, CLZ):
        self.ws = ws; self.c = c; self.X = X;
        self.s3 = s3; self.Z = Z; self.CLZ = CLZ

        def construct_IK(self):
                IK_k2 = []; IK_k3 = []
                for k in range(len(self.CLZ)):
                        for x in self.CLZ[k]:
                                z = self.Z[k]
                                for i in range(self.ws):
                                        y = lrotate(x, i, self.ws
                                            )
                                        T = lrotate(y, self.c,
                                            self.ws) ^ lrotate(z,
                                            i, self.ws)
                                        IK_k2.append(y ^ self.X)
                                        IK_k3.append(T ^ self.s3)
                                        if(lrotate(y, 1, self.ws)
                                            == x): break

                return( list( zip(*sorted( zip(IK_k2, IK_k3)))[1]
                    ))
```

C.5 Class datastr.py

```
import numpy as np
from functions import *
from indexset import *

class datastr:
        def __init__(self, ws, a, b, c, Z, CLZ, se0, se1):
                self.ws = ws; self.a = a; self.b = b; self.c = c
                self.se0 = se0; self.se1 = se1; self.Z = Z ;
                self.CLZ = CLZ

        def construct_ds(self):
                N = 1<<self.ws; DS = []
                for ke0 in range(N):
                        for ke1 in range(N):
                                se2 = rf(self.se1, self.ws, self.
                                    a, self.b, self.c) ^ self.se0
                                    ^ ke0
                                se3 = rf(se2, self.ws, self.a,
                                    self.b, self.c) ^ self.se1 ^
                                    ke1
                                Xe = rf(se3, self.ws, self.a,
                                    self.b, self.c) ^ se2
                                IKe = indexset(self.ws, self.c,
                                    Xe, se3, self.Z, self.CLZ)
                                Ie = IKe.construct_IK()
                                DS.append([se3, Xe, Ie])

                return DS
```

C.6 Class extractkeys.py

```
import numpy as np
from functions import *
from segments import *
from indexset import *

class extract_keys:
        def __init__(self, ws, a, b, c, se0, se1, pe_round,
            t_round, DSd, Z, CLZ):
                self.ws = ws; self.a = a; self.b = b; self.c = c
                self.se0 = se0; self.se1 = se1; self.DSd = DSd
                self.pe_round = pe_round; self.t_round = t_round
                self.Z = Z; self.CLZ = CLZ

        def filter_keys(self):
                KEYS = []
                #kei: i-th subkey starting from encryption round
                #sei: i-th half-state starting from encryption
                    round

                for ke0 in range(1<<self.ws):
                        for ke1 in range(1<<self.ws):
                                se2 = rf(self.se1, self.ws, self.
                                    a, self.b, self.c) ^ self.se0
                                    ^ ke0
```

```
                                        se3 = rf(se2, self.ws, self.a,
                                            self.b, self.c) ^ self.se1 ^
                                            ke1
                                        Xe = rf(se3, self.ws, self.a,
                                            self.b, self.c) ^ se2

                                        IKe = indexset(self.ws, self.c,
                                            Xe, se3, self.Z, self.CLZ)
                                        Ie = IKe.construct_IK()
                                        go to fun()

                        return KEYS

        ## ******* fun() ******* ##

        for k in range(len(self.CLZ)):
                for x in self.CLZ[k]:
                        z = self.Z[k]
                        for i in range(self.ws):
                                y = lrotate(x, i, self.ws)
                                f = lrotate(y, self.c, self.ws) ^ lrotate
                                    (z, i, self.ws)
                                if(self.t_round == 25):
                                        go to fun1()
                                elif(self.t_round == 27):
                                        go to fun2()

                                if(lrotate(y, 1, self.ws) == x):
                                        break

        ## ******* fun1() ******* ##

        for j in range(1<<self.ws):
                K = keyexpansion([ke0, ke1, j, Ie[j] ^ y ], self.t_round,
                     self.ws, self.a, self.b, self.c)
                se6 = f ^ Xe ^ K[2] ^ K[4]
                se7 = se3 ^ K[5] ^ y ^ Ie[f ^ K[2] ^ K[4]]
                pt = [se6, se7]
                subkey = [K[i+6] for i in range(self.pe_round)]
                TEMP = encryption(pt, subkey, self.pe_round, self.ws,
                    self.a, self.b, self.c)
                if(TEMP[-1] == compute_s7(self.DSd, [K[-i] for i in range
                    (1,7)], self.ws)):
                        KEYS.append([K[i] for i in range(4)])

        ## ******* fun2() ******* ##

        S6 = []; S7 = []
        for j in range(1<<self.ws):
                K = keyexpansion([ke0, ke1, j, Ie[j] ^ y ], self.t_round,
                     self.ws, self.a, self.b, self.c)
                se6 = f ^ Xe ^ K[2] ^ K[4]
                se7 = se3 ^ K[5] ^ y ^ Ie[f ^ K[2] ^ K[4]]
                S6.append(se6); S7.append(se7); S7_U = np.unique(S7)
                for u in S7_U:
                        indices = [l for l, t in enumerate(S7) if t == u]
                        te = rf(u, self.ws, self.a, self.b, self.c)
```

```
for ind in indices:
        K = keyexpansion([ke0, ke1, ind, Ie[ind]
            ^ y ], self.t_round, self.ws, self.a,
            self.b, self.c)
        pt = [u, te ^ S6[ind] ^ K[6]]
        subkey = [K[i+7] for i in range(self.
            pe_round)]
        TEMP = encryption(pt, subkey, self.
            pe_round, self.ws, self.a, self.b,
            self.c)
        if(TEMP[-1] == compute_s7(self.DSd, [K[-i
            ] for i in range(1,7)], self.ws)):
                KEYS.append([K[i] for i in range
                    (4)])
```