Comparing Constraint Programming, Mixed Integer Linear Programming and Satisfiability in Differential Cryptanalysis for Lightweight Ciphers

Gilbert Khonstantine
School of Physical and Mathematical Sciences, Nanyang Technological University

Assoc. Prof. Thomas Peyrin
David Gerault
School of Physical and Mathematical Sciences, Nanyang Technological University

Abstract - Cyber threats have been one of the most important issues to prevent information leakage. As a result, ciphers have been implemented as one form of data protection. Lightweight ciphers are widely utilized to encrypt day-to-day information. Thus, the security of a lightweight cipher needs to be assured. This paper will discuss the differential attack on symmetric key Add-Rotate-XOR (ARX) Cipher and Substitution-Permutation Network (SPN) Cipher by using Constraint Programming (CP), Mixed Integer Linear Programming (MILP) and Satisfiability (SAT). We will scrutinize their solving time as they calculate the minimum number of active S-box(es) for SPN cipher and the maximum differential probability for ARX cipher. In addition, we will also observe time differences applying the SAT solver and Picat solver translated from CP language. The effect of implementing the Matsui's Algorithm as one of the constraints will also be observed. An ARX and an SPN cipher that is used is SPECK and GIFT subsequently.

Keywords – CP, SAT, MILP, SPECK, GIFT, Matsui's Algorithm, Differential Cryptanalysis

1 INTRODUCTION

1.1 Ciphers

Ciphers are a way of disguising a plaintext between 2 parties so that the third party is not able to read it. In this paper, we will discuss lightweight block ciphers, ARX and SPN cipher. A block cipher is an encryption algorithm that encrypts a block of plaintext at a time, rather than encrypting one bit at a time. Lightweight here means it needs less computational power compared to block cipher. ARX and SPN cipher are two of many lightweight block ciphers. An ARX cipher means that the encryption algorithm includes addition, rotation, and xor operation. While an SPN cipher is a cipher that has block size substitution and bitwise permutation encryption algorithm. One example of ARX and SPN cipher that will be used in this paper is SPECK and GIFT subsequently.

1.1.1 GIFT [1]

GIFT is one example of many SPN ciphers. GIFT has 2 versions, GIFT-64 which is for 64-bit long encryption and GIFT-128 for 128 bits. Both versions have the key of length 128 bits. GIFT cipher's encryption consists of three steps:

1. Subcells

The encryption starts from applying S-box to every 4 bits of the plaintext for every round. An s-box is a one-to-one function that maps a block of plaintext to a block of ciphertext. The S-box for GIFT cipher is using the hexadecimal code shown below (Figure 1). One can summarise the behavior of S-box into the Difference Distribution Table (DDT). Each entry of DDT (Figure 6) represents the occurrence frequency of output difference, given the input difference.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$GS(x)$	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

Figure 1: GIFT S-box

2. Bitwise Permutation

The next step of encryption is performing bit-wise permutation to the current ciphertext after going through the previous step. Bitwise permutation has the following function

$b_{P(i)} \leftarrow b_i$, $\forall i \in \{0, ..., r-1\}$. Where r is the block size.

The table below shows how bitwise permutation is performed for GIFT-64 and GIFT-128 respectively.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{64}(i)$	0	17	34	51	48	1	18	35	32	49	2	19	16	33	50	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{64}(i)$	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P_{64}(i)$	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P_{64}(i)$	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

Figure 2: bitwise permutation for GIFT-64

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

Figure 3: bitwise permutation for GIFT-128

3. AddRoundKey

The final step is adding the round key and round constants to the current state of ciphertext. The r/2-bit round key is extracted from the key state, then partition the r/2-bit round key into 2 s-bit words, where s is either 16 (for GIFT64) or 32 (for GIFT128). For GIFT-64, the first half of the round key will be XORed to the ciphertext with index 4i+1. Subsequently, the rest of the round key will be XORed to the ciphertext with index 4i, where i = {0, ...,15}. While for GIFT-128, the first half of the round key will be XORed to the ciphertext with index 4i+2. Subsequently, the rest of the round key will be XORed to the ciphertext with index 4i+2, where i = {0, ...,31}. Both versions have the same round constant, 1 and C = $c_5c_4c_3c_2c_1c_0$. Then XOR round constant into the ciphertext at index r-1,19,15,11,7 and 3 respectively.

Determining the key schedule. For both versions of GIFT, the round constant and the key schedule are identical. The difference exists when extracting the round key. For GIFT-64, the round key is extracted from two 16-bit key states. While for GIFT-128, the round key is extracted from four 16-bit key states. Then updating the key state as follows,

$k_7||k_6||...||k_1||k_0 \leftarrow k_1 \gg 2||k_0 \gg 12||...||k_3||k_2$,

where $\gg$ i mean i bits rotation to the right.

Determining round constant. The round constant is updated by using this function,

$(c_5,c_4,c_3,c_2,c_1,c_0) \leftarrow (c_4,c_3,c_2,c_1,c_0,c_5 \oplus c_4 \oplus 1)$.

Initializing the first round constant to zero, the round constant then updated before being used for the next round.

The following table is the round constants respectively to their own round.

Rounds	Constants
1 - 16	01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E
17 - 32	1D,3A,35,2B,16,2C,18,30,21,02,05,0B,17,2E,1C,38
33 - 48	31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04

Figure 4: Round constant for GIFT cipher

In conclusion, the table below illustrates how the three steps are integrated together and the subsequent table is the Difference Distribution Table (DDT) for GIFT.

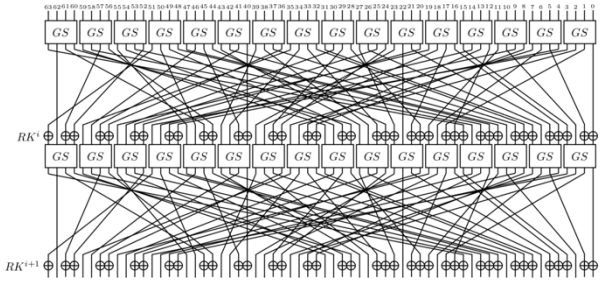

Figure 5: 2 rounds of GIFT-64 encryption

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

Figure 6: GIFT DDT Table

1.1.2 SPECK [2]

Speck is a lightweight cipher made by National Security Agency (NSA) and it is part of Add-Rotate-XOR (ARX) cipher. SPECK has 10 variants, with 2n block size and key size mn, where n is the word size and m are the key words. The encryption for SPECK started from partitioning a plain text into 2 equal bits, (x,y). Then it follows the following function. The output of the function is the ciphertext for that particular round.

$R^k(x,y) = (((x \gg \alpha) \boxplus) \oplus k, (y \ll \beta) \oplus ((x \gg \alpha) \boxplus y) \oplus k)$

where k is the round key, $\gg \alpha$ is rotate α bits to the right and $\ll \beta$ is rotate β bits to the left, $\oplus$ is xor operation, which takes 2 binary input and returns 0 if both outputs have the same value, return 1 otherwise, and $\boxplus$ is modular addition, it adds two inputs and then modulo by 2^n, where n is the word size. Figure 8 (left) shows how the SPECK cipher

encryption algorithm works starting after the plaintext is partitioned. The figure below shows the parameters that are available for SPECK's variants.

Block Size $2n$	Key Size mn	Word Size n	Key Words m	Rounds T	α	β
32	64	16	4	22	7	2
48	72	24	3	22	8	3
	96		4	23	8	3
64	96	32	3	26	8	3
	128		4	27	8	3
96	96	48	2	28	8	3
	144		3	29	8	3
128	128	64	2	32	8	3
	192		3	33	8	3
	256		4	34	8	3

Figure 7: SPECK parameters

Determining the key schedule. For SPECK cipher, the round function is used to generate round keys. Suppose that K is the master key where K = (l_{m-2}, ..., l_0, k_0), then for the next round l and k are defined as:

$l_{i+m-1} = (k_i \boxplus (l_i \gg \alpha)) \oplus i$
$k_{i+1} = (k_i \ll \beta) \oplus l_{i+m-1}$, $i \in \{0, ..., t-1\}$, t is the number of rounds.

Figure 8 (right) shows how the key schedule is determined and integrated with the encryption function, R^i.

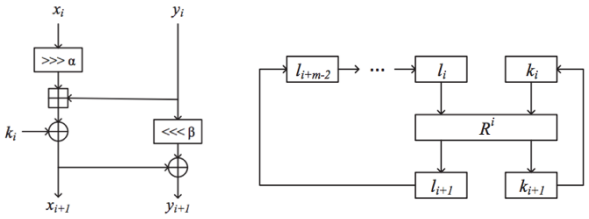

Figure 8: SPECK encryption and key schedule.

1.2 Differential Attack [4]

Differential Attack is one of many attacks on a cipher. This method is a chosen plaintext attack, in other words, the attacker is able to choose the input differences (ΔX) and scrutinize the output differences (ΔY) when attempting to derive the key of the cipher. The idea of this attack is to analyze the input and output differences with the highest probability in order to get the key. In this paper, for GIFT cipher, we will calculate the number of minimum active s-boxes. While for SPECK, we will calculate the maximum differential probability. Differential probability is the number of occurrences (ΔX, ΔY) divided by the total number of possible pairs.

1.3 Matsui's Algorithm [3]

This algorithm has the input of the optimal differential probability for k-1 round and has the output of the differential probability for the nth round.

Let P_k be the probability for round k and P_{k-1} for round k-1. Suppose x is an estimator for the probability of round k. Then, the idea of Matsui's algorithm is to utilize the probability P_{k-1} and x, such that we can obtain P_k. By satisfying the inequality, $P_{k-1}P_k \geq x$.

1.4 Constraint Programming

Constraint Programming (CP) is one of many methods used to solve the optimization problem. The basic model of CP consists of 3 parts; initialization, constraints and objective function. The software that is used for these findings is Minizinc.

2 Modeling

In this paper, we will calculate GIFT64's minimum number of active s-boxes and maximum differential probability for SPECK32. Both will be done for the related-key model. A cipher with the related key is when there is a mathematical relationship that related too the key is known by the attacker,

2.1 CP Minizinc GIFT64 Model

2.1.1 GIFT64 Related-Key Model

First, initialize the variables:

- GIFT64 DDT table (DDT);
- Bitwise permutation order of GIFT64 (perm);
- Other related variables, such as initial differential (delta), differential after DDT (DSB), and differential after permutation (DX), active S-box counter (sbox_counter).
- delta, DSB, and DX are in binary. They are 2-dimensional arrays where the number of rows is the number of rounds when performing differential attack;

Next, we move on to a set of constraints as attached in Appendix A.1 along with the code.

Finally, we set the objective function to minimize sbox_counter.

2.1.2 GIFT64 Related-Key with Matsui's Algorithm Model

This model uses the same constraint as previous one, except we add a constraint and introduce new variables, they are:

- A parameter that is used to estimate the minimum number of active s-box for the next round (estimator);
- An array that consists of the minimum number of active s-boxes for the previous k-1 rounds (bound).

The constraint added is:

- The sum of sbox_counter and bound is lesser or equal to the estimator

Note that this estimator will be unsatisfiable if it is too low, using the previous round of estimator and keep adding 1 each time is one way to find the minimum number of active s-boxes for round k. The complete constraints are attached in Appendix A.2 along with the code.

2.2 CP Minizinc SPECK32 Model

2.2.1 SPECK32 Related-Key Model

First, initialize the variables, note that SPECK cipher partition its plaintext into 2 equal sized plaintexts. Then we initialize the variables:

- The lefthand side of the initial delta from the partition (LD);
- LD after goes through bitwise rotation (LDR);
- The righthand side of the initial delta from the partition (RD);
- RD after bitwise rotation is performed (RDR);
- The result of modular addition between LDR and RD is stored as MA;
- The result of XOR between MA and RDR is stored as RXOR;
- LD, LDR, RD, RDR, MA, and RXOR are 2-dimensional arrays with the number of rows is the number of rounds when performing differential attack;
- mask, this will be useful when calculating the differential probability. The mask has 16 entries as 1, except for the first element, it is 0;
- DP is a float, used to calculate the maximum differential probability.

Next, we move on to a set of constraints as attached in Appendix A.3 along with the code.

Then our objective function is to maximize DP.

2.2.2 SPECK32 Related-Key Model with Matsui's Algorithm

This model uses the same constraint as previous one, except we add a constraint and introduce new variables, they are:

- A parameter that is used to estimate the maximum differential probability for the next round (estimator);
- An array that consists of the maximum differential probability for the previous k-1 rounds (bound).

The constraint added is:

- The product of bound and DP is not greater than the estimator.

Note that this estimator will be unsatisfiable if it is too low, using the previous round of estimator and keep multiplying 2^{-1} each time is one way to find the maximum differential probability for round k. The complete constraints are attached in Appendix A.4 along with the code.

3 Findings

In this section, we will compare the results of the related-key model with and without Matsui's algorithm, as well as comparing the results that solving by using Picat solver for Constraint Programming and Chuffed solver from Minizinc. We are comparing the number of rounds that the solver is able to reach within one hour or if the solver is able to reach 12 rounds within one hour. We are using the 2.3 GHz Intel Core i5.

3.1 GIFT64

Below is the table that compares 2 models with 2 different solvers. Solving time is measured in seconds, the lower the better.

No. of rounds	Minimum number of active S-boxes	GIFT Related-Key Model cumulative time (s)		GIFT Related-Key Model with Matsui's Algorithm cumulative time (s)	
		Chuffed	Picat CP	Chuffed	Picat CP
1	1	0.126	0.163	0.134	0.163
2	2	0.288	0.659	0.261	0.468
3	3	0.464	1.564	0.394	1.087
4	5	2.184	2.622	0.583	2.072
5	7	19.494	4.459	4.161	3.418
6	10	2,031	6.98	137.161	5.646
7	13	-	12.77	-	10.78
8	16	-	26.2	-	18.7
9	18	-	57.49	-	35.66
10	20	-	73.09	-	77.56
11	22	-	138.4	-	118.66
12	24	-	231.2	-	167.41

Figure 9: Table of comparison Chuffed and Picat CP with 2 different GIFT models

3.2 SPECK32

Below is the table that compares 2 models with 2 different solvers. Solving time is measured in seconds, the lower the better.

		SPECK Related-Key	SPECK Related-Key Model with

No. of rounds	Maximum Differential Probability	Model cumulative time (s)		Matsui's Algorithm cumulative time (s)	
		Chuffed	Picat CP	Chuffed	Picat CP
1	2^{0}	0.156	0.127	0.169	0.162
2	2^{-1}	0.33	0.556	0.379	0.347
3	2^{-3}	0.592	1.378	0.956	0.757
4	2^{-5}	2.792	2.78	9.156	1.764
5	2^{-9}	227.79	5.061	64.84	3.934
6	2^{-13}	-	11.43	-	47.63
7	2^{-18}	-	64.99	-	324.2
8	2^{-24}	-	316.2	-	1029

Figure 10: Table of comparison Chuffed and Picat CP with 2 different SPECK models

4 Comparing with MILP and SAT

In this section, we will refer to the work of my peers, Inggriany Dwitami and Tan Chen Hui, for the result of solving the same problem as this paper by using SAT solver and MILP solver respectively. We are comparing the number of rounds that the solver is able to reach within one hour or if the solver is able to reach 15 rounds within one hour. For MILP, Tan Chen Hui is using the Gurobi solver, while for SAT, Inggriany Dwitami is using the Cryptominisat solver. To save up some space, in this section we will denote Gurobi as MILP, while Cryptominisat as SAT. We are using the 2.3 GHz Intel Core i5.

4.1 GIFT64

4.1.1 Comparing CP, MILP, SAT for GIFT64 Related-Key Model

Below is the table that compares the solving time when finding the minimum number of active s-boxes with 4 different solvers by using GIFT related-key model. Solving time is measured in seconds, the lower the better.

No. of Rounds	Minimum number of active S-boxes	Solvers' Solving Time (s)			
		Chuffed	Picat CP	MILP	SAT
1	1	0.126	0.163	0.02	0.09
2	2	0.288	0.659	0.09	0.091
3	3	0.464	1.564	0.16	0.107
4	5	2.184	2.622	2.19	0.134
5	7	19.494	4.459	3.59	0.183
6	10	2,031	6.98	7.11	0.512
7	13	-	12.77	24.7	2.055
8	16	-	26.2	140.54	7.954
9	18	-	57.49	561.26	15.00
10	20	-	73.09	1091.6	25.12
11	22	-	138.4	2068.3	43.88
12	24	-	231.2	-	79.27
13	26	-	436.2	-	131.4
14	28	-	703.2	-	184.2
15	30	-	1082.2	-	249.5

Figure 11: Table of comparison between Chuffed, Picat CP, Gurobi (MILP), and Cryptominisat (SAT) for GIFT64 Related-key model

4.1.1 Comparing CP, MILP, SAT for GIFT64 Related-Key Model with Matsui's Algorithm

Below is the table that compares the solving time for finding the minimum number of active s-boxes with 4 different solvers by using GIFT related-key model with Matsui's Algorithm. Solving time is measured in seconds, the lower the better.

No. of Rounds	Minimum number of active S-boxes	Solvers' Solving Time (s)			
		Chuffed	Picat CP	MILP	SAT
1	1	0.134	0.163	0.01	0.02
2	2	0.261	0.468	0.03	0.03
3	3	0.394	1.087	0.05	0.04
4	5	0.583	2.072	0.08	0.06
5	7	4.161	3.418	0.25	0.124
6	10	137.161	5.646	3.02	0.512
7	13	-	10.78	4.42	1.765
8	16	-	18.7	95.64	10.37
9	18	-	35.66	237.4	13.89
10	20	-	77.56	606.3	18.19
11	22	-	118.66	771.45	21.15
12	24	-	167.41	2352.5	22.98
13	26	-	197.35	-	25.32
14	28	-	236.34	-	28.13
15	30	-	265.43	-	30.76

Figure 12: Table of comparison between Chuffed, Picat CP, Gurobi (MILP), and Cryptominisat (SAT) for GIFT64 Related-key model with Matsui's Algorithm

4.2 SPECK32

4.2.1 Comparing CP, MILP, SAT for SPECK32 Related-Key Model

Below is the table that compares the solving time when finding the maximum differential probability with 4 different solvers by using SPECK related-key model. Solving time is measured in seconds, the lower the better.

		Solvers' Solving Time (s)

No. of Rounds	Maximum Differential Probability	Chuffed	Picat CP	MILP	SAT
1	2^{0}	0.156	0.127	0.00	0.007
2	2^{-1}	0.33	0.556	0.14	0.028
3	2^{-3}	0.592	1.378	0.44	0.078
4	2^{-5}	2.792	2.78	1.59	0.254
5	2^{-9}	227.79	5.061	15.3	1.984
6	2^{-13}	-	11.43	319.69	9.182
7	2^{-18}	-	64.99	-	82.91
8	2^{-24}	-	316.2	-	3092

Figure 13: Table of comparison between Chuffed, Picat CP, Gurobi (MILP), and Cryptominisat (SAT) for SPECK32 Related-key model

4.2.1 Comparing CP, MILP, SAT for SPECK32 Related-Key Model with Matsui's Algorithm

Below is the table that compares the solving time when finding the maximum differential probability with 4 different solvers by using SPECK related-key model with Matsui's Algorithm. Solving time is measured in seconds, the lower the better.

No. of Rounds	Maximum Differential Probability	Solvers' Solving Time (s)			
		Chuffed	Picat CP	MILP	SAT
1	2^{0}	0.169	0.162	0.01	0.007
2	2^{-1}	0.379	0.347	0.15	0.025
3	2^{-3}	0.956	0.757	0.42	0.067
4	2^{-5}	9.156	1.764	1.86	0.156
5	2^{-9}	64.84	3.934	31.44	0.935
6	2^{-13}	-	47.63	1083.2	8.966
7	2^{-18}	-	324.2	-	121.9
8	2^{-24}	-	1029	-	-

Figure 14: Table of comparison between Chuffed, Picat CP, Gurobi (MILP), and Cryptominisat (SAT) for SPECK32 Related-key model with Matsui's Algorithm

5 Conclusion and Further Direction of Research

From part 4, it can be concluded that cryptominisat solver works very well on solving for SPN cipher, when finding the minimum number of active s-boxes, compared to other solvers. While the picat solver is dominating other solvers when finding the maximum differential probability, compared to other solvers.

Also, it can be noted that Matsui's Algorithm lessens the time when finding the minimum number of active s-boxes (SPN cipher) but it increases the time when finding the maximum differential probability (ARX cipher).

The findings on this paper are just one case of many cases, therefore it gives a space to prove that Matsui's algorithm will decrease the time needed when it comes to differential attack for SPN cipher, but it increases the time when it comes to differential attack for ARX cipher. Also, proving that the SAT method is possibly the best method to perform an attack on SPN cipher compared to CP or MILP method.

6 Acknowledgment

I would like to thank my supervisor, Assoc. Prof. Thomas Peyrin, and mentor, David Gerault, for this URECA research paper. They give me motivation and ideas along with this research. Also, I would like to thank my peers for their findings on part 4, Inggriany Dwitami and Tan Chen Hui, without them this research might not be complete. Finally, I thank the URECA Department for giving me such an amazing research experience.

7 References

[1] Banik, S., Pandey, S. K., Peyrin, T., Sasaki, Y., Sim, S. M., & Todo, Y. (2017, September). GIFT: a small present. In *International Conference on Cryptographic Hardware and Embedded Systems* (pp. 321-345). Springer, Cham.

[2] Song, L., Huang, Z., & Yang, Q. (2016, July). Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In *Australasian Conference on Information Security and Privacy* (pp. 379-394). Springer, Cham.

[3] Biryukov, A., & Velichkov, V. (2014, February). Automatic search for differential trails in ARX ciphers. In *Cryptographers' Track at the RSA Conference* (pp. 227-250). Springer, Cham.

[4] Heys, H. M. (2002). A tutorial on linear and differential cryptanalysis. *Cryptologia*, *26*(3), 189-221.

[5] Lipmaa, H., & Moriai, S. (2001, April). Efficient algorithms for computing differential properties of addition. In *International Workshop on Fast Software Encryption* (pp. 336-350). Springer, Berlin, Heidelberg.

Appendix

A. Constraints for the Minizinc Model

A.1 Constraints for GIFT64 Related-key Model

The following are the set of constraints used for GIFT64 related-key model.

- The sum of initial differentials (delta) must be a positive integer, otherwise, we would not get any information from the differential attack;
- For every 4 bits of the delta, it is referred to DDT table then assign it to DSB, since S-box is not linear, DDT table is used as a reference to find the differential path;
- Permutate DSB accordingly to the bitwise permutation order of GIFT then assign it to DX;
- Assign the next row of the delta to be equal to DX, DX of the current round is the delta for the next round;
- sbox_counter is the cumulative sum of all active s-boxes in the differential path. An active s-box is when an s-box gives a non-zero output. Therefore, the sbox_counter will increase by 1 for every non-zero 4 bits of DSB.

The following are the constraints in Minzinc code, accordingly with the points above:

- sum([delta[0,j]|j in 0 ..63]) > 0;
- forall(r in 0..n-1)(
  forall(i in 0..15)(
  table(array1d(0..7,[delta[r,4*i+3],delta[r,4*i+2],delta[r,4*i+1],delta[r,4*i],DSB[r,4*i+3],DSB[r,4*i+2],DSB[r,4*i+1],DSB[r,4*i]]),DDT))
  );
- constraint forall(r in 0..n-1)(forall(i in 0..63)(
  DX[r,perm[i]]=DSB[r,i]));
- constraint forall(r in 0..n-1)(forall(i in 0..63)(
  DX[r,i]=delta[r+1,i]));
- constraint sbox_counter = sum(r in 0..n-1, i in 0..15) (
  DSB[r,4*i+3]+DSB[r,4*i+2]+DSB[r,4*i+1] + DSB[r,4*i] > 0);

A.2 Constraints for GIFT64 Related-key Model with Matsui's Algorithm

The following are the set of constraints used for the GIFT64 related-key model with Matsui's algorithm.

- The sum of initial differentials (delta) must be a positive integer, otherwise, we would not get any information from the differential attack;
- For every 4 bits of the delta, it is referred to DDT table then assign it to DSB, since S-box is not linear, DDT table is used as a reference to find the differential path;
- Permutate DSB accordingly to the bitwise permutation order of GIFT then assign it to DX;
- Assign the next row of the delta to be equal to DX, DX of the current round is the delta for the next round;
- sbox_counter is the cumulative sum of all active s-boxes in the differential path. An active s-box is when an s-box gives a non-zero output. Therefore, the sbox_counter will increase by 1 for every non-zero 4 bits of DSB;
- The product of bound and DP is not greater than the estimator.

The following are the constraints in Minzinc code, accordingly with the points above:

- sum([delta[0,j]|j in 0 ..63]) > 0;
- forall(r in 0..n-1)(
  forall(i in 0..15)(
  table(array1d(0..7,[delta[r,4*i+3],delta[r,4*i+2],delta[r,4*i+1],delta[r,4*i],DSB[r,4*i+3],DSB[r,4*i+2],DSB[r,4*i+1],DSB[r,4*i]]),DDT))
  );
- constraint forall(r in 0..n-1)(forall(i in 0..63)(
  DX[r,perm[i]]=DSB[r,i]));
- constraint forall(r in 0..n-1)(forall(i in 0..63)(
  DX[r,i]=delta[r+1,i]));
- constraint sbox_counter = sum(r in 0..n-1, i in 0..15) (
  DSB[r,4*i+3]+DSB[r,4*i+2]+DSB[r,4*i+1] + DSB[r,4*i] > 0);
- constraint forall(i in 0..n-2)(sbox_counter + bound[n-i-2]<= estimator);

A.3 Constraints for SPECK32 Related-key Model

The following are the functions and constraints used for SPECK32 related-key model.

First, we define functions that are useful to calculate the differential probability.

- myxor, which is a xor function for 2 variables;
- XOR, which accepts 3 Boolean values as input and gives a Boolean value as the output;
- AND, a logical operator and;
- eq, a function that takes 3 inputs then returns 1 if all the 3 inputs have the same value, 0 otherwise.

- neq, the negation of eq.

Next, we move on to a set of constraints as follows:

- One of the sums of initial differentials (LD or RD) must be positive, otherwise, the differential probability will be 0;
- Rotate LD 7 bits to the right then store as LDR;
- Rotate RD 2 bits to the left, then store as RDR;
- MA in this round is the LD for the next round, thus store it in the next row of LD;
- RXOR in this round is RD for the next round, thus store it in the next row of RD;
- The XOR result between MA and RDR is assigned to RXOR;
- For Modular Addition constraint, we refer to the paper written by Helger Lipmaa et.al.[5], which takes input α (LDR) and β (RD), output γ (MA). The constraint is as follows:
  $eq(\alpha \ll 1, \beta \ll 1, \ \gamma \ll 1) \cap$
  $\big(myxor(\alpha, \beta, \gamma)\ XOR\ (\beta \ll 1)\big) = 0$;
- Store DP as $2^{-W_h(\text{neq(LDR, RD, MA) AND mask})}$, where W_h is the sum of its input.

The following are the constraints in Minizinc code, starting with the functions and then the constraints:

1. Functions:

- function var 0..1: myxor(var 0..1 :x, var 0..1 :y) = ((x+y)=1);
- function var int: XOR(var 0..1 :x, var 0..1 :y, var 0..1 :z) = (x+y+z) mod 2;
- function var bool: AND(var int:x, var int:y) = (x==y) /\ (x!=0) ;
- function var int:eq(var int:x,var int:y, var int:z) = if x==y /\ y==z /\ x== z then 1 else 0 endif ;
- function var int:neq(var int:x,var int:y, var int:z) = if x==y /\ y==z /\ x== z then 0 else 1 endif ;

2. Constraints:

- constraint sum([RD[0,j]|j in 0 ..15]) > 0 \/ sum([LD[0,j]|j in 0 ..15]) > 0;
- constraint forall(r in 0..n-1)(forall(i in 0..15)(LDR[r,(i+7) mod 16] = LD[r,i]));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(RDR[r,i] = RD[r,((i+2) mod 16)]));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(MA[r,i] = LD[r+1,i]));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(RD[r+1,i] = RXOR[r,i]));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(RXOR[r,i] = myxor(RDR[r,i],MA[r,i])));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(AND((eq(AND(MA[r,(i+1) mod 16],mask[15-i]) , AND(LDR[r,(i+1) mod 16],mask[15-i]), AND(RD[r,(i+1) mod 16],mask[15-i]))) , (myxor(XOR(MA[r,i],LDR[r,i],RD[r,i]) , AND((RD[r,(i+1) mod 16]),mask[15-i])))) == 0));
- constraint DP = sum(r in 0..n-1,i in 0 .. 15)(AND(neq(MA[r,i],LDR[r,i],RD[r,i]), mask[i]));

A.4 Constraints for SPECK32 Related-key Model with Matsui's Algorithm

The following are the functions and constraints used for SPECK32 related-key model with Matsui's algorithm.

First, we define functions that are useful to calculate the differential probability.

- XOR, which accepts 3 Boolean values as input and gives a Boolean value as the output;
- myxor, which is a xor function for 2 variables;
- AND, a logical operator and;
- eq, a function that takes 3 inputs then returns 1 if all the 3 inputs have the same value, 0 otherwise.
- neq, the negation of eq.

Next, we move on to a set of constraints as follows:

- One of the sums of initial differentials (LD or RD) must be positive, otherwise, the differential probability will be 0;
- Rotate LD 7 bits to the right then store as LDR;
- Rotate RD 2 bits to the left, then store as RDR;
- MA in this round is the LD for the next round, thus store it in the next row of LD;
- RXOR in this round is RD for the next round, thus store it in the next row of RD;
- The XOR result between MA and RDR is assigned to RXOR;
- For Modular Addition constraint, we refer to the paper written by Helger Lipmaa et.al.[5], which takes input α (LDR) and β (RD), output γ (MA). The constraint is as follows:

$eq(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \cap$
$\big(myxor(\alpha, \beta, \gamma)\ XOR\ (\beta \ll 1)\big) = 0$;

- Store DP as 2^{-W_h} where W_h is the sum of its input;
- The sum of sbox_counter and bound is lesser or equal to the estimator.

The following are the constraints in Minizinc code, starting with the functions and then the constraints:

1. Functions:

- function var 0..1: myxor(var 0..1 :x, var 0..1 :y) = ((x+y)=1);
- function var int: XOR(var 0..1 :x, var 0..1 :y, var 0..1 :z) = (x+y+z) mod 2;
- function var bool: AND(var int:x, var int:y) = (x==y) /\ (x!=0) ;
- function var int:eq(var int:x,var int:y, var int:z) = if x==y /\ y==z /\ x== z then 1 else 0 endif ;
- function var int:neq(var int:x,var int:y, var int:z) = if x==y /\ y==z /\ x== z then 0 else 1 endif ;

2. Constraints:

- constraint sum([RD[0,j]|j in 0 ..15]) > 0 \/ sum([LD[0,j]|j in 0 ..15]) > 0;
- constraint forall(r in 0..n-1)(forall(i in 0..15)(LDR[r,(i+7) mod 16] = LD[r,i]));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(RDR[r,i] = RD[r,((i+2) mod 16)]));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(MA[r,i] = LD[r+1,i]));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(RD[r+1,i] = RXOR[r,i]));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(RXOR[r,i] = myxor(RDR[r,i],MA[r,i])));
- constraint forall(r in 0..n-1)(forall(i in 0..15)(AND((eq(AND(MA[r,(i+1) mod 16],mask[15-i]) , AND(LDR[r,(i+1) mod 16],mask[15-i]), AND(RD[r,(i+1) mod 16],mask[15-i]))) , (myxor(XOR(MA[r,i],LDR[r,i],RD[r,i]) , AND((RD[r,(i+1) mod 16]),mask[15-i])))) == 0));
- constraint DP = sum(r in 0..n-1,i in 0 .. 15)(AND(neq(MA[r,i],LDR[r,i],RD[r,i]), mask[i]));
- constraint forall(i in 0..n-2)(bound[n-i-2]+DP<=estimator);