INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2023, VOL. 69, NO. 2, PP. 261–267
Manuscript received April 25, 2023; revised April, 2023. DOI: 10.24425/ijet.2023.144359

Optimal SAT Solver Synthesis of Quantum Circuits Representing Cryptographic Nonlinear Functions

Adam Jagielski

***Abstract*—In this article we present a procedure that allows to synthesize optimal circuit representing any reversible function within reasonable size limits. The procedure allows to choose either the *NCT* or the *MCT* gate set and specify any number of ancillary qubits to be used in the circuit. We will explore efficacy of this procedure by synthesizing various sources of nonlinearity used in contemporary symmetric ciphers and draw conclusions about properties of those transformations in quantum setting. In particular we will try to synthesize optimal circuit representing ASCON cipher SBOX which recently won NIST competition for Lightweight Cryptography standard.**

***Keywords*—quantum computing; circuit synthesis; cryptography; satisfiability problem**

I. Introduction

CRYPTOGRAPHY is a fundamental aspect of modern communication and data security, and its importance has only grown in the age of digital information. With the rapid development of quantum computing, however, traditional cryptographic methods are at risk of becoming obsolete due to their vulnerability to attacks by quantum computers. As a result, there is a pressing need for the development of quantum-safe cryptographic techniques that can withstand attacks by both classical and quantum computers.

Circuit synthesis is essential in the security analysis of ciphers because it enables us to evaluate the security of encryption schemes against both classical and quantum attacks. By synthesizing optimal and sub-optimal circuits representing ciphers, we can identify potential vulnerabilities and weaknesses in the encryption scheme, and explore the effectiveness of different cryptographic transformations in terms of their ability to resist attacks. Additionally, circuit synthesis allows us to analyze the computational resources required to execute the circuit and, therefore, the difficulty of attacking the encryption scheme. This knowledge is essential in the development of advanced encryption techniques that are resistant to attacks and ensure the confidentiality and integrity of sensitive information. Overall, circuit synthesis plays a critical role in the security analysis of ciphers and is essential in the development of effective and secure cryptographic methods that can withstand attacks by both classical and quantum computers.

Author is with Military University of Technology in Warsaw, Poland, (e-mail: adam.jagielski@wat.edu.pl).

II. Synthesis Procedure

In this section we present a modification of procedure demonstrated in [1]. Its main principle of operation is reduction to a boolean satisfiability (SAT) problem, which represents a question of existence of a circuit for given reversible function using exact number of gates. This approach, where we aim to synthesize a circuit with set number of resources, is called *Exact Synthesis*. We will start by describing the reduction of exact synthesis problem to an equivalent SAT problem.

A. Describing a valid NCT and MCT circuit

The aim of this part is to generate high-level set of constraints that are enough to describe any valid MCT circuit.

As we recall a MCT circuit is a sequence of gates from *"Multiple Control Toffoli"* set which contains *NOT, CNOT, Toffoli* gates and Toffoli gate equivalents with higher number of control qubits. NCT circuit is more restricted version where only allowed gates are *NOT, CNOT* and *Toffoli*.

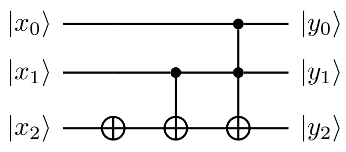

Fig. 1. NCT gate set

For any NCT and MCT circuit there are two main parameters that describe its global properties - circuit gate count (G) and circuit width (W). In this paper we consider depth of circuit to be equal to the gate count as we do not include gates parallelism into consideration and we consider circuit as a strict sequence of gates.

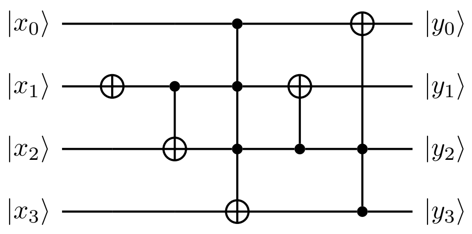

Fig. 2. Example MCT circuit

The first thing we need to address is how to efficiently and unambiguously encode the given circuit in such a way that the encoding can be easily used in the synthesis procedure. To do this, we will divide our circuit graph into a checkerboard consisting of horizontal lines, equivalent to lines of data qubits, and vertical layers. We can observe that each gate occupies one layer, and each layer is occupied by one gate. Therefore, from now on, we will use the terms "layers" and "gates" of a circuit interchangeably.

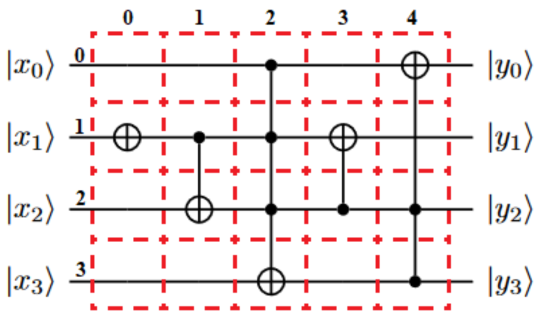

Fig. 3. Circuit split into checkerboard

Now let us consider the possible states of a single board field. Each field can have one of three states:

- Field is empty;
- Field contains a target qubit of a gate in given layer;
- Field contains a control qubit of a gate in given layer.

Hence, we will require two bits of data to describe each field. Let $0 \leq i < G, 0 \leq j < W$ be a pair of coordinates indicating the field, where i indicates an index of a layer and j indicates index of a line. We will use two boolean variables to encode state of each field: $c_{i,j}$ and $t_{i,j}$ which mark whether the field is control or target respectively. The circuit above can be encoded with $G * W * 2 = 4 * 5 * 2 = 40$ boolean variables:

$$\begin{gathered}
c_{0,0}=0,\ c_{1,0}=0,\ c_{2,0}=1,\ c_{3,0}=0,\ c_{4,0}=0\\
c_{0,1}=0,\ c_{1,1}=1,\ c_{2,1}=1,\ c_{3,1}=0,\ c_{4,1}=0\\
c_{0,2}=0,\ c_{1,2}=0,\ c_{2,2}=1,\ c_{3,2}=1,\ c_{4,2}=1\\
c_{0,3}=0,\ c_{1,3}=0,\ c_{2,3}=0,\ c_{3,3}=0,\ c_{4,3}=1\\
\\
t_{0,0}=0,\ t_{1,0}=0,\ t_{2,0}=0,\ t_{3,0}=0,\ t_{4,0}=1\\
t_{0,1}=1,\ t_{1,1}=0,\ t_{2,1}=0,\ t_{3,1}=1,\ t_{4,1}=0\\
t_{0,2}=0,\ t_{1,2}=1,\ t_{2,2}=0,\ t_{3,2}=0,\ t_{4,2}=0\\
t_{0,3}=0,\ t_{1,3}=0,\ t_{2,3}=1,\ t_{3,3}=0,\ t_{4,3}=0
\end{gathered} \tag{1}$$

Naturally, some constraints for those variables emerge:

1) A field cannot be both a control and a target;
2) There can be only one target in each layer;
3) The number of controls in each layer of MCT circuit is unrestricted as long as other constraints are fulfilled, the number of controls in NCT circuit in each layer cannot exceed two.

The constraints can be written formally as:

$$\forall_{0\leq i<G, 0\leq j<W} \left(\neg c_{i,j} \vee \neg t_{i,j}\right) \tag{2}$$

$$\forall_{0\leq i<G} \left(\sum_{0\leq j<W} t_{i,j} = 1 \right) \tag{3}$$

$$\forall_{0\leq i<G} \left(\sum_{0\leq j<W} c_{i,j} \leq 2 \right) \tag{4}$$

For NCT circuit all of the three hold, for MCT circuit the last one is omitted. The second and third constraints are cardinality constraints that specify that a given set of variables must have at least or at most a certain number of true values. These constraints are challenging to transform into standard CNF clauses used for solving SAT problems. Transformations presented in [2] yielded the best results for our setup.

Constraints listed above are enough to describe a valid NCT or MCT circuit implementing unspecified function. Solving problems with those only will produce some "random" proper circuit of given gate count and width.

B. *Assigning a functionality to the circuit*

We know how to encode a proper circuit using boolean variables and how to construct a set of constrains that allows to produce *some* circuit. In this section we will describe another set of constraints which will be necessary to ensure that the circuit implements intended reversible function. Let us start with investigating how the circuit interacts with chosen input.

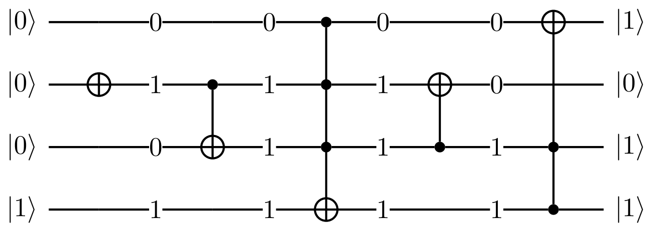

Fig. 4. MCT circuit with data transformations

The graph above represents how the input is transformed into output of the circuit, one gate after another. From the graph and from the definitions of MCT gates, we can deduce that two conditions must be met for one bit to be changed on its line after a gate execution:

1) The field in given line and layer must contain a target;
2) For all the control fields on another lines in given layer, the bits in those fields must be set to one.

Formally the bit changes its value moving from $b_{i,j}$ to $b_{i+1,j}$ when those are met:

1) $t_{i,j} = 1 \wedge c_{i,j} = 0$;
2) $\bigwedge_{j'=0, j'\neq j}^{w-1} (c_{i,j'} \implies b_{i,j'})$.

Unfortunately, the second condition, which is quite complex, has to be computed separately for each index j. We can notice though, that the first condition enforces the state of $t_{i,j} = 1$ and thus $c_{i,j} = 0$. Because now for $j' = j$ the implication predecessor must be negative, the implication is positive and it does not bear any effect on the value of the for-all quantifier. The condition now can be written as:

$$\left(\bigwedge_{j'=0}^{w-1} (c_{i,j'} \implies b_{i,j'})\right) = C_i \tag{5}$$

The condition is now completely agnostic in respect of the second (j) coordinate an therefore can be computed once per layer instead of once per field. This can dramatically save number of variables and clauses in final formula. The state $b_{i+1,j}$ of bit, after processing with ith gate can be written as:

$$b_{i+1,j} = b_{i,j} \oplus (t_{i,j} \wedge C_i) : C_i = \bigwedge_{j=0}^{w-1} (c_{i,j} \implies b_{i,j}) \tag{6}$$

This is enough to encode transformation of single input bit-vector into single output, yet it is not enough to encode whole permutation of the reversible function, hence we introduce additional upper index for bit states:

$$\begin{aligned} b^k_{i,j} :& i \in \{0, \dots, G-1\}, \\ & j \in \{0, \dots, W-1\}, \\ & k \in \{0, \dots, 2^W - 1\} \end{aligned} \tag{7}$$

The new index specifies from which input sequence it was derived. The full variable $b^k_{i,j}$ can be read as: "The state of jth bit coming from input vector k after being processed by i gates." k is a natural number decoded from binary input vector.

Notice that the t and c variables do not have the new index. That is because they describe the circuit itself which is common for all of the possible inputs.

The encoding of bits transformation must be repeated for each $k \in \{0, \dots, 2^W - 1\}$. That produces additional variables in number of $(2^W - 1) * W * (G + 1)$.

At last we have to introduce edge constraints for inputs and output. This is done simply by assigning values for all $b^k_{0,j}$ variables according to the input values they represent and for all $b^k_{G,j}$ variables according to the desired output values of the reversible function we want to synthesize. Those variables are the only ones with set values beforehand solving the problem.

The final CNF formula with described constraints contains $\mathcal{O}(G * W * 2^W)$ variables. The exact number of variables and clauses differs depending on the encoding of cardinal constraints and whether intermediate variables have been used in transformations of those constraints into CNF form.

The solution of the problem produces values of $c_{i,j}$ and $t_{i,j}$ variables which can be decoded into a circuit which implements given reversible function.

What is worth noticing, CNF formulas for exact synthesis of two different functions of the same size using the same gate number are virtually identical. The only difference are edge assignments on the output variables $b^k_{G,j}$.

C. Don't cares and ancillas

Up to this time we only considered fully described reversible function, i.e. functions where full permutation of states is known and desired to be implemented by the circuit. The synthesis procedure allows to reduce edge constraints in various ways. The first case of incomplete permutation can be described as follows. Let permutation $P = [3, 1, 2, 0]$. We already know how to implement a set of constrains to synthesize this permutation. Now lets consider incomplete permutation $P' = [3, 1, X, X]$. The elements $P'(2) = X$ and $P'(3) = X$ are so called don't care states. This means that when synthesizing a circuit for the permutation we do not care about the outputs of the circuit $P'(2), P'(3)$. We only require that for $P'(0) = 3$ and $P'(1) = 1$. Both circuits implementing permutations P and $P" = [3, 1, 0, 2]$ would be valid in this situation.

The second example is implementation of incomplete outputs with don't care bits. This allows us to implement efficiently surjective functions embedded in reversible functions. In this example we want to implement a $\mathcal{B}^3 \to \mathcal{B}^2$ function with following truth table.

TABLE I
EMBEDDED FUNCTION TRUTH TABLE

inputs			outputs		
x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	x	0	1
0	0	1	x	0	0
0	1	0	x	1	0
0	1	1	x	1	0
1	0	0	x	0	1
1	0	1	x	1	1
1	1	0	x	0	0
1	1	1	x	1	1

The function has been embedded into a $\mathcal{B}^3 \to \mathcal{B}^3$ with a single don't care output variable y_2. Solving problem with those reduced constraints on variable y_2 will yield a circuit that implements some 3×3 function which truncated to 3×2 will be equivalent to chosen function.

At last the most important usage of don't care variables is introduction of ancillary variables into a function. In this case we will want to synthesize a circuit that implements some 2×2 function which is allowed to use an additional ancilla bit. The truth table for the function is defined as:

TABLE II
2×2 FUNCTION TRUTH TABLE

inputs		outputs	
x_1	x_0	y_1	y_0
0	0	0	1
0	1	1	0
1	0	0	0
1	1	1	1

Adding another unused bit of input and output to this truth table produces truth table III.

In this setting we did not reduce restrictions on the circuit because the variable x_2 directly maps to y_2 and every circuit implementing original function implements the new extended

TABLE III
EMBEDDED FUNCTION TRUTH TABLE

inputs			outputs		
x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	1	1	1

function as well and vice versa. The solutions for this problem are exactly the same as solutions for the 2×2 function. To extend the number of possible solutions we remove restrictions for states where ancilla qubits are not all set to zeroes. This is based on a assumption that ancillas are always in ”ready” all-zeroes state before and after being used.

TABLE IV
FUNCTION TRUTH TABLE WITH ANCILLARY QUBIT

inputs			outputs		
x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	x	x	x
1	0	1	x	x	x
1	1	0	x	x	x
1	1	1	x	x	x

Now this problem has more solutions than the original one as we reduced its restrictiveness. In fact given some set number of gates adding an ancillary line might render original unsolvable problem solvable which can be used for achieving smaller circuits using ancillas.

D. Optimal synthesis

In general to find an optimal circuit and prove its optimality is to find a circuit of some cost and prove there is no equivalent circuit with lesser cost. Fortunately for us, the exact synthesis procedure not only allows us to find circuits of given gates number but also in case of failure we know that a circuit of this size does not exist due to unsatisfiability of the problem. We can use this property in two directions:

1) *Forward* - we start at a gate count equal to zero and try to solve the problem. If there is no solution we increment the number of gates. We repeat incrementing and solving until a circuit is found. When we find a circuit we know it is optimal because there was no solution for all of the smaller gate counts.
2) *Backward* - First we have to notice, that if there exists a circuit of size k that implements a function, there always exist equivalent circuits of size $k+2, k+4, k+6 \ldots$. That is because we can add two NOT gates on one line at the end of the circuit which will cancel each other therefore resulting in equivalent circuit. This implication can be inverted. If there is no circuit of size k there is no circuits of size $k-2, k-4, k-6, \ldots$. Hence if we have a circuit of circuit k and we prove that there is no circuit of size $k-1$ and $k-2$ we absolutely know that the circuit of size k is optimal. We start synthesis with some upper bound for the circuit size. One of them for MCT circuits size is $G \leq (W-1) * 2^W + 1$ from [3]. We know that there is some circuit that implements required function not bigger that the chosen bound. We start at k equal to the bound. If we find a circuit of size k, we decrement that value, we stop when we find such a size k that there it exists a circuit of this size but there exists no circuit of size $k-1$ and $k-2$.

This concludes how exact synthesis of circuit can be used for optimal synthesis.

III. CIRCUITS TO FIND

In the experimental part of this paper we searched for optimal implementations of small SBOXes and other nonlinear operations. The transformations have been taken from finalists of following competitions:

1) NIST Lightweight Cryptography Call;
2) NIST Hash Function Call;
3) NESSIE;

Additionally we investigated MiniAES SBOX.

SBOXes and equivalents up to size 5×5 where researched:

- 3×3 functions
  - Xoodyak Chi [4];
- 4×4 functions
  - Elephant SBOX [5];
  - GIFT-COFB SubCells [6];
  - Photon-Beetle SBOX [7];
  - MiniAES SBOX [8];
  - 2 Whirlpool SBOXes [9];
  - 2 JH SBOXes [10];
- 5×5 functions
  - ASCON / ISAP-A SBOX [11];
  - KECCAK / ISAP-K Chi [12].

We investigated six different settings for each function:

- NCT circuit optimal synthesis
  - No ancillary qubits (NoAnc);
  - Single ancilla (SAnc);
  - Double ancilla (DAnc);
- MCT circuit optimal synthesis
  - No ancillary qubits;
  - Single ancilla;
  - Double ancilla;

Where single ancilla means using at most one ancillary qubits, double ancilla means using at most n ancillary bits for a function of size $n \times n$.

A. Relationships between settings

Naturally a hierarchy between problems emerges, for example, every NCT-NoAnc circuit is a proper MCT-NoAnc circuit, therefore MCT-NoAnc optimal solution cannot be bigger then one for NCT-NoAnc. Similarly MCT-NoAnc circuit is a proper MCT-SAnc and MCT-DAnc circuit.

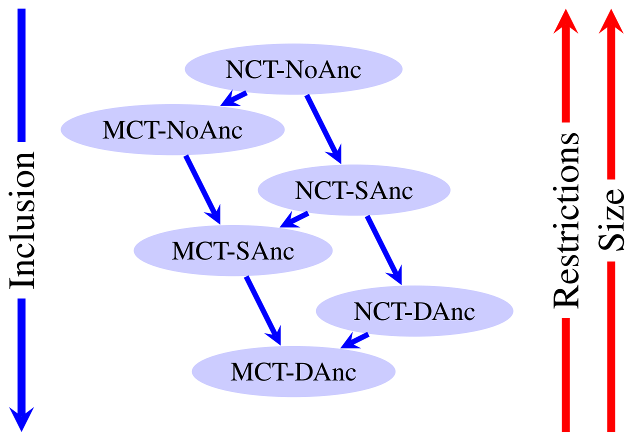

Fig. 5. Problems hierarchy

The graph arrows indicate inclusion order of solutions sets. Circuits from settings in nodes with outgoing arrows are valid in settings where the arrow finishes. The graph is transient and arrows marking indirect inclusions have been omitted. This graph can be used for finding bounds for sizes for optimal solutions in different settings. Node with NCT-NoAnc is minimal node with respect to inclusion and therefore maximal node in terms of optimal solutions sizes, node with MCT-DAnc setting is the opposite. For some reversible function let t be a size of optimal circuit in NCT-NoAnc and s be a size of optimal circuit in MCT-DAnc. Sizes of optimal circuits in all of six settings fit in range $[s, t]$. This hierarchy might be used for optimization of synthesis across settings. In some cases where $s = t$ it is enough to solve for two edge cases to infer all other sizes.

IV. Experimental results

In the final section we present results of synthesis for chosen circuits and settings. The table below presents sizes of circuits found using the procedure. All of the problems were solved using *Parkissat* SAT-solver which is the winner of Parallel track 2022 SAT-Solver Competition and which was found to be the most efficient in our setting. The solver was working on 128 cores machine. For every problem we started with searching for solutions in edge cases (NCT-NoAnC and MCT-DAnc settings) and we skipped other setting if edge cases had the same size of solution.

- **Xoodyak** - The only 3×3 function. In this case NCT and MCT cases are equivalent. This circuit was synthesized in NCT-NoAnc and NCT-DAnc settings. Both settings resulted in the same circuit and therefore there was no gain associated to adding ancillaries. NCT-SAnc setting has the same solution as two edge cases for this function;

TABLE V
Synthesis results

SBOX	Setting	GC	NOT	CNOT	Toff.	3Toff.	4Toff.
			3×3 Functions				
Xoodyak	N-NA	6	0	3	3	-	-
	N-DA	6	0	3	3	-	-
			4×4 Functions				
Elephant	N-NA	10	1	4	5	-	-
	M-DA	10	1	4	5	0	-
GIFT-COFB	N-NA	9	2	3	4	-	-
	M-DA	9	2	3	4	0	-
Photon-Beetle	N-NA	11	1	5	5	-	-
	M-DA	11	1	5	5	0	-
Mini-AES	N-NA	13	2	6	5	-	-
	N-SA	13	2	6	5	-	-
	N-DA	13	2	6	5	-	-
	M-NA	12	2	6	2	2	-
	M-SA	12	2	6	2	2	-
	M-DA	12	2	6	2	2	-
Whirlpool e	N-NA	13	1	4	8	-	-
	N-SA	13	1	4	8	-	-
	N-DA	13	1	4	8	-	-
	M-NA	12	1	4	5	2	-
	M-SA	12	1	4	5	2	-
	M-DA	12	1	4	5	2	-
Whirlpool r	N-NA	13	3	5	5	-	-
	M-DA	13	3	5	5	0	-
JH 0	N-NA	10	1	2	7	-	-
	M-DA	10	1	2	7	0	-
JH 1	N-NA	12	2	4	6	-	-
	M-DA	12	2	4	6	0	-
			5×5 Functions				
ASCON	N-NA	17	1	6	9	-	-
	N-SA	17	1	6	9	-	-
	N-DA	17	1	6	9	-	-
	M-NA	16	1	6	7	2	0
	M-SA	16	1	6	7	2	0
	M-DA	16	1	6	7	2	0
Keccak	N-NA	13	0	5	8	-	-
	N-SA	13	0	5	8	-	-
	N-DA	13	0	5	8	-	-
	M-NA	12	3	3	5	1	0
	M-SA	12	3	3	5	1	0
	M-DA	12	3	3	5	1	0

- **Elephant, GIFT-COFB, Photon-Beetle, Whirlpool-r, JH-0, JH-1** - For those functions both edge cases resulted in a circuit of the same size. Therefore the solution in all settings is the same as the solution found for NCT-NoAnc which has additional benefits of minimal gate library and minimal number of lines even if the settings allows not to have those restrictions;
- **Mini-AES, Whirlpool-e, Keccak** - Synthesis of those functions resulted in solutions of different sizes for the edge cases, due to this fact the remaining settings where solved for as well. For all of the circuits there was no gain achieved due to adding ancillas, the reduction of size comes solely from extension of the gate set. Hence, for NCT settings the solution is the result achieved for NCT-NoAnc and for MCT settings the one from MCT-NoAnc with additional benefit of not using ancillas.
- **ASCON** - This function proved to be to complex to be solved optimally. The synthesis procedure times out (24h limit) while trying to find solution at 13 gates. In this case we used reverse order of synthesis. We started

looking for solutions at 32 gates and after finding a circuit we decreased number of gates until times out where achieved. We strongly believe that even tough we could not disprove existence of smaller circuits in particular settings the presented ones still might be the optimal due to solving times patterns which is explained later in conclusions. While searching for solutions we did not achieved any improvements on the size of circuits due to adding ancillas.

TABLE VI
SYNTHESIZED CIRCUITS

SBOX	Settings	Circuit
		3×3 Functions
Xoodyak	All 6	(0,2); (0,1,2); (1,0); (1,0,2); (2,1); (2,0,1)
		4×4 Functions
Elephant	All 6	(0,1,2); (2,0,3); (0,1); (1); (0,3); (2,1); (1,2,3); (3,1,2); (1,0); (3,0,1);
GIFT-COFB	All 6	(1,0,2); (0,1,3); (2,1); (3); (1); (3,0,1); (3,2); (1,3); (2,0,3);
Photon-Beetle	All 6	(3,1,2); (2,3); (1,2,3); (3,1,2); (0,2); (1,0,3); (2); (3,0); (3,2); (3,1); (2,1,3);
Mini-AES	NCT	(0,1); (1,0); (1); (3,1); (2,0,3); (0,3); (1,2); (0,1,2); (1,0,3); (3,1,2); (1,0,3); (2,0); (0);
	MCT	(0,1); (1,0); (1); (3,1); (2,0,3); (1,2); (3,1,2); (0,3); (1,0,2,3); (3,0,1,2); (2,0); (0);
Whirlpool e	NCT	(2,0,1); (1,0,3); (0,1,2); (3,0); (0); (1,2,3); (2,0,3); (0,1,2); (1,0,2); (3,1); (1,3); (3,0,1); (0,1);
	MCT	(3,1,2); (0,1,2,3); (2,3); (1,2,3); (2,0,1); (1,0); (1,0,2,3); (0); (3,1); (1,0,3); (3,2); (0,1,3);
Whirlpool r	All 6	(3); (2,0); (0,2,3); (2); (2,0,1); (1,3); (3,1,2); (1,0,3); (0,3); (0,1); (1); (2,3); (1,0,2);
JH 0	All 6	(0,2,3); (0,1); (3,0,1); (2,1,3); (1,0,2); (0,1,3); (3,0,2); (1,0,3); (0); (3,0);
JH 1	All 6	(0,2,3); (2,1); (3,0,1); (0,2); (2,1,3); (3,0); (1,0,3); (3,0,2); (0,1,2); (2,3); (1); (0);
		5×5 Functions
ASCON	NCT	(4,2,3); (2,0); (4,2); (2,0,1); (0,3,4); (2,3); (0,1,2); (1,0,2); (2); (3,1,2); (0,3); (4,0,2); (1,3,4); (4,0,2); (3,4); (4,0); (1,3,4);
	MCT	(4,2,3); (2,0); (4,2); (2,3); (2,0,1); (0,1); (3,2,4); (1,0,2); (0,3); (1,0,2,3); (2); (1,2,4); (0,3,4); (3,1,2); (1,0,2,4); (4,0);
Keccak	NCT	(1,0,3); (1,2,3); (3,0,4); (3,0); (0,2); (1,0,3); (0,1,2); (2,4); (2,3,4); (4,1); (4,0,1); (1,2,3); (1,3);
	MCT	(4); (3,0,4); (0,1,2); (0,2); (2,4); (1,0,2,4); (4); (2,3,4); (4,0,1); (4,1); (1,2,3); (2);

Circuits in table are interpreted as a sequence of gates, each gate is represented as a tuple. In each tuple first number indicates the index of target qubit. All other indices point to control qubits. As an example Xoodyak function circuit is shown below:

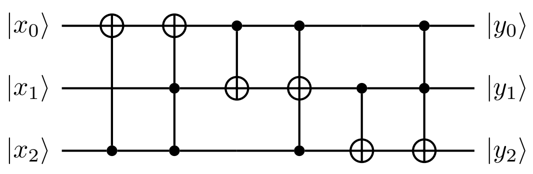

Fig. 6. Xoodyak optimal circuit

V. Conclusions

Presented procedure enables optimal synthesis of arbitrary functions within certain scope of sizes. All of the 3×3 and 4×4 functions were solved and one of size 5×5 as well. The other 5×5 function was not proved to be solved optimally. Given computational setup described earlier each task was dealt with within 24 hours and most of them within an hour. The only outlier was ASCON SBOX which timed out.

While solving subsequent tasks of exact synthesis we noticed a characteristic time pattern. Let k_0 be the size of optimal circuit, i.e for size k_0 the SAT problem instance is the first one to be satisfiable. For all sizes $k \in [0, 1, 2, 3, \ldots, k_0 - 1]$ SAT problems are unsatisfiable and the time to solve them grows exponentially as k grows. Then the time needed for $k = k_0$ drops drastically by order of magnitude up to thousands. Solving for $k > k_0$ usually takes about the same time as for $k = k_0$. This means that if using reverse order of synthesis we find some circuit of size k and than solving for $k - 1$ takes disproportionately long time, we may suspect that k is optimal size. Tough this argument is not enough to prove optimality of certain circuits it may provide some intuition about possibility of existence of smaller solutions. This pattern is observed for ASCON circuits which leads to our belief that the presented circuits may be optimal.

Further, we can observe that for each function adding ancillaries does not benefit with shortening of the circuit. For seven out of eleven circuits extending gate set from NCT to MCT did not yield smaller circuits. For remaining four functions extending gate set from NCT to MCT results with reduction of the circuit by at most one gate.

Those results may raise certain suspicion about correctness of implementing ancillaries in synthesis procedure. We verified it by synthesizing small circuits which are known to benefit from added ancillas and in fact the procedure correctly finds optimal solutions using added lines. For example decomposition of 3-Toffoli gate into NCT gates is one of those:

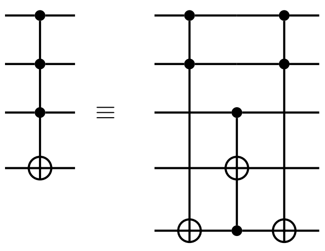

Fig. 7. Example MCT circuit

We expect that our results may be useful in further workings on vast subject of quantum security of encryption schemes. Taking into consideration last events on NIST Lightweight Cryptography Call and winning of ASCON in this competition we find that those results might be vital for assessing security of the new standard in quantum regime.

References

[1] R. Wille and R. Drechsler, *Towards a Design Flow for Reversible Logic*. Dordrecht: Springer Netherlands, 2010. [Online]. Available: https://doi.org/10.1007/978-90-481-9579-4

[2] C. Sinz, "Towards an Optimal CNF Encoding of Boolean Cardinality Constraints," in *Principles and Practice of Constraint Programming - CP 2005*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and P. van Beek, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3709, pp. 827–831. [Online]. Available: https://doi.org/10.1007/11564751_73

[3] D. Miller, D. Maslov, and G. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Proceedings - Design Automation Conference*, Jul. 2003, pp. 318–323. [Online]. Available: https://doi.org/10.1109/DAC.2003.1219016

[4] J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer, "Xoodyak, a lightweight cryptographic scheme," *IACR Transactions on Symmetric Cryptology*, pp. 60–87, Jun. 2020. [Online]. Available: https://doi.org/10.46586/tosc.v2020.iS1.60-87

[5] T. Beyne, C. Yu Long, C. Dobraunig, and b. Mennink, "Elephant v2." [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/elephant-spec-final.pdf

[6] S. Banik, A. Chakraborti, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, S. Meng Sim, and Y. Todo, "Gift-cofb v1.1." [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/gift-cofb-spec-final.pdf

[7] Z. Bao, A. Chakraborti, N. Datta, J. Guo, M. Nandi, T. Peyrin, and K. Yasuda, "Photon-beetle authenticated encryption and hash family." [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/photon-beetle-spec-final.pdf

[8] R. C.-W. Phan, "Mini advanced encryption standard (mini-aes): A testbed for cryptanalysis students," *Cryptologia*, vol. 26, no. 4, pp. 283–306, Oct. 2002. [Online]. Available: https://doi.org/10.1080/0161-110291890948

[9] V. Rijmen and P. S. L. M. Barreto, "The whirlpool hashing function," 2003.

[10] W. Hongjun, "The hash function jh," 2011. [Online]. Available: https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf

[11] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2 submission to nist." [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf

[12] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak specifications." [Online]. Available: http://keccak.noekeon.org/