CHAPTER 4

SECURE CRYPTOGRAPHIC COMPONENTS

4.1 Introduction

This chapter presents the findings of the systematic literature review on existing lightweight block ciphers in order to identify the secure cryptographic components. The secure cryptographic components are analysed for the purpose of lightweight block cipher development. Other than that, the 3-dimensional cipher is discussed to introduce the method used in the proposed cryptographic algorithm.

4.2 Systematic Literature Review on Lightweight Block Cipher

In this section, a systematic literature review on existing lightweight block ciphers was conducted in order to understand their cryptographic algorithm designs. Component details of lightweight block ciphers that include the block size, key size, structure, number of rounds, and cryptographic functions are listed in Table 4.1. The lightweight block cipher components are further studied in the following sections to achieve the goal of the research which is to identify cryptographic components that can enhance the security strength of lightweight block ciphers.

Table 4.1: Lightweight Block Cipher Components

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
1	µ2 (Yeoh et al., 2020)	64	80	FN	15	1.Add Round Key 2.Substitution 3.Permutation	1.Rotation 2.Substitution 3.Counter XOR	PRESENT (S-box)
2	ACT (Jithendra & Kassim, 2020)	64	80	SPN	31	1.Add Round Key 2.Sub Nibble 3.Permutation	1.LFSR 2.XOR 3.Shift	-
3	ANU (Bansod et al., 2016b)	64	80/128	FN	25	1.Rotation 2.Substitution 3.XOR 4.Permutation	1.Rotation 2.Substitution 3.Constant XOR	PRESENT (Key Schedule)
4	ANU–II (Dahiphale et al., 2018)	64	80/128	FN	25	1.Substitution 2.Shift 3.Add Round Key	1.Rotation 2.Substitution 3.Constant XOR	ANU (Component); PRESENT (Key Schedule)
5	BEST-1 (John, 2014)	64	128	ARX	12	1.Initial Transformation 2.Round Function 3.Final Transformation	1.Transformation Key Generation 2.Subkey Generation	-
6	Blowfish (Schneier, 1993)	64	32-448	FN	16	1.Substitution 2.Modulo Addition 3.XOR	1.Permutation 2.Substitution 3.XOR	-
7	BORON (Bansod et al., 2017)	64	80/128	SPN	25	1.Add Round Key 2.Substitution 3.Block Shuffle 4.Permutation 5.XOR	1.Rotation 2.Substitution 3.Counter XOR	PRESENT (Key Schedule)
8	BRIGHT (Sehrawat & Gill, 2019)	64/128	80/96/128/ 192/256	FN	32/33/34/ 35/36/37	1.Add Round Key 2.ARX Operation 3.Permutation	1.XOR 2.LFSR	SPECK (Key Schedule)
9	CAST (Adams, 1997)	64/128	128/160/ 192/224/ 256/varies	FN	12/16/48	1.Substitution 2.Rotation 3.Modulo Addition 4.Modulo Subtraction 5.XOR	1.Modulo Addition 2.Modulo Subtraction 3.XOR 3.Shift	-
10	CHAM (Koo et al., 2017)	64/128	128/256	ARX	80/96	1.Modulo Addition 2.XOR 3.Rotation	1.Rotation 2.XOR	-

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
11	CRAFT (Beierle et al., 2019)	64	128 + 64 Tweak	SPN	31	1.Substitution 2.Mix Column 3.Permute Nibble 4.Constant XOR 5.Add Round Tweakey	-	Midori (S-box); Piccolo & Midori (Key Schedule)
12	CUBE (Berger et al., 2015)	64/125/216	128/250/432	SPN	15	1.Add Round Key 2.Substitution 3.MDS Matrix 4.Permutation	1.XOR 2.Matrix Multiplication	NOEKEON (S-box)
13	DESL (Leander et al., 2007)	64	56	FN	16	1.Add Round Key 2.Substitution 3.Permutation	1.Permutation 2.Rotation	DES (Component)
14	DLBCA (Al-Dabbagh, 2017)	32	80	FN	15	1.Add Round Key 2.Substitution 3.Permutation 4.Rotation	1.Substitution 2.Rotation	HISEC (S-box)
15	DoT (Patil et al., 2019)	64	80/128	SPN	31	1.Add Round Key 2.Substitution 3.Permutation 4.Rotation	1.Rotation 2.Substitution 3.Counter XOR	PRESENT (Key Schedule)
16	EPCBC (Yap et al., 2011)	48/96	96	SPN	32	1.Substitution 2.Permutation	1.F-function 2.Add Round Key	PRESENT (Component & S-box)
17	ESF (Liu et al., 2014)	64	80	FN	31	1.Add Round Key 2.Substitution 3.Permutation	1.Shift 2.Substitution 3.Counter XOR	LBlock (Component); PRESENT (Component & Key Schedule); Serpent (S-box)
18	FeW (Kumar et al., 2019)	64	80/128	FN	32	1.F-function 2.Weight Function	1.Rotation 2.Substitution 3.Counter XOR	CLEFIA (Component); PRESENT (Key Schedule); Hummingbird-2 (S-box)
19	FlexAE (Nascimento & Xexéo, 2019)	64 + Tweak	128 + Tweak	FN	Variable	1.Add Round Key 2.Block Shuffle 3.Substitution	1.Permutation	AES (S-box)

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
20	GIFT (Banik et al., 2017)	64/128	128	SPN	28/40	1.Sub Cell 2.Permutation 3.Add Round Key	1.Rotation 2.Constant XOR	PRESENT (Component)
21	GRANULE (Bansod et al., 2018a)	64	80/128	FN	32	1.Permutation 2.Substitution 3.Add Round Key	1.Shift 2.Substitution 3.Counter XOR	PRESENT (Key Schedule)
22	Halka (Das, 2014)	64	80	NLFSR	24	1.Add Round Key 2.Substitution 3.Permutation	1.Rotation 2.Substitution 3.Counter XOR	PRESENT (Key Schedule); AES (S-box)
23	HERMES (Malutan et al., 2019)	64	128	SPN	30	1.Add Round Key 2.Substitution 3.Rotation 4.XOR 5.Permutation	1.Substitution 2.Rotation 3.Modulo Addition 4.Modulo Subtraction	-
24	HIGHT (Hong et al., 2006)	64	128	ARX	32	1.Initial Transformation 2.Round Function 3.Final Transformation	1.Whitening Key Generation 2.Subkey Generation	-
25	HISEC (Al Dabbagh et al., 2014)	64	80	GFN	15	1.Add Round Key 2.Substitution 3.Permutation 4.Rotation 5.XOR	1.Substitution 2.Rotation	PRESENT (Component)
26	Hummingbird (Engels et al., 2010)	16	256 + 80 Internal State	Hybrid	4	1.Add Round Key 2.Substitution 3.Permutation	-	-
27	Hummingbird-2 (Engels et al., 2011)	16	128 + 128 Internal State + 64 IV	Hybrid	4	1.Add Round Key 2.Substitution 3.Permutation	-	-
28	Hybrid PRESENT & Salsa20 (Kubba & Hoomod, 2019)	64	128	SPN	20	1.Padding 2.Add Round Key 3.Permutation 4.Keystream XOR	1.Chaos Key Generation 2.Quarter Round Function 3.Row Round Function	PRESENT (Component); Salsa20 (Key Schedule)

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
29	ICEBERG (Standaert et al., 2004)	64	128	SPN	16	1.Substitution 2.Permutation 3.Add Round Key 4.Matrix Multiplication	1.Key Expansion 2.Shift 3.Substitution 4.Permutation	-
30	IDEA (Lai & Massey, 1991)	64	128	ARX	8.5	1.XOR 2.Modulo Addition 3.Modulo Multiplication	1.Key Partition 2.Rotation	-
31	ILEA (Jha et al., 2020)	64	128	SPN	12	1.Balanced Block Mixer 2.Add Round Key 3.Substitution 4.Permutation 5.Constant XOR	-	PRINCE (S-box)
32	Improved DLBCA (Al-Dabbagh et al., 2018)	32	80	FN	15	1.Add Round Key 2.Substitution 3.Permutation 4.Rotation	1.Substitution 2.Rotation	DLBCA (Component); HISEC (S-box)
33	Improved GOST (Poschmann et al., 2010)	64	256	FN	32	1.Modulo Addition 2.Substitution 3.Rotation	-	PRESENT (S-box)
34	Improved IDEA (Lerman et al., 2013)	64	128	ARX	6.5	1.XOR 2.Modulo Addition	1.XOR 2.Rotation 3.Modulo Addition	IDEA (Component); MESH (Key Schedule)
35	Improved LBlock (Al-Dabbagh & Shaikhli, 2013)	64	80	FN	32	1.XOR 2.Substitution 3.Rotation 4.Permutation	1.Rotation 2.Substitution 3.Constant XOR	LBlock (Component, S-box & Key Schedule)
36	Improved RoadRunneR (Liu et al., 2018)	64	80/128	FN	10/12	1.Merged Substitution 2.Diffusion Layer 3.Add Round Key	1.Whitening Key Generation 2.Subkey Generation	RoadRunneR (Component)
37	Improved Simeck (Encarnacion et al., 2020)	32/64	64/128	FN	32/44	1.XOR 2.Bitwise AND 3.Rotation	1.Constant XOR 2.LFSR	Simeck (Component)

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
38	Improved SM4 (Chen et al., 2021)	64	64	FN	32	1.Rotation 2.XOR 3.Add Round Key 4.Substitution 5.Reverse Order Transformation	1.XOR 2.Rotation	SM4 (Component)
39	JAC_Jo (Shantha & Arockiam, 2018)	32	64	FN	32	1.Rotation 2.Bitwise AND 3.XOR	1.Constant XOR 2.LFSR	Simeck (Component & Key Schedule)
40	Kasumi (ETSI, 2014)	64	128	FN	8	1.Function FL 2.Function FO 3.Function FI	1.KL 2.KO 3.KI	-
41	KATAN (De Cannière et al., 2009)	32/48/64	80	NLFSR	254	1.LFSR	1.LFSR	Trivium (Component)
42	KHAZAD (Barreto & Rijmen, 2000)	64	128	SPN	8	1.Non-linear Layer 2.Linear Diffusion Layer 3.Add Round Key	1.Constant XOR	-
43	Khudra (Kolay & Mukhopadhyay, 2014)	64	80	FN	18	1.F-function 2.Feistel Permutation	1.Round Constant 2.XOR	PRESENT (S-box)
44	KLEIN (Gong et al., 2011)	64	64/80/96	SPN	12/16/20	1.Sub Nibble 2.Rotate Nibble 3.Mix Nibble	1.Shift 2.Feistel Transformation 3.XOR	AES (Component)
45	KTANTAN (De Cannière et al., 2009)	32/48/64	80	NLFSR	254	1.LFSR	1.LFSR	Trivium (Component)
46	LBC-IoT (Ramadan et al., 2021)	32	80	FN	32	1.Substitution 2.Permutation 3.Add Round Key	1.Substitution 2.Permutation 3.Rotation	-
47	LBlock (Wu & Zhang, 2011)	64	80	FN	32	1.XOR 2.Substitution 3.Rotation 4.Word Permutation	1.Rotation 2.Substitution 3.Constant XOR	-

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
48	LCA (Aboshosha et al., 2019)	64	256	FN	16	1.XOR 2.Modulo Addition 3.Modulo Subtraction 4.Rotation	1.Key Partition 2.Rotation	-
49	LED (Guo et al., 2011)	64	64/128	SPN	32/48	1.Add Round Key 2.Constant XOR 3.Sub Cell 4.Shift Row 5.Mix Column	-	AES (Component); PRESENT (S-box)
50	LiCi (Patil et al., 2017)	64	128	FN	31	1.Substitution 2.Add Round Key 3.Rotation	1.Shift 2.Substitution 3.Counter XOR	PRESENT (Key Schedule)
51	LILLIPUT (Berger et al., 2015)	64	80	FN	30	1.Non-linear Layer 2.Linear Layer 3.Permutation	1.Key Mixing 2.Permutation 3.Subkey Generation	-
52	Loong (Liu et al., 2019)	64	64/80/128	SPN	16/20/32	1.Add Round Key 2.Sub Cell 3.Mix Row 4.Mix Column	1.Constant XOR 2.Modulo Addition	-
53	LRBC (Biswas et al., 2020)	16	16	Hybrid	24	1.Key XOR/XNOR 2.Substitution 3.Permutation 4.L-box Computation	1.Key Combination	-
54	LWE (Toprak et al., 2020)	64	64	Hybrid	3	1.Key XNOR 2.Substitution 3.XOR	1.Substitution 2.XOR 3.Permutation	-
55	MANTIS (Beierle et al., 2016)	64	128 + 64 Tweak	SPN	14	1.Mix Column 2.Permute Cells 3.Add Round Tweakey 4.Constant XOR 5.Sub Cell	1.Rotation 2.XOR	PRINCE (Component); Midori (S-box)
56	MANTRA (Bansod et al., 2018b)	64	80/128	FN	32	1.Substitution 2.Add Round Key 3.Rotation	1.Shift 2.Substitution 3.Counter XOR	PRESENT (Key Schedule)

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
57	mCrypton (Lim & Korkishko, 2005)	64	64/96/128	SPN	12	1.Substitution 2.Permutation 3.Column-to-Row Transposition 4.Add Round Key	1.Substitution 2.Rotation	-
58	MIBS (Izadi et al., 2009)	64	64/80	FN	32	1.Add Round Key 2.Substitution 3.Mixing 4.Permutation	1.Rotation 2.Substitution 3.Counter XOR	mCrypton (S-box); PRESENT (Key Schedule)
59	Midori (Banik et al., 2015)	64/128	128	SPN	16/20	1.Sub Cell 2.Shuffle Cell 3.Mix Column 4.Add Round Key	-	-
60	MISTY (Matsui, 1997)	64	128	FN	8	1.FO 2.FI 3.FL	1.FI	-
61	NUCLEAR (Salunke et al., 2019)	64	128	FN	25	1.Substitution 2.XOR 3.Rotation	2.XOR 3.Rotation	-
62	NUX (Bansod et al., 2018)	64	80/128	FN	31	1.Add Round Key 2.F-function 3.Permutation	1.Rotation 2.Substitution 3.Counter XOR	PRESENT (Key Schedule)
63	NVLC (Al-Rahman et al., 2018)	64	80/128	SPN	20	1.Add Round Key 2.Mix Column 3.Substitution 4.Shift	1.Shift 2.Substitution 3.Counter XOR	PRESENT (Key Schedule)
64	OLBCA (Al-Dabbagh & Shaikhli, 2014)	64	80	GFN	22	1.F-function 2.Rotation 3.XOR 4.Permutation	1.Substitution 2.Rotation	-
65	Piccolo (Shibutani et al., 2011)	64	80/128	GFN	25/31	1.Whitening Key XOR 2.F-function 3.Permutation	1.Constant Values	-
66	PICO (Bansod et al., 2016)	64	128	SPN	32	1.Add Round Key 2.Sub Column 3.Bit Shuffle	1.XOR 2.Rotation	SPECK (Key Schedule)

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
67	PRESENT (Bogdanov et al., 2007)	64	80/128	SPN	31	1.Add Round Key 2.Substitution 3.Permutation	1.Rotation 2.Substitution 3.Counter XOR	-
68	PRIDE (Albrecht et al., 2014)	64	128	SPN	20	1.Add Round Key 2.Substitution 3.Permutation	-	-
69	PRINCE (Borghoff et al., 2012)	64	128	SPN	12	1.Add Round Key 2.Substitution 3.Permutation 4.Constant XOR	-	-
70	PRINTcipher (Knudsen et al., 2010)	48/96	80/160	SPN	48/96	1.Add Round Key 2.Linear Diffusion 3.Round Counter 4.Keyed Permutation 5.Substitution	1.Keyed Permutation	-
71	PriPresent (Girija et al., 2020)	64/80	80/128	SPN	31	1.Add Round Key 2.Substitution 3.Permutation	1.Random Pentatope Number Generation	-
72	PUFFIN (Cheng et al., 2008)	64	128	SPN	32	1.Substitution 2.Add Round Key 3.Permutation	1.Permutation 2.Bit Inversion	ICEBERG (S-box)
73	PUFFIN2 (Wang & Heys, 2009)	64	80	SPN	34	1.Substitution 2.Add Round Key 3.Permutation	1.Substitution 2.Permutation 3.Position Selection	ICEBERG (S-box)
74	QTL (Li et al., 2016)	64	64/128	FN	16/20	1.Constant XOR 2.Add Round Key 3.Substitution 4.Permutation	-	PRESENT & mCrypton (S-box)
75	RARE (Omrani et al., 2018)	64	80	SPN	13	1.Substitution 2.Permutation	1.Chaotic Function 2.Rotation 3.XOR	-
76	RC5 (Rivest, 1994)	32/64/128	0-2040	ARX	0-255	1.Modulo Addition 2.Rotation 3.XOR	1.Word Conversion 2.Initialize Array 3.Key Mixing	-
77	RECTANGLE (Zhang et al., 2015)	64	80/128	SPN	25	1.Add Round Key 2.Sub Column 3.Shift Row	1.Sub Column 2.Feistel Transformation 3.Constant XOR	-

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
78	RoadRunneR (Baysal & Şahin, 2015)	64	80/128	FN	10/12	1.Substitution 2.Diffusion Layer 3.Add Round Key	1.Whitening Key Generation 2.Subkey Generation	-
79	SAFER (Massey, 1993)	64	64	SPN	Variable	1.XOR 2.Byte Addition 3.Permutation 4.Pseudo-Hadamard Transform	1.Rotation 2.Modulo Addition	-
80	SAT_Jo (Shantha & Arockiam, 2018)	64	80	SPN	31	1.Add Round Key 2.Substitution 3.Permutation	1.Fibonacci Sequence 2.Modulo Addition	-
81	SEA (Standaert et al., 2006)	48/96/144	48/96/144	FN	Variable	1.XOR 2.Substitution 3.Word Rotation 4.Rotation 5.Modulo Addition	1.XOR 2.Substitution 3.Word Rotation 4.Rotation 5.Modulo Addition	-
82	SFN (Li et al., 2018)	64	96	Hybrid	32	1.Add Round Key 2.Substitution 3.Permutation 4.Mix XOR	1.Add Round Key 2.Substitution 3.Mix Column 4.Mix Row/Mix XOR	Midori & PRINCE (S-box)
83	Simeck (Yang et al., 2015)	32/48/64	64/96/128	FN	32/36/44	1.XOR 2.Bitwise AND 3.Rotation	1.Constant XOR 2.LFSR	SIMON (Component) & SPECK (Key Schedule)
84	SIMON (Beaulieu et al., 2013)	32/48/64/ 96/128	64/72/96/ 128/144/ 192/256	FN	32/36/42/ 44/52/54/ 68/69/72	1.XOR 2.Bitwise AND 3.Rotation	1.Constant XOR 2.LFSR	-
85	SIT (Usman et al., 2017)	64	64	Hybrid	5	1.Add Round Key 2.Swapping 3.Substitution	1.Key Partition 2.F-function 3.Matrix Transformation 4.Key Arrangement 5.XOR	KHAZAD (Key Schedule)
86	SKINNY (Beierle et al., 2016)	64/128	64/128/192/ 256/384	SPN	32/36/40/ 48/56	1.Sub Cell 2.Constant XOR 3.Add Round Tweakey 4.Shift Row 5.Mix Column	1.Tweakey framework	AES (Component)

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
87	SKIPJACK (NSA, 1998)	64	80	FN	32	1.Permutation 2.XOR 3.Round counter	1.Rotation	-
88	SPARX (Dinu et al., 2016)	64/128	128/256	ARX	24/32/40	1.Modulo Addition 2.XOR 3.Rotation	1.Modulo Addition 2.Permutation	SPECKEY & NOEKEON (Component)
89	SPECK (Beaulieu et al., 2013)	32/48/64/ 96/128	64/72/96/ 128/144/ 192/256	ARX	22/23/26/ 27/28/29/ 32/33/34	1.XOR 2.Modulo Addition 3.Rotation	1.XOR 2.LFSR	Threefish (Component)
90	Sriram (Ramudu & Shanthi, 2015)	64	96/128	SPN	27	1.Add Round Key 2.Substitution 3.Permutation	1.Rotation 2.Substitution 3.Constant XOR	PRESENT (Key Schedule)
91	SR-LED (Patil et al., 2015)	64	128	SPN	12	1.Add Round Key 2.Constant XOR 3.Bit Slice 4.Sub Cell 5.Shift Row 6.Mix Column	1.XOR 2.LFSR	LED (Component); SPECK (Key Schedule); RECTANGLE (S-box)
92	TEA (Wheeler & Needham, 1994)	64	128	FN	64	1.XOR 2.Modulo Addition	1.Modulo Addition	-
93	TED (Thorat et al., 2020)	64	128	FN	26	1.Substitution 2.Rotation 3.Round Constant 4.Add Round Key 5.Permutation	1.Rotation 2.Substitution 3.Counter XOR	PRESENT (Key Schedule)
94	T-TWINE (Sakamoto et al., 2020)	64 + Tweak	80/128	GFN	36	1.Tweak Schedule 2.Substitution 3.Add Round Key	1.Counter XOR	TWINE & SKINNY (Component)
95	TWINE (Suzaki et al., 2011)	64	80/128	GFN	32/36	1.Substitution 2.Add Round Key 3.Block Shuffle	1.Counter XOR	-
96	UBRIGHT (Sehrawat & Gill, 2020)	32	80	GFN	22	1.Add Round Key 2.ARX Operation 3.Permutation	-	BRIGHT (Component); Chaskey & RoadRunneR (Key Schedule)

No.	Algorithm	Block Size (Bits)	Key Size (Bits)	Structure	Rounds	Function		Based on Existing Algorithm
						Encryption Algorithm	Key Schedule Algorithm	
97	VAYU (Gaurav et al., 2016c)	64	80/128	FN	31	1.Substitution 2.Rotation 3.XOR 4.Permutation	1.Rotation 2.Substitution 3.Constant XOR	PRESENT (Key Schedule)
98	VH (Dai et al., 2015)	64	64/80/96/ 112/128	SPN	10/11/12/ 13/14	1.Substitution 2.Permutation 3.Add Round Key	1.XOR 2.Pseudorandom Transformation	-
99	XSX (Santos et al., 2016)	64	64	SPN	4	1.Add Round Key 2.Bit Switch	1.Random Number Generation 2.XOR	-

The comparison of lightweight block ciphers in Table 4.1 shows that the construction of the observed algorithms is based on confusion and diffusion properties. Confusion and diffusion are two properties introduced by Claude Shannon that must be associated with lightweight block ciphers (Shannon, 1949). In confusion, the relationship between the key and ciphertext in an encryption operation is obscured. Confusion indicates how changing one bit of the key affects the entire ciphertext. When a situation like that happens, the attacker would probably need to solve the entire key simultaneously, rather than piece by piece to break the algorithm.

For diffusion, the influence of one plaintext symbol is spread over many ciphertext symbols with the goal of hiding the statistical properties of the plaintext. Diffusion indicates how changing one bit of the plaintext affects the entire ciphertext. This means that frequency statistics of the plaintext are diffused over several characters in the ciphertext, which means that much more ciphertext is needed to execute a meaningful statistical attack.

Both confusion and diffusion cannot provide security by themselves alone. Therefore, this research aims to implement confusion and diffusion elements to develop a secure algorithm. Each of the cryptographic functions of lightweight block cipher from Table 4.1 is categorized into confusion and diffusion categories as shown in Figure 4.1. From the categorization of confusion and diffusion elements, an in-depth study is conducted that focused on the main purpose of this research which is to develop a secure lightweight block cipher.

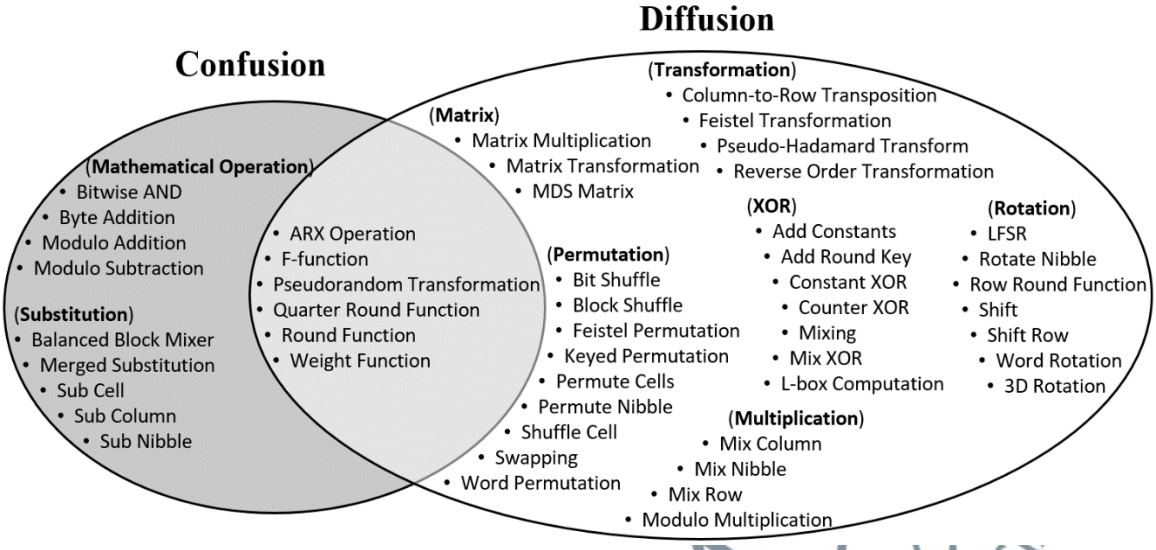

Figure 4.1: Functions of Lightweight Block Cipher

From the comparison of lightweight block ciphers, XOR is a basic mathematical operation used in almost every algorithm that performs a bitwise exclusive-or function between a fixed bit input and produces the same length bit output (Adams, 1997). There are various implementations of the XOR function to increase the diffusion property of a lightweight block cipher. The most implemented XOR function is the add round key that operates the XOR function between the cipher and the round key. In constant XOR or sometimes called add constants process, the cipher state is XOR with the round constant where the values are obtained either from the table, computation, or round number (Jha et al., 2020). Counter XOR function operates XOR operation on the round counter and cipher state (Yeoh et al., 2020). L-box computation is a diffusion table that is arranged using XOR and XNOR gate operations (Biswas et al., 2020). Mix XOR layer is a linear transformation that applies XOR operation among cipher states (Li et al., 2018). Similarly, the mixing layer is represented by XOR operation on each of the individual cipher state bit in sequence (Izadi et al., 2009).

Multiplication function is another mathematical operation implemented in block ciphers. In modulo multiplication, two numbers are multiplied and the modulus function is operated to obtain the result (Lai & Massey, 1991). For mix column, the multiplication between the diffusion matrix and the cipher state array is performed (Li et al., 2018). Mix nibble performs similarly to the mix column function in AES but operates in the form of nibble (Gong et al., 2011). Besides that, mix row is a multiplication operation of a given matrix with the cipher state in the finite field GF(2^4) (Liu et al., 2019).

Matrix operation is also being adopted in algorithms to achieve diffusion. Matrix multiplication is based on the parallel application of a simple involution binary matrix multiplication (Standaert et al., 2004). In matrix transformation, the matrix is transformed into arrays of cipher state bits (Usman et al., 2017). Maximum Distance Separable (MDS) matrix is a linear transformation function that applies a quasi-involutive Feistel-MDS transformation on each plane of a cipher cube with a given MDS code (Berger et al., 2015).

Diffusion property in lightweight block ciphers can also be achieved using the transformation technique. Column-to-Row Transposition is a simple transformation of a cipher state column to a row representation (Lim & Korkishko, 2005). For Feistel Transformation, the component makes use of XOR and rotation functions in a Feistel-like manner (Zhang et al., 2015). Pseudo-Hadamard Transform is an unorthodox linear transformation component that allows the cipher rapid diffusion of small changes in the plaintext or the key over the resulting ciphertext (Massey, 1993). In Reverse order transformation component, the final round output is arranged in reverse order to obtain the ciphertext (Chen et al., 2021).

Permutation is one of the most implemented diffusion components in cryptographic algorithms. Bit shuffle which is similar to bit permutation shifted the cipher state bits to the new position using a permutation table (Bansod et al., 2016). Meanwhile, block shuffle layer takes the 16-bit input and gives the 16-bit shuffled output using a permutation table (Bansod et al., 2017). Permutation layer is the default Feistel permutation that permutes the cipher state according to the Feistel network (FN) structure (Kolay & Mukhopadhyay, 2014). Other than that, keyed permutation is chosen in a key-dependent manner (Knudsen et al., 2010). For word permutation, the permutation method is implemented on a 4-bit cipher state (Wu & Zhang, 2011). Permute nibble function is applied to the nibble positions of the cipher state (Beierle et al., 2019). In permute cells and shuffle cells, the cipher state cells are permuted according to the permutation table (Beierle et al., 2016). Apart from that, swapping operation is applied to diminish the data originality by altering the order of the cipher state bits (Usman et al., 2017).

Another widely adopted diffusion component in lightweight block cipher is the rotation function. Rotation function is implemented by a given number of bits positioned in a specific direction either to the left or to right (Adams, 1997). Besides that, rotate nibble or called word rotation applies 4-bit rotation to the left or right in a specific position (Gong et al., 2011). In 3D rotation, the cipher state is rotated in a specific clockwise direction. For shift function, the cipher state bits is shifted by a specific number of bit position either to the left or to right (Al-Rahman et al., 2018). Using shift row layer, the rows of the cipher state cell array are rotated according to the specific position and direction (Beierle et al., 2016). Meanwhile, row round function adjusts the rows of the cipher state arrays (Kubba

& Hoomod, 2019). Apart from that, LFSR is a shift register whose input bit is a linear function of its previous state (De Cannière et al., 2009).

Besides XOR, other mathematical operations such as byte addition, modulo addition, modulo subtraction, and bitwise AND have been applied in encryption algorithms to achieve confusion property. For byte addition function, a byte-by-byte addition operation between two inputs is processed (Massey, 1993). Modulo addition is a process of adding two numbers, then modulo the result by an integer, and taking the remainder as the final answer (Wheeler & Needham, 1994). In modulo subtraction, the cipher state is manipulated by subtracting the plaintext with the round key in binary operation and taking the modulus output (Aboshosha et al., 2019). Bitwise AND operation compares one data bit to the corresponding bit, where the result is set to 0 if any of the two bits are equal to 0, and set to 1 if both bits are equal to 1 (Beaulieu et al., 2013).

Most cryptographic algorithms adopt substitution components to provide confusion property in the cipher design. Substitution process applies four bits input cipher state to the S-box simultaneously and generates four bits output (Bansod et al., 2016c). In merged substitution, eight 4-bit S-boxes are merged into one component, sharing the same quadratic permutation, thus reducing the resource overhead (Liu et al., 2018). Another substitution method namely balanced block mixer applies one-to-one value mapping on all inputs of the mixer to change the output values (Jha et al., 2020). For sub cell function, S-box is applied to every cell of the cipher state in parallel (Banik et al., 2015). Similarly, sub column operation is a parallel application of S-boxes in the same column of the cipher state (Zhang et al., 2015). On the other hand, sub nibble function implements 4-bit nibble substitution using the S-box (Gong et al., 2011).

On top of the individual diffusion and confusion functions, there exist components that are able to provide both diffusion and confusion properties at once, which are built based on the combination of multiple functions. ARX (add–rotate–XOR) operation relies on modular addition to provide non-linearity while wordwise rotations and XOR provide diffusion (Sehrawat & Gill, 2019). Other than that, an F-function that is comprised of substitution and diffusion layers is being implemented in Feistel network ciphers (Kolay & Mukhopadhyay, 2014). In the round function, multiple components are used at a time that includes XOR and rotation (Hong et al., 2006). Meanwhile, the quarter round function implements XOR, AND, and rotation operations (Kubba & Hoomod, 2019). Weight function consists of the application of rotation and XOR operation as linear operation and S-box as non-linear operation (Kumar et al., 2019). Besides all the above functions, the pseudorandom transformation method is used to construct encryption and decryption transformation tables to achieve confusion and diffusion properties (Dai et al., 2015).

4.3 Comparison of Lightweight Block Cipher Components

In the development of lightweight block cipher, the two most common components used to construct a secure algorithm are substitution and permutation (Banik et al., 2017). These two components contribute to the addition of confusion and diffusion properties in the algorithm. Substitution component replaces a cipher string by mapping a value input to another output using a transformation table such as an S-box. Meanwhile, the permutation component rearranges a cipher string by reordering the positions of each value input to produce an output using a permutation table or other rotation methods. Analyses of the substitution and permutation components are discussed in the following subsections to

obtain the secure cryptographic components that are used in the development of lightweight block cipher.

4.3.1 Comparison of Substitution Component

S-box or substitution box is an important component in the design of block ciphers to obscure the relationship between the key and ciphertext, thus ensuring confusion property (Omrani et al., 2018). An S-box takes an input bit *m*, and transforms it into an output bit *n*, where *n* is not necessarily equal to *m*. These transformations are repeated a few times depending on the number of elements in the S-box. The primary usage of the S-box is to provide a nonlinear element in the block cipher, which increases the complexity of the algorithm to make it more secure (Patil et al., 2019). The introduction of the nonlinearity element from the S-box would effectively resist different types of cryptanalytic attacks. There are many types of substitution functions in lightweight block ciphers such as a single S-box, multiple S-boxes, and multiple S-box sizes as shown in Table 4.2.

Table 4.2: Substitution Function

No.	Algorithm	Rounds	S-box			Max DDT	Max LAT	Maximum Significant Attack Round	
			Size (Bits)	Table	Design			Differential Cryptanalysis	Linear Cryptanalysis
1	ACT	31	Input:4 Output:4	(4,B,D,8,1,6,2,F,A,5,E,3,9,C,7,0)	-	4	4	15	15
2	ANU	25	Input:4 Output:4	(2,9,7,E,1,C,A,0,4,3,8,D,F,6,5,B)	-	4	4	15	12
3	ANU–II	25	Input:4 Output:4	(E,4,B,1,7,9,C,A,D,2,0,F,8,5,3,6)	-	4	4	12	12
4	BORON	25	Input:4 Output:4	(E,4,B,1,7,9,C,A,D,2,0,F,8,5,3,6)	-	4	4	15	15
5	CUBE	15	Input:4 Output:4	(7,A,2,C,4,8,F,0,5,9,1,E,3,D,B,6)	NOEKEON	4	4	9	10
6	DoT	31	Input:4 Output:4	(3,F,E,1,0,A,5,8,C,4,B,2,9,7,6,D)	-	4	4	30	24
7	FeW	32	Input:4 Output:4	(2,E,F,5,C,1,9,A,B,4,6,8,0,7,3,D)	Hummingbird	4	4	11	11
8	GIFT	28	Input:4 Output:4	(1,A,4,C,6,F,3,9,2,D,B,7,5,0,8,E)	-	4	4	14	13
9	GRANULE	32	Input:4 Output:4	(E,7,8,4,1,9,2,F,5,A,B,0,6,C,D,3)	-	4	4	14	14
10	Improved RoadRunneR	10	Input:4 Output:4	(0,8,6,D,5,F,7,C,4,E,2,3,9,1,B,A)	-	4	4	-	-
11	Improved SM4	32	Input:4 Output:4	(0,C,9,D,3,5,B,1,6,E,A,8,7,4,2,F)	-	4	4	-	-
12	KLEIN	12	Input:4 Output:4	(7,4,A,9,1,F,B,0,C,3,2,6,8,E,D,5)	-	4	4	9	9
13	LBC-IoT	32	Input:4 Output:4	(0,8,6,D,5,F,7,C,4,E,2,3,9,1,B,A)	-	4	4	-	-
14	LCA	16	Input:4 Output:4	(8,7,3,C,D,B,4,1,6,A,9,F,0,5,E,2)	-	4	4	-	-
15	LiCi	31	Input:4 Output:4	(3,F,E,1,0,A,5,8,C,4,B,2,9,7,6,D)	-	4	4	12	12
16	LILLIPUT	30	Input:4 Output:4	(4,8,7,1,9,3,2,E,0,B,6,F,A,5,D,C)	-	4	4	17	17
17	Loong	16	Input:4 Output:4	(C,A,D,3,E,B,F,7,9,8,1,5,0,2,4,6)	-	4	4	4	4
18	LRBC	24	Input:4 Output:4	(Biswas et al., 2020)	-	-	-	-	-
19	MANTRA	32	Input:4 Output:4	(2,5,D,A,F,3,4,9,B,0,6,C,8,E,1,7)	-	4	4	16	16
20	MIBS	32	Input:4 Output:4	(4,F,3,8,D,A,C,0,B,5,7,E,2,6,1,9)	mCrypton	4	4	20	20
21	NUCLEAR	25	Input:4 Output:4	(E,4,B,1,7,9,C,A,D,2,0,F,8,5,3,6)	-	4	4	20	16
22	NUX	31	Input:4 Output:4	(E,7,8,4,1,9,2,F,5,A,B,0,6,C,D,3)	-	4	4	20	15
23	NVLC	20	Input:4 Output:4	(1,0,5,3,E,2,F,7,D,A,9,B,C,8,4,6)	-	4	4	-	-
24	Piccolo	25	Input:4 Output:4	(E,4,B,2,3,8,0,9,1,A,7,F,6,C,5,D)	-	4	4	7	7
25	PICO	32	Input:4 Output:4	(1,2,4,D,6,F,B,8,A,5,E,3,9,C,7,0)	-	4	4	24	24
26	PRIDE	20	Input:4 Output:4	(0,4,8,F,1,5,E,9,2,7,A,C,B,D,6,3)	-	4	4	-	-
27	RARE	13	Input:4 Output:4	(D,E,0,8,B,1,9,F,6,3,7,A,2,5,C,4)	-	4	4	-	-
28	RoadRunneR	10	Input:4 Output:4	(0,8,6,D,5,F,7,C,4,E,2,3,9,1,B,A)	-	4	4	5	5
29	SAT_Jo	31	Input:4 Output:4	(1,0,E,9,B,D,6,7,2,F,5,C,4,A,8,3)	-	4	4	-	-
30	SKINNY	32	Input:4 Output:4	(C,6,9,0,1,A,2,B,3,8,5,D,4,E,7,F)	-	4	4	-	-
31	Sriram	27	Input:4 Output:4	(6,3,5,8,1,E,2,B,F,C,9,7,A,0,4,D)	-	4	4	-	-

No.	Algorithm	Rounds	S-box			Max DDT	Max LAT	Maximum Significant Attack Round	
			Size (Bits)	Table	Design			Differential Cryptanalysis	Linear Cryptanalysis
32	TED	26	Input:4 Output:4	(9,4,F,A,E,1,0,6,C,7,3,8,2,B,5,D)	-	4	4	12	12
33	VAYU	31	Input:4 Output:4	(6,3,A,5,C,8,1,B,0,D,9,E,F,2,7,4)	-	4	4	8	8
34	PRINCE	12	Input:4 Output:4	(B,F,3,2,A,C,9,1,6,7,8,0,E,5,D,4)	-	4	4	-	-
35	ILEA	12			PRINCE			-	-
36	PUFFIN	32	Input:4 Output:4	(D,7,3,2,9,A,C,1,F,4,5,E,6,0,B,8)	ICEBERG	4	4	-	-
37	PUFFIN2	34						-	-
38	TWINE	32	Input:4 Output:4	(C,0,F,A,2,B,9,5,8,3,D,7,1,E,6,4)	-	4	4	15	15
39	T-TWINE	36			TWINE			19	19
40	Midori	16	Input:4 Output:4	(C,A,D,3,E,B,F,7,8,9,1,5,0,2,4,6)	-	4	4	7	7
41	CRAFT	31			Midori			18	22
42	MANTIS	14						-	-
43	RECTANGLE	25	Input:4 Output:4	(6,5,C,A,1,E,7,9,B,0,3,D,8,F,4,2)	-	4	4	15	14
44	SR-LED	12			RECTANGLE			-	-
45	HISEC	15	Input:4 Output:4	(F,C,2,7,9,0,5,A,1,B,E,8,6,D,3,4)	-	4	4	-	-
46	DLBCA	15			HISEC			-	-
47	Improved DLBCA	15						-	-
48	OLBCA	22						8	8
49	PRESENT	31	Input:4 Output:4	(C,5,6,B,9,0,A,D,3,E,F,8,4,7,1,2)	-	4	4	20	16
50	µ2	15			PRESENT			6	6
51	EPCBC	32						-	-
52	Hybrid PRESENT & Salsa20	20						-	-
53	Improved GOST	32						-	-
54	Khudra	18						6	6
55	LED	32						6	6
56	PriPresent	31						-	-
57	PRINTcipher	48	Input:3 Output:3	(0,1,3,6,7,4,5,2)	-	-	-	24	24
58	SEA	Variable	Input:3 Output:3	(0,5,6,7,4,3,1,2)	-	-	-	-	-
59	Halka	24	Input:8 Output:8	(Das, 2014)	-	-	-	-	-
60	KHAZAD	8	Input:8 Output:8	(Barreto & Rijmen, 2000)	-	-	-	-	-
61	LWE	3	Input:8 Output:8	(Toprak et al., 2020)	-	-	-	-	-

No.	Algorithm	Rounds	S-box			Max DDT	Max LAT	Maximum Significant Attack Round	
			Size (Bits)	Table	Design			Differential Cryptanalysis	Linear Cryptanalysis
62	SKIPJACK	32	Input:8 Output:8	(NSA, 1998)	-	-	-	-	-
63	VH	10	Input:8 Output:8	(Dai et al., 2015)	-	-	-	4	4
64	FlexAE	Variable	Input:8 Output:8	(Nascimento & Xexéo, 2019)	AES	-	-	9	10
65	DESL	16	Input:6 Output:4	(E,5,7,2,B,8,1,F,0,A,9,4,6,D,C,3, 5,0,8,F,E,3,2,C,B,7,6,9,D,4,1,A, 4,9,2,E,8,7,D,0,A,C,F,1,5,B,3,6, 9,6,F,5,3,8,4,B,7,1,C,2,0,E,A,D)	-	-	-	-	-
66	Blowfish	16	Input:8 Output:32 (4 S-boxes)	(Schneier, 1993)	-	-	-	-	-
67	Kasumi	8	Input:7 Output:7 & Input:9 Output:9	(ETSI, 2014)	-	-	-	-	-
68	QTL	16/20	Input:4 Output:4 (2 S-boxes)	(C,5,6,B,9,0,A,D,3,E,F,8,4,7,1,2) (4,F,3,8,D,A,C,0,B,5,7,E,2,6,1,9)	PRESENT & mCrypton	4/4	4/4	6	6
69	SFN	32	Input:4 Output:4 (2 S-boxes)	(C,A,D,3,E,B,F,7,8,9,1,5,0,2,4,6) (B,F,3,2,A,C,9,1,6,7,8,0,E,5,D,4)	Midori & PRINCE	4/4	4/4	16	16
70	SIT	5	Input:4 Output:4 (2 S-boxes)	(3,F,E,0,5,4,B,C,D,A,9,6,7,8,2,1) (9,E,5,6,A,2,3,C,F,0,4,D,7,B,1,8)	-	4/4	4/4	-	-
71	ICEBERG	16	Input:4 Output:4 (2 S-boxes) & Input:8 Output:8	(D,7,3,2,9,A,C,1,F,4,5,E,6,0,B,8) (4,A,F,C,0,D,9,B,E,6,1,7,3,5,8,2) & (Standaert et al., 2004)	-	-	-	-	-
72	Hummingbird	4	Input:4 Output:4 (4 S-boxes)	(8,6,5,F,1,C,A,9,E,B,2,4,7,0,D,3) (0,7,E,1,5,B,8,2,3,A,D,6,F,C,4,9) (2,E,F,5,C,1,9,A,B,4,6,8,0,7,3,D) (0,7,3,4,C,1,A,F,D,E,6,B,2,8,9,5)	-	-	-	-	-
73	Hummingbird-2	4	Input:4 Output:4 (4 S-boxes)	(7,C,E,9,2,1,5,F,B,6,D,0,4,8,A,3) (4,A,1,6,8,F,7,C,3,0,E,D,5,9,B,2) (2,F,C,1,5,6,A,D,E,8,3,4,0,B,9,7) (F,4,5,8,9,7,2,1,A,3,0,E,6,C,D,B)	-	-	-	-	-
74	mCrypton	12	Input:4 Output:4 (4 S-boxes)	(4,F,3,8,D,A,C,0,B,5,7,E,2,6,1,9) (1,C,7,A,6,D,5,3,F,B,2,0,8,4,9,E) (7,E,C,2,0,9,D,A,3,F,5,8,6,4,B,1) (B,0,A,7,D,6,4,2,C,E,3,9,1,5,F,8)	-	4	4	8	8

No.	Algorithm	Rounds	S-box			Max DDT	Max LAT	Maximum Significant Attack Round	
			Size (Bits)	Table	Design			Differential Cryptanalysis	Linear Cryptanalysis
75	ESF	31	Input:4 Output:4 (8 S-boxes)	(Liu et al., 2014)	Serpent	-	-	-	-
76	CAST	12/16/48	Input:32 Output:32 (8 S-boxes)	(Adams, 1997)	-	-	-	-	-
77	LBlock	32	Input:4 Output:4 (10 S-boxes)	(Wu & Zhang, 2011)	-	4	4	15	15
78	Improved LBlock	32			LBlock			13	13
79	HERMES	30	Input:4 Output:4 (16 S-boxes)	(Malutan et al., 2019)	-	-	-	-	-

4-bit S-box is the most implemented substitution function in lightweight block ciphers. The aim of using a 4-bit S-box in block cipher design is to have less gate count for hardware implementation (Bansod et al., 2018b). This type of S-box is compact and efficient even if the number of encryption rounds has to be increased to maintain the security margins. 4-bit S-box reduces the resources consumed by the hardware implementation of the circuit and meets the initial requirements of lightweight. The S-box has an optimum cycle count and adequate security (Izadi et al., 2009). The cost for the compact S-box is considerably low due to its small area footprint, thus making it low energy consumption (Poschmann et al., 2010). A strong S-box not only makes the design robust but also introduces an avalanche effect (Bansod et al., 2016). The robustness of the 4-bit S-box fulfils the requirement to be resistant against cryptanalytic attacks (Liu et al., 2018).

Referring to the list of algorithms tabulated in Table 4.1, some algorithms do not implement S-box in their block ciphers. Therefore, these algorithms are not included in Table 4.2. KATAN is an example of a block cipher with no S-box implementation that was developed to reduce the burden on the hardware requirements (De Cannière et al., 2009). Non-S-box-based lightweight block ciphers are tuned for optimal performance in hardware and software (Koo et al., 2017). However, non-S-box-based algorithms require the replacement function to provide confusion property to the lightweight block cipher. Examples of functions that are used to replace the S-box are by using MA-box (Multiplication and Addition) and AX-box (Addition and XOR) (Lai & Massey, 1991).

The smallest S-box size found in this literature review is the 3-bit S-box which contains eight elements compared to 16 elements in the 4-bit S-box. An example of a 3-bit S-box implementation is the SEA block cipher that is constructed using a simple algebraic structure (Standaert et al., 2006). Although the size is small, the 3-bit S-box is optimal with respect to linear and differential properties (Knudsen et al., 2010).

One of the uncommon types of S-box identified in Table 4.2 can be found in the DESL algorithm that contains 6-bit input and produces 4-bit output (Leander et al., 2007). This type of S-box is inspired by the DES block cipher. However, the DESL applied only one S-box instead of eight S-boxes in DES to reduce the gate complexity, thus performing better than the DES algorithm.

AES-type 8-bit S-box is argued to be unsuitable for lightweight block cipher due to its large area and power consumption compared to the compact 4-bit S-box (Biswas et al., 2020). However, there are algorithms that used 8-bit S-box such as Halka, KHAZAD, and VH lightweight block ciphers. Other than these, Kasumi block cipher applied 7-bit and 9-bit S-boxes in its encryption algorithm. The reason for using both types of S-boxes is the small area requirement that even a lightweight block cipher can be proposed with these S-boxes (Das, 2014). On top of that, larger S-boxes are more resistant to differential and linear cryptanalysis (Schneier, 1993).

Multiple S-boxes have been applied simultaneously in various cryptographic algorithms. As an example, two S-boxes implemented in SFN have improved the security of the block cipher (Li et al., 2018). For four S-boxes used in QTL, it took only 128 bytes of storage which is considered small enough to be used in very limited computing environments (Lim & Korkishko, 2005). In addition, the implementation of four different

S-boxes is primarily to avoid symmetries when different bytes of the input are equal. Multiple S-boxes along with the diffusion layer in block cipher make cryptanalytic attacks not practical to be executed (Usman et al., 2017). The multiple S-boxes used in block ciphers are carefully chosen to fulfil S-box properties such as no fixed point, completed, best non-linearity, best differential probability, and optimum algebraic order (Wu & Zhang, 2011).

Another important point to highlight is the implementation of existing S-boxes in lightweight block ciphers. The most reused S-boxes in block ciphers are PRESENT, Midori, HISEC, PRINCE, and ICEBERG as shown in Table 4.2. Despite the age of PRESENT, its S-box is still being applied in many algorithms since it is strong with regard to linear and differential properties and has a low area footprint (Poschmann et al., 2010). Meanwhile, Midori S-box has been selected based on its low energy consumption which is needed for low-constraint devices (Beierle et al., 2019). Other than that, HISEC S-box can produce confusion property which gives nonlinearity elements to the block cipher (Al-Dabbagh, 2017). Apart from that, lightweight block cipher can be optimized with respect to latency using the PRINCE S-box (Jha et al., 2020). On the other hand, ICEBERG S-box implemented separate logic gates to generate each output bit of the S-box in which the cost of the S-boxes does not dominate the hardware resources of the data path (Cheng et al., 2008).

The choice of S-box in an algorithm has a huge influence on the result of differential and linear cryptanalysis (Cui & Jin, 2017). From the S-box, Differential Distribution Table (DDT) and Linear Approximation Table (LAT) are generated through specific computations to be applied during the differential and linear cryptanalysis respectively. The

values in both tables would help in determining the number of active S-boxes, thus analyzing the strength of an algorithm against cryptanalytic attacks.

A comparison of substitution function features is presented in Table 4.3. Four types of S-boxes that include small S-box, large S-box, different input and output sizes S-box, and multiple S-boxes are compared to observe the characteristics of each implementation in lightweight block ciphers.

Table 4.3: Comparison of Substitution Function Features

Feature	Small S-box	Large S-box	Different Input and Output Sizes S-box	Multiple S-boxes
Area Footprint	↓	↑	↓ (Small S-box) / ↑ (Large S-box)	↓ (Small S-box) / ↑ (Large S-box)
Avalanche Effect	↑	↑	↑	↑
Compact	✓	×	✓ (Small S-box) / × (Large S-box)	✓ (Small S-box) / × (Large S-box)
Complexity	↓	↑	↓ (Small S-box) / ↑ (Large S-box)	↑
Confusion Property	✓	✓	✓	✓
Cryptanalytic Attacks Resistant	✓	✓	✓	✓
Cycle Count	↓	↑	↓ (Small S-box) / ↑ (Large S-box)	↓ (Small S-box) / ↑ (Large S-box)
Diminish Data Originality	✓	✓	✓	✓
Gate Count	↓	↑	↓ (Small S-box) / ↑ (Large S-box)	↓ (Small S-box) / ↑ (Large S-box)
Implementation Cost	↓	↑	↓ (Small S-box) / ↑ (Large S-box)	↑
Involution	✓	✓	✓	✓
Latency	↓	↑	↓ (Small S-box) / ↑ (Large S-box)	↓ (Small S-box) / ↑ (Large S-box)
Lightweight	✓	×	✓ (Small S-box) / × (Large S-box)	×
Power Consumption	↓	↑	↓ (Small S-box) / ↑ (Large S-box)	↓ (Small S-box) / ↑ (Large S-box)
Robustness	✓	✓	✓	✓
Security	✓	✓	✓	✓
Speed	↑	↓	↑ (Small S-box) / ↓ (Large S-box)	↑ (Small S-box) / ↓ (Large S-box)
Storage Requirement	↓	↑	↓ (Small S-box) / ↑ (Large S-box)	↓ (Small S-box) / ↑ (Large S-box)

Symbols "✓" (Yes) and "×" (No) represent the availability of each feature, while "↑" (High) and "↓" (Low) indicate valuation of the respective feature

The results from the comparison show that the substitution features are dependent on the size of the S-boxes that influence the performance of the substitution functions. Since lightweight block cipher is supposedly designed to be simpler than the conventional algorithm, substitution function such as 4-bit S-box is preferred given its adequate security for resource-constrained devices. The findings of the substitution function features are used in the identification of the secure cryptographic components.

4.3.2 Comparison of Permutation Component

Permutation or linear layer is a core component in any substitution-permutation network block cipher (Chen et al., 2020). The component can significantly influence the security and efficiency of an algorithm. In general, there are two types of permutation components namely permutation and rotation functions. Table 4.4 lists the implementation of permutation functions in lightweight block ciphers that includes the data type, number of permutation bits, and permutation method. In addition, the table listed the type of rotation, number of rotation bits, rotation direction, and rotation parameter. This information provides a broad overview of the various strategies that can be used in identifying the secure cryptographic components.

Table 4.4: Permutation and Rotation Function

No.	Algorithm	Rounds	Permutation			Rotation				Maximum Significant Attack			
										Differential		Linear	
			Data Type	Bits/ Total Bits	Method	Type	Bits/ Total Bits	Direction	Parameter	Rounds	Active S-boxes	Rounds	Active S-boxes
1	μ2	15	Bit	1/16	Formulation	-				6	36	6	36
			Block	16/64	Feistel								
2	ACT	31	Bit	1/64	Table	-				15	30	15	32
3	ANU	25	Bit	1/32	Table	Bit	1/32	Left	3	15	40	12	36
			Block	32/64	Feistel			Right	8				
4	ANU–II	25	Block	32/64	Feistel	Bit	1/32	Left	10	12	39	12	42
								Right	3				
5	BEST-1	12	Byte	8/64	Unspecified	Bit	1/8	Left	1, 3, 4 & 6	-			
6	Blowfish	16	Block	32/64	Feistel	-				-			
7	BORON	25	Nibble	4/16	Unspecified	Bit	1/16	Left	1, 4, 7 & 9	15	40	15	40
8	BRIGHT	32	Block	Varies	Feistel	Bit	1/16	Left	2 & 6	-			
9	CAST	12	Block	Varies	Feistel	Bit	1/32	Left	Varies	-			
10	CHAM	80	Block	16/64	Unspecified	Bit	1/16	Left	1 & 8	-			
11	CRAFT	31	Nibble	4/64	Unspecified	-				18	32	22	32
12	CUBE	15	-			Bit	1/64	3D	Varies	9	33	10	38
13	DESL	16	Bit	1/32	Unspecified	-				-			
			Block	32/64	Feistel								
14	DLBCA	15	Bit	1/16	Unspecified	Bit	1/16	Left	12	-			
			Block	16/32	Unspecified								
15	DoT	31	Byte	8/64	Unspecified	Bit	1/32	Left	25	30	35	24	40
								Right	31				
16	EPCBC	32	Bit	1/64	Formulation	-				-			
17	ESF	31	Bit	1/32	Formulation	Bit	1/32	Left	7	-			
			Block	32/64	Feistel								
18	FeW	32	Byte	8/32	Unspecified	Bit	1/16	Left	1, 5, 9 & 12	11	34	11	34
			Block	32/64	Feistel								
19	FlexAE	Variable	Nibble	Varies	Unspecified	-				9	26	-	-
			Block	Varies	Feistel								
20	GIFT	28	Bit	1/64	Table	-				14	28	13	26
21	GRANULE	32	Nibble	4/16	Unspecified	Bit	1/32	Left	2	14	46	14	46
			Block	32/64	Feistel			Right	7				

No.	Algorithm	Rounds	Permutation			Rotation				Maximum Significant Attack			
										Differential		Linear	
			Data Type	Bits/ Total Bits	Method	Type	Bits/ Total Bits	Direction	Parameter	Rounds	Active S-boxes	Rounds	Active S-boxes
22	Halka	24	Bit	1/64	Table	-				-			
23	HERMES	30	B	1/16	Table	Bit	1/16	Left Right	Formulation	-			
24	HIGHT	32	Byte	8/64	Unspecified	Bit	1/8	Left	1, 2, 3, 4, 6 & 7	-			
25	HISEC	15	Bit Block	1/32 32/64	Unspecified Feistel	Bit	1/32	Left	20	-			
26	Hummingbird	4	-			Bit	1/16	Left	6 & 10	-			
27	Hummingbird-2	4	-			Bit	1/16	Left Right	1, 3 & 8 1	-			
28	Hybrid PRESENT & Salsa20	20	Bit	1/64	Table	Bit	1/32	Left	7, 9, 13 & 18	-			
29	ICEBERG	16	Bit	1/4 1/8 1/64	Table	-				-			
30	IDEA	8.5	Block	16/64	Unspecified					-			
31	ILEA	12	Byte	8/64	Formulation					-			
32	Improved DLBCA	15	Bit Block	1/16 16/32	Unspecified Feistel	Bit	1/16	Left	12	-			
33	Improved GOST	32	Block	32/64	Feistel	Bit	1/32	Left	11	-			
34	Improved IDEA	6.5	Block	16/64	Unspecified	-				-			
35	Improved LBlock	32	Bit Block	1/32 32/64	Unspecified Feistel	Bit	1/32	Left	20	13	32	13	32
36	Improved RoadRunneR	10/12	Bit Block	1/16 32/64	Table Feistel	-				-			
37	Improved Simeck	32/44	Block	16/32 or 32/64	Feistel	Bit	1/32 or 1/64	Left Right	3 5	-			
38	Improved SM4	32	-			Bit	1/16	Left	1, 4, 6 & 9	-			
39	JAC_Jo	32	Block	16/32	Feistel	Bit	1/16	Left	1 & 5	-			
40	Kasumi	8	Block	32/64	Feistel	-				-			

No.	Algorithm	Rounds	Permutation			Rotation				Maximum Significant Attack			
										Differential		Linear	
			Data Type	Bits/ Total Bits	Method	Type	Bits/ Total Bits	Direction	Parameter	Rounds	Active S-boxes	Rounds	Active S-boxes
41	KATAN	254	-			Bit	1/13, 1/19 or 1/25	Right	1	-			
							1/19, 1/29 or 1/39	Left	1				
42	KHAZAD	8	-			-				-			
43	Khudra	18	Nibble	4/16	Feistel	-				6	6	6	6
			Block	16/64									
44	KLEIN	12	Nibble	4/64	Unspecified	-				9	33	9	33
45	KTANTAN	254	-			Bit	1/13, 1/19 or 1/25	Right	1	-			
							1/19, 1/29 or 1/39	Left	1				
46	LBC-IoT	32	Bit	1/16	Table	Bit	1/16	Left	7	-			
			Block	16/32	Feistel								
47	LBlock	32	Nibble	4/16	Unspecified	Bit	1/32	Left	8	15	32	15	32
			Block	32/64	Feistel								
48	LCA	16	Block	32/64	Feistel	Bit	1/32	Left	13	-			
49	LED	32	-			Nibble	4/16	Left	1, 2 & 3	6	37	6	37
50	LiCi	31	Block	32/64	Feistel	Bit	1/32	Left	3	12	39	12	36
								Right	7				
51	LILLIPUT	30	Block	16/64	Table	-				17	36	17	34
					Feistel								
52	Loong	16	Nibble	4/64	Unspecified	-				4	32	4	32
53	LRBC	24	Nibble	4/64	Unspecified	-				-			
					Formulation								
54	LWE	3	Nibble	4/16	Unspecified	-				-			
			Byte	8/16									
			Block	32/64									
55	MANTIS	14	Bit	1/16	Table	-				-			
56	MANTRA	32	Block	32/64	Feistel	Bit	1/32	Left	3	16	32	16	32
								Right	8				
57	mCrypton	12	Bit	1/16	Unspecified	-				8	32	8	32
			Nibble	4/16									

No.	Algorithm	Rounds	Permutation			Rotation				Maximum Significant Attack			
										Differential		Linear	
			Data Type	Bits/ Total Bits	Method	Type	Bits/ Total Bits	Direction	Parameter	Rounds	Active S-boxes	Rounds	Active S-boxes
58	MIBS	32	Nibble	4/16	Table	-				20	30	20	35
			Block	32/64	Feistel								
59	Midori	16	Nibble	4/16	Table	-				7	35	7	35
60	MISTY	8	Block	16/32	Feistel	-				-			
				32/64									
61	NUCLEAR	25	Block	16/32	Feistel	Bit	1/16	Left	3	20	35	16	32
								Right	6				
62	NUX	31	Bit	1/16	Table	Bit	1/16	Left	8	20	36	15	39
			Block	16/32	Feistel			Right	3				
63	NVLC	20	-			Bit	1/8	Left	Formulation	-			
						Byte	8/32	Left	2				
64	OLBCA	22	Block	16/32	Unspecified	Bit	1/16	Left	12	8	14	8	14
65	Piccolo	25	Byte	8/64	Unspecified	-				7	7	7	7
			Block	16/64									
66	PICO	32	Bit	1/16	Table	-				24	48	24	45
67	PRESENT	31	Bit	1/64	Table	-				20	40	16	20
68	PRIDE	20	Bit	1/64	Unspecified	-				-			
69	PRINCE	12	Nibble	4/64	Unspecified	-				-			
70	PRINTcipher	48	Bit	1/16	Formulation	-				24	24	24	24
71	PriPresent	31	Bit	1/64	Table	-				-			
72	PUFFIN	32	Bit	1/64	Table	-				-			
73	PUFFIN2	34	Bit	1/64	Table	-				-			
74	QTL	16	Bit	1/16	Table	-				6	42	6	42
			Block	16/64	Feistel								
75	RARE	13	Bit	1/64	Unspecified	-				-			
76	RC5	0-255	-			Bit	1/16 or 1/32	Left	Formulation	-			
77	RECTANGLE	25	-			Bit	1/16	Left	1, 12 & 13	15	32	14	32
78	RoadRunneR	10	Block	32/64	Feistel	Bit	1/16	Left	1, 12 & 13	5	36	5	36
79	SAFER	Variable	Byte	8/64	Unspecified	-				-			
80	SAT_Jo	31	Bit	1/64	Table	-				-			
81	SEA	Variable	-			Bit	1/24	Left	1	-			
								Right	1				
						Byte	8/24	Left	1				

No.	Algorithm	Rounds	Permutation			Rotation				Maximum Significant Attack			
										Differential		Linear	
			Data Type	Bits/ Total Bits	Method	Type	Bits/ Total Bits	Direction	Parameter	Rounds	Active S-boxes	Rounds	Active S-boxes
82	SFN	32	Bit	1/32	Table	-				16	29	16	29
83	Simeck	32/36/ 44	-			Bit	1/16, 1/24 or 1/32	Left	1 & 5	-			
84	SIMON	32/36/ 42/44	-			Bit	1/16, 1/24 or 1/32	Left	1, 2 & 8	-			
85	SIT	5	Block	16/64	Feistel					-			
86	SKINNY	32	-			Nibble	4/16	Right	1, 2 & 3	-			
87	SKIPJACK	32	Byte	8/2048	Table	-				-			
88	SPARX	24	Block	32/64	Unspecified	Bit	1/16	Left	8	-			
								Right	8				
89	SPECK	22/22/ 23/26/27	-			Bit	1/16, 1/24 or 1/32	Left	2 & 3	-			
								Right	7 & 8				
90	Sriram	27	Bit	1/64	Table	-				-			
91	SR-LED	12	-			Nibble	4/16	Left	1, 2 & 3	-			
92	TEA	64	-			-				-			
93	TED	26	Bit	1/32	Formulation	Bit	1/32	Left	11	12	36	12	33
			Block	32/64	Feistel			Right	7				
94	T-TWINE	36	Nibble	4/64	Table					19	32	19	32
95	TWINE	32	Nibble	4/64	Table	-				15	32	15	32
96	UBRIGHT	22	Block	16/32	Unspecified	Bit	1/16	Left	2 & 6	-			
97	VAYU	31	Bit	1/32	Table	Bit	1/32	Left	3 & 7	8	54	8	36
			Block	32/64	Feistel			Right	3 & 7				
98	VH	10	Bit	1/64	Table	-				4	21	4	24
99	XSX	4	Bit	1/64	Formulation	-				-			

4.3.2.1 Permutation Function

Permutation function is one of the permutation components in a block cipher design. The function consists of four methods that include the Feistel, formulation, table, and unspecified permutations. For the Feistel permutation, it is divided into block and nibble types. Block permutation layer of the Feistel network increased the minimum number of active S-boxes in VAYU algorithm (Bansod et al., 2016c). In LBC-IoT block cipher, two blocks swap of the Feistel structure was used to enhance the diffusion property (Ramadan et al., 2021). Blowfish Feistel network discarded the use of bitwise permutation to introduce simple block operations that are efficient on microprocessors (Schneier, 1993). The block permutation layer in the Feistel design of MIBS algorithm used simple wiring and did not require extra gates (Izadi et al., 2009). On the other hand, nibble permutation of the Feistel structure in Khudra was designed to reduce the number of S-boxes usage (Kolay & Mukhopadhyay, 2014). Although the number of S-boxes usage is reduced, the block cipher did not require an additional diffusion layer to secure the algorithm.

Formulation-based permutation is another option besides the conventional permutation methods. Instead of using predetermined permutation parameters, this method executes on-the-fly mathematical computation to obtain the permutation parameters. Bit formulation permutation was designed to maximize the number of active S-boxes of µ2 block cipher (Yeoh et al., 2020). The bitwise formulation permutation provided fast diffusion without hardware implementation cost in ESF algorithm (Liu et al., 2014). Apart from the bit formulation method, LRBC block cipher implemented round transposition operation using nibble formulation permutation while ILEA applied byte permutation to provide an additional level of security to the algorithms (Biswas et al., 2020).

Permutation table is a common function in the design structure of block ciphers. The table can be structured into different forms of permutation such as bit, nibble, byte, or block to suit the design of the dedicated algorithm. Bit permutation table was implemented in PRESENT given the simplicity of the method which is being adopted in many lightweight block ciphers (Bogdanov et al., 2007). The bit permutation is very easy to be implemented in hardware and software environments of SFN algorithm (Li et al., 2018). Minimal memory requirement of DoT block cipher was achieved using a bit permutation table with high diffusion mechanism (Patil et al., 2019). Cipher bits that are permuted using a permutation table provided complete diffusion with just three rounds of ACT algorithm (Jithendra & Kassim, 2020). The regular bit permutation table helped VAYU to improve the robustness of the algorithm round function (Bansod et al., 2016c). It can be seen that the bit permutation table of PUFFIN is involution and satisfies the property that no two outputs of an S-box are connected to the same S-box in the next encryption round (Cheng et al., 2008). Bit permutation table of Halka block cipher proved that no trails can be constructed thus preventing structural attacks (Das, 2014). ICEBERG permutation table was designed to disturb the bit alignment of the algorithm to provide resistance against cryptanalytic attacks (Standaert et al., 2004). Besides the bit permutation, nibble-based permutation table is also being used in block ciphers such as TWINE algorithm that achieved hardware efficiency while minimizing the hardware-oriented design choices (Suzaki et al., 2011). Midori is another algorithm that applied the nibble-type permutation table to improve diffusion speed and increase the number of active S-boxes in each encryption round (Banik et al., 2015). Other implementations of the permutation table are

SKIPJACK which used byte permutation and LILLIPUT which adopted block permutation to maximize the number of active S-boxes of the algorithms (NSA, 1998).

Table 4.4 shows a type of permutation categorized as an unspecified method. Unlike the Feistel, table, and formulation methods, algorithms implemented this type of permutation did not specify the technique used to generate such permutation parameters. The unspecified permutation can be divided into bit, block, byte, and nibble implementations. Bit permutation provides full dependency after a minimal number of encryption rounds as implemented in PRINTcipher (Knudsen et al., 2010). The bit permutation of an improved LBlock algorithm produced a higher number of active S-box since the method spread the cipher bits into four S-boxes, thus making it secure against cryptographic attacks (Al-Dabbagh & Shaikhli, 2013). Besides the bit permutation method, block permutation is applied in block ciphers such as SIT to diminish the data originality by altering the order of bits block in the algorithm (Usman et al., 2017). In order to improve the diffusion rate of UBRIGHT algorithm, a block permutation was applied in the last step of every encryption round (Sehrawat & Gill, 2020). Similar to the block permutation method, byte permutation provided a systematic way to ensure SAFER algorithm archives the necessary diffusion (Massey, 1993). In addition, to enhance diffusion property, Piccolo block cipher adopted the byte-based permutation between every encryption rounds (Shibutani et al., 2011). Lastly, another method to be highlighted is the nibble permutation that applies a 4-bit permutation operation. Nibble permutation is considered cheap not only in hardware but also in software environments as implemented in mCrypton block cipher (Lim & Korkishko, 2005). Loong reported that the encryption algorithm did not require resources in hardware implementation using the nibble permutation method (Liu et al.,

2019). PRINCE algorithm that adopted nibble permutation guaranteed at least 16 S-boxes are activated in four consecutive encryption rounds (Borghoff et al., 2012). CRAFT attained full diffusion after seven rounds with nibble permutation and resists against differential and linear attacks (Beierle et al., 2019).

A comparison of permutation function features is presented in Table 4.5. Three types of permutation functions that include Feistel permutation, formulation permutation, and table permutation are compared to observe the characteristics of each implementation in lightweight block ciphers.

Table 4.5: Comparison of Permutation Function Features

Feature	Feistel Permutation	Formulation Permutation	Permutation Table
Avalanche Effect	↑	↑	↑
Complexity	↓	↑	↓
Cryptanalytic Attacks Resistant	✓	✓	✓
Diffusion Property	✓	✓	✓
Diminish Data Originality	✓	✓	✓
Efficiency	↑	↑	↑
Gate Count	↓	↓	↓
Implementation Cost	↓	↑	↓
Increase Active S-boxes	✓	✓	✓
Involution	✓	✓	✓
Lightweight	✓	✓	✓
Randomization	↓	↑	↑
Robustness	✓	✓	✓
Security	✓	✓	✓
Simplicity	↑	↓	↑
Speed	↑	↓	↑
Storage Requirement	↓	↓	↑

Symbol "✓" (Yes) represents the availability of each feature, while "↑" (High) and "↓" (Low) indicate valuation of the respective feature

The results from the comparison show that all permutation functions offer many benefits for lightweight block cipher adoption due to their excellent characteristics, especially for software and hardware implementations. Although the strength of each permutation function is not similar, every function has its advantages in increasing the security of lightweight block cipher. This statement is supported by the implementation of permutation functions in the majority of the analysed algorithms as listed in Table 4.4, showing the importance of permutation functions in lightweight block cipher design.

4.3.2.2 Rotation Function

Rotation function is a simple permutation operation adopted in block cipher design due to its various advantages. In general, there are three types of rotations that include one-way, two-way, and multiple directions. On top of the different rotation directions, implementation of the function can be applied in multiple forms of input such as bit, byte, and nibble rotations. Throughout the comparison, two factors influence the output of the function. The first factor is the rotation direction and the other is the rotation parameter. Although the factors have been identified, the best combinations cannot be concluded due to the strength of the cipher output being subject to the overall structure of the dedicated algorithm. However, the advantages of each type of rotation are discussed in this section for a better understanding of the function.

One-way bit rotation to the left direction is the most applied rotation function among lightweight block ciphers as shown in Table 4.4. JAC_Jo used bit rotation in its encryption function and key schedule to harden the differential and linear parameters of the cipher (Shantha & Arockiam, 2018). The P-layer of the RECTANGLE algorithm adopts bit

rotation that makes each column dependent on some other columns, which aimed to provide high diffusion (Zhang et al., 2015). Linear function on each byte of RoadRunneR algorithm provided diffusion using the bit rotation method (Baysal & Şahin, 2015). The bit rotation method is very efficient for hardware implementations as adopted in Simeck block cipher (Yang et al., 2015). Improved LBlock algorithm modified the bit rotation method by 20-bit to produce better diffusion than the 8-bit used in the original LBlock, thus increasing the number of active S-boxes (Al-Dabbagh & Shaikhli, 2013). NVLC block cipher presented a full diffusion feature to prevent shortcut attack accumulation that depends on the bit rotation method (Al-Rahman et al., 2018).

Two-way bit rotation directions that consist of the left and right rotations are often implemented in lightweight block ciphers. By properly using the bit rotations method, ANU–II algorithm can generate the maximum number of active S-boxes in the minimum number of rounds (Dahiphale et al., 2018). Hummingbird-2 adopted left and right bit rotations to increase the resistance of the algorithm against related-key attacks (Engels et al., 2011). The bit rotations method applied in SEA block cipher provided predictable low-cost diffusion within the algorithm (Standaert et al., 2006). Improved Simeck modified the bit rotations method to enhance the avalanche effect of the block cipher (Encarnacion et al., 2020). The bit rotations method needs only wires, thus helping NUX algorithm in reducing the number of GEs of the block cipher (Bansod et al., 2018).

Besides implementing conventional predetermined rotation parameters, there are block ciphers that are used on the fly mathematical computations to obtain the rotation parameters. As an example, the strength of RC5 block cipher depended on the cryptographic properties of the data-dependent bit rotation method (Rivest, 1994). An

advantage of the data-dependent bit rotation method used in CAST algorithm is the immunity against linear and differential attacks (Adams, 1997).

Another type of rotation method in block cipher is the multiple directions rotation known as 3-dimensional rotation which was adopted in CUBE algorithms to add better permutation and diffusion layer in ciphers (Berger et al., 2015). The 3D rotation permutes the cipher bits in multiple directions, which is better than the conventional rotation method that applies one-way and two-way rotation directions.

A comparison of rotation function features is presented in Table 4.6. Three types of rotation functions are highlighted including one-way direction (Left or Right Rotation), two-way directions (left and right rotation), and multiple directions (3D rotation) to observe the characteristics of each implementation in lightweight block ciphers.

Table 4.6: Comparison of Rotation Function Features

Feature	One-way Direction (Left or Right Rotation)	Two-way Directions (Left and Right Rotation)	Multiple Directions (3D Rotation)
Avalanche Effect	↑	↑	↑
Complexity	↓	↓	↓
Cryptanalytic Attacks Resistant	✓	✓	✓
Diffusion Property	✓	✓	✓
Diminish Data Originality	✓	✓	✓
Efficiency	↑	↑	↑
Gate Count	↓	↓	↓
Implementation Cost	↓	↓	↓
Increase Active S-boxes	✓	✓	✓
Involution	✓	✓	✓
Lightweight	✓	✓	✓
Randomization	↓	↓	↑
Robustness	✓	✓	✓
Security	✓	✓	✓
Simplicity	↑	↑	↑
Speed	↑	↑	↑
Storage Requirement	↓	↓	↑

Symbol "✓" (Yes) represents the availability of each feature, while "↑" (High) and "↓" (Low) indicate valuation of the respective feature

The results from the comparison show that the one-way direction rotation is the most implemented function in lightweight block ciphers. However, the comparison indicates that the two-way direction rotation function also shares similar features as the one-way direction rotation. Apart from that, the only difference between the three rotation functions is that the 3D rotation requires a higher storage requirement. Despite the disadvantage of the storage requirement, the 3D rotation managed to provide better randomization of the block cipher output compared to the conventional rotation methods, thus increasing the security of the lightweight algorithm. The adoption of the 3D rotation has increased the confusion and diffusion characteristics of existing block ciphers.

4.4 A Summary of Secure Cryptographic Components

This section highlights the cryptographic components of lightweight block ciphers to answer *Research Question 3* and to fulfil *Research Objective 1*, thus producing *Research Contribution 1*. The analysis of lightweight block cipher components in the previous sections highlighted a few findings that are useful in the development of a secure cryptographic algorithm. Initially, functions of existing lightweight block ciphers were studied in Section 4.2 to identify the methods used by developers in designing cryptographic algorithms. Then, each of the algorithm functions is analysed to observe their strength in order to discover the secure cryptographic components that are selected from the substitution and permutation components of lightweight block ciphers as shown in Figure 4.2.

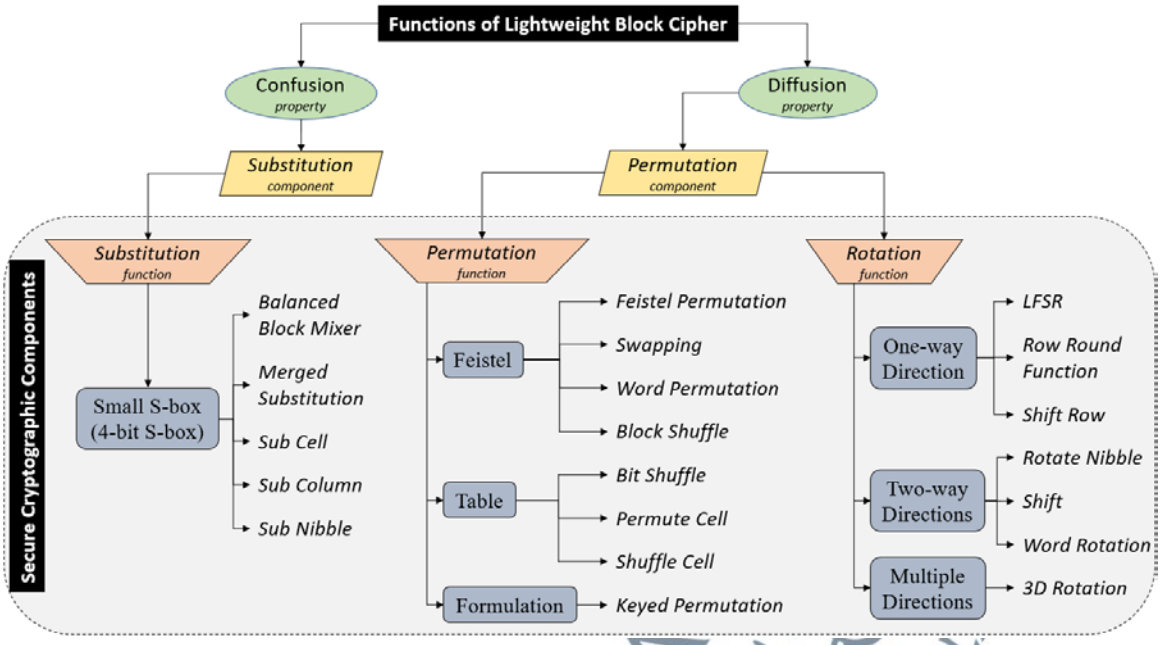

Figure 4.2: Secure Cryptographic Components

In the substitution component, four types of functions were reviewed in Section 4.3.1 which include small S-box, large S-box, different input and output sizes S-box, and multiple S-boxes. However, only the small S-box type is considered suitable for lightweight algorithm implementation to provide confusion property. From the comparison conducted on substitution component features, five functions are selected as the secure cryptographic components namely balanced block mixer, merged substitution, sub cell, sub column, and sub nibble.

Permutation component is divided into two categories, which are the permutation function and rotation function. Although permutation and rotation functions operate in different ways, both functions can provide diffusion property in lightweight block ciphers. There are three types of permutation classification selected from Section 4.3.2.1 as the secure cryptographic components namely Feistel (Feistel permutation, swapping, word

permutation, and block shuffle functions), table (bit shuffle, permute cell, and shuffle cell functions), and formulation (keyed permutation function).

Meanwhile, there are three types of rotation classification selected from Section 4.3.2.2 as the secure cryptographic components. The selected rotation classifications are the one-way direction (LFSR, row round function, and shift row functions), two-way directions (rotate nibble, shift, and word rotation functions), and multiple directions (3D rotation function).

4.5 3-Dimensional Cipher

Looking at the unique features of the 3D rotation that is one of the selected secure cryptographic components from Figure 4.2, further exploration is conducted on the 3-dimensional (3D) cipher in this research. 3D cipher or a cube is an array of plaintext in a block cipher algorithm that is performed on a 3D bits array. The cube is composed of three independent vectors including the *n*-bits length (*X*-axis), *n*-bits width (*Y*-axis), and *n*-bits depth (*Z*-axis) where the block length is divisible by n^2 as shown in Figure 4.3. A cube consists of *n-Slice* where each *Slice* represents a portion of the plaintext array that contains n^2 bits data. 3D cipher design is an approach to improve the security strength of block cipher algorithms. The construction of the 3D design was first proposed on the AES algorithm (Nakahara, 2008). The 3D design increased the AES block size from 256 to 512 bits, expanded the key size from 128 to 512 bits, and extended the encryption rounds from 14 to 22 rounds. Confusion and diffusion properties of the modified AES algorithm increased with the implementation of the 3D cipher.

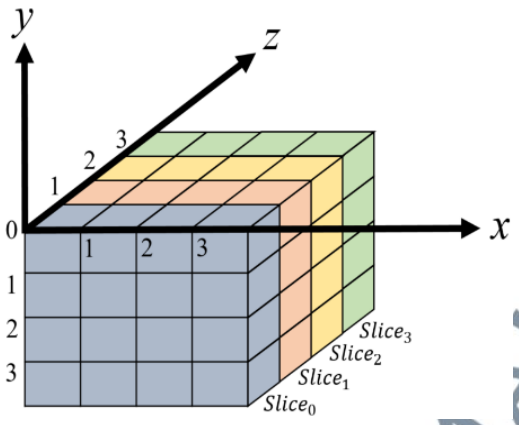

Figure 4.3: Cube

Apart from AES, the 3D design has been implemented in other cryptographic approaches. In previous work, a block cipher with a 3D rotation that produced an effective algorithm and high randomness property was proposed (Suri & Deora, 2011). The method comprises a high number of encryption rounds with 64 iterations and a large key size. Next, an immune-inspired approach in constructing a 3D-AES that passed the statistical tests was introduced (Ariffin et al., 2011). The approach increased the AES block size to 512 bits. In 2014, a unified byte permutation for a 3D block cipher that is capable to make the encryption process faster was developed (Mala, 2014). The design increased the AES block and key sizes to 512 bits. Later on, a non-alternate 3D structure that is secured against linear and differential cryptanalysis was suggested (Wang & Jin, 2018). The technique enlarged the block size up to 512 bits. Most recently, a key schedule algorithm employing 3D hybrid cubes that did not correlate the input and output data was proposed that produced a large key size (Mushtaq et al., 2019).

Even though implementation of the 3D structure would increase the strength of a block cipher algorithm, the gap that must be addressed in 3D structure is the increased number of block sizes, key sizes, and encryption rounds that can reduce the efficiency of the algorithm. Therefore, the aim of the research is to propose a new lightweight block cipher by enhancing the existing 3D design to increase the security and efficiency performances of the algorithm.

4.6 Chapter Summary

In order to design a secure lightweight block cipher, this chapter listed secure cryptographic components based on substitution and permutation components that can be used as guidance for cryptographic algorithm developers. Cryptographic functions from existing algorithms were compared before being selected as the secure cryptographic components based on their security strength.

One of the solutions to improve the security strength of the lightweight block cipher is by adopting the 3D rotation method which is one of the identified secure cryptographic components. The implementation of the 3D rotation would increase the confusion and diffusion properties of block ciphers. This research focused on enhancing the existing 3D design structure to propose a better solution for lightweight block cipher without reducing the efficiency of the algorithm.

The information gathered from the findings are used in the following chapters to answer all of the research questions in order to achieve the research objectives. In the following Chapter 5, a high-performance algorithm called RECTANGLE is discussed to observe its strengths and weaknesses.