LIGHTER-R: Optimized Reversible Circuit Implementation For SBoxes

Vishnu Asutosh Dasu*, Anubhab Baksi†, Sumanta Sarkar‡ and Anupam Chattopadhyay§
*Manipal Institute of Technology, Manipal, India
†§School of Computer Science & Engineering, Nanyang Technological University, Singapore
‡TCS Innovation Labs, Hyderabad, India
*vishnu.asutosh@learner.manipal.edu, †anubhab001@e.ntu.edu.sg,
‡sumanta.sarkar1@tcs.com, §anupam@ntu.edu.sg

2019 32nd IEEE International System-on-Chip Conference (SOCC) 978-1-7281-3483-3/20/$31.00 ©2020 IEEE 10.1109/SOCC46988.2019.1570548320

***Abstract*—This article presents the 'LIGHTER-R' tool that aims at implementing a given 4-bit SBox using reversible logic libraries. An SBox is a basic building block in the symmetric key cryptography. In order to analyze the security of the symmetric key algorithms in the futuristic reversible computing paradigm, one needs to implement those algorithms in the reversible manner. Our tool offers an end-to-end flow for implementing a given 4-bit SBox (which is the common choice in recent designs) in reversible circuits while optimizing the cost for a given cost metric. It extends the tool LIGHTER, which is developed for classical computing. LIGHTER-R enjoys the advantages of LIGHTER; such as, easy to use and free.**

***Index Terms*— symmetric key, sbox, implementation, optimization, reversible computing**

I. Introduction

CMOS technology is predominantly used in today's hardware. However, advancements in CMOS technology have reached a saturation point. As the Moore's law conjectures in '75, the number of transistors in an integrated circuit is nearly doubled every two years. Naturally, the dimensions of the transistors are being shrunk rapidly, at nearly an exponential scale. The present day feature width is typically 7 nm; in comparison, the diameter of a helium atom is about 0.1 nm. Even if we ignore the increasing difficulties related to manufacturing (lithography/fabrication), performance (power consumption, heat dissipation, quantum effects), and economy; this technology is about to meet a dead end quite soon as the transistors can not be made infinitesimal (unless a major breakthrough in physics is achieved meanwhile).

To overcome this bottleneck, several new approaches relating to data processing and also non-von Neumann architectures are being analyzed. Some of the approaches try to replace the CMOS technology by a direct alternative (short-term alternative), such as the *memristor* [1]. Other approaches deal with finding a whole new paradigm of computing (long-term alternative). One such long-term goal is the so-called *reversible logic*, or *reversible computing*, or *conservative logic*.

Reversible computing allows computation to be performed in a *reversible* manner; i.e., computations are bijective onto itself and they do not increase entropy (see Section II for more information). So, the input can be uniquely identified given the output in the reversible paradigm. The most simple reversible logic is the (1-bit) NOT gate, where the input is the inverse of the output. In contrast, the (2-input) XOR gate is not a reversible logic; since it is not possible to compute both the inputs of it given its output, in general. In order to get a reversible XOR gate, the minimum requirement is to insert one extra output line; say, this output line directly gives the first input line. In this case, the logic becomes reversible – in fact, this gate is known as the CNOT gate (see Section II-A for more details). Any such extra output line inserted to the circuit is termed as a *garbage* line. Also, there can be situations, where extra input line has to be inserted to make the circuit reversible — any such input lines are known as *ancilla* lines. Such lines are typically initialized with logic 0 or 1. It is also assumed that a reversible circuit does not have any fan-out (feed-forward), i.e., output lines do not drive multiple circuits; as well as any feedback.

On the other hand, symmetric key cryptographic algorithms are used quite extensively when it comes to data protection/privacy and related matters. One may be interested in analyzing the security of those algorithms (also termed as, *ciphers*) when those are running on a reversible computer. When a symmetric key cipher will be running on a reversible computer, new attack methodologies can be applied, such as the *Grover's search* [2], which are not possible on the existing (classical) computer. It therefore makes sense to (optimally) implement symmetric key algorithms in the reversible paradigm. One may refer to [3]; where the authors implement AES[1], the most frequently referred symmetric key cipher in the reversible paradigm. Very recently, in [4], the authors revise through previous works on AES; and also present a largely improved reversible implementation of the AES *substitution box* (*SBox* for short — see Section III-A for more details).

Similar to [4], our work also deals with the SBoxes, in particular the 4×4 SBoxes. To begin with, we take the free tool LIGHTER [5]; which aims at optimally implementing 4-bit SBoxes for a given CMOS logic library and the corresponding cost metric (it also works with CPU instructions where each operation is of unit cost). We take this tool as the back-end such that it can work with the common reversible logic libraries, and turn it to a full fledged tool named "LIGHTER-R" (the last 'R' stands for reversible). Similar to LIGHTER, our tool

[1]See https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

allows an easy-to-use command line interface and is free & open-source[2].

The rest of the document is as organized as follows. In Section II, we present the basics of reversible computing. Next, in Section III, we give a brief overview of LIGHTER. Then, in Section IV, we describe how we use LIGHTER as the back-end for reversible implementation of 4×4 SBoxes. Finally, we conclude in Section V.

II. Reversible Computing Basics

It is known that every irreversible computing loses energy. As mentioned in [6], for every bit of information lost, $kT \ln 2$ Joule of energy is dissipated; where k is the Boltzmann constant and T is the temperature in Kelvin scale. It has been verified experimentally in [7]. It is proposed to carry out computations in reversible manner, so as to achieve the theoretical zero energy dissipation in [8]. This idea helps to develop the area of reversible logic synthesis, where computations are done in a reversible manner in both physical and logical abstraction. Several key futuristic technologies, including *quantum computing* and *superconducting digital circuits* [9], have been demonstrated via simulation or experiment that reversible circuits indeed reduce power consumption asymptotically. For more information in this regard, one may refer to [10, Section 2].

A. Logic Gates

One natural way to achieve reversible circuits is to use reversible gates, where fundamental operations are carried out in a reversible manner. Multiple gates have been proposed; among which, we present the relevant gates only. The standard circuit diagrams are presented in Figure 1. The circuit diagrams of the generalized Toffoli and Fredkin gates can be constructed by adding more control lines to the Toffoli gate (Figure 1c) and the Fredkin gate (Figure 1e), respectively.

- *NOT* gate: NOT$(a) = \overline{a}$.
- *CNOT (Controlled NOT)/ Feynman* : CNOT(a, b) = $(a, a \oplus b)$. This gate flips one output line if and only if the the other input line is at logic 1.
- *CCNOT/ Toffoli* gate: CCNOT(a, b, c)=$(a, b, ab \oplus c)$. This gate can be generalized with Tof$_n$ gate, where first $n-1$ variables are used as control lines; i.e., Tof$_n(a_0, a_1, \ldots, a_{n-2}, a_{n-1}) = (a_0, a_1, \ldots, a_{n-2}, a_0a_1 \cdots a_{n-2} \oplus a_{n-1})$. The NOT and CNOT gates are denoted as Tof$_1$ and Tof$_2$ respectively.
- *Swap gate*: Swap$(a, b) = (b, a)$. This gate simply swaps the input lines.
- *Controlled swap* (CSwap)/ *Fredkin* gate: Fred(a, b, c) = $(a, \overline{a}b \vee ac, \overline{a}c \vee ab)$. Thus, this gate swaps the target inputs if the control input is at logic 1. This is generalized with Fred$_n$ gate $(n > 1)$, where $n-2$ variables are used as control lines; and the other two lines are swapped if and only if all the control lines are at logic 1; i.e., Fred$_n$ $(a_0, a_1, \ldots, a_{n-2}, a_{n-1}) = (a_0, a_1, \ldots, a_{n-3}, \overline{a_0a_1 \cdots a_{n-3}}\ a_{n-2} \vee a_0a_1 \cdots a_{n-3}a_{n-1}, \overline{a_0a_1 \cdots a_{n-3}}a_{n-1} \vee a_0a_1 \cdots a_{n-3}a_{n-2})$. The Swap gate can be considered as a Fredkin$_2$ gate.
- *Peres gate*: Per(a, b, c) = $(a, a \oplus b, ab \oplus c)$. This gate can be generalized with Per$_n$ gate $(n > 2)$, where first $n-2$ variables are used as control lines; i.e., Per$(a_0, a_1, \ldots, a_{n-3}, a_{n-2}a_{n-1}) = (a_0, a_1, \ldots, a_{n-3}, a_0a_1 \cdots a_{n-3} \oplus a_{n-2}, a_0a_1 \cdots a_{n-3}a_{n-2} \oplus a_{n-1})$.

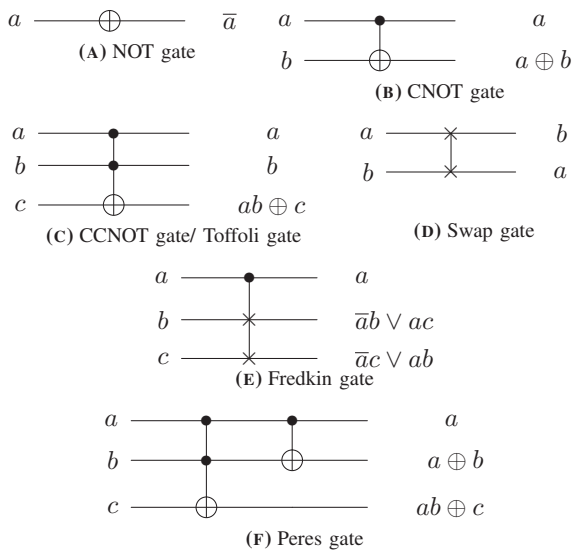

Fig. 1: Circuit diagrams of a few reversible gates

Since the outputs are permutation of the inputs, these gates can be equivalently represented by permutation matrices, e.g., the CNOT and Swap gates can be represented by the following permutation matrices:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \ \text{Swap} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

As an example, consider the realization of a half adder as given in Figure 2. Here, one input is always taken as logic 0 (ancilla); and also one output line is not used (garbage). Note that, if the ancilla line is not set to constant logic 0 (i.e., it is a variable), then this circuit reduces to the Peres gate; or a Tof$_2$ and a Tof$_3$ gate.

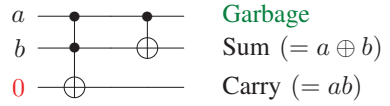

Fig. 2: Reversible circuit for half adder

As a side note, it can be observed that a Swap gate can be realized by using three CNOT gates in succession; see Figure 3. This becomes useful if the Swap gate is not available in the logic library.

[2]The LIGHTER-R tool can be found at https://github.com/vdasu/lighter-r.

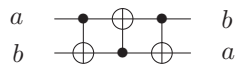

FIG. 3: Swap gate realized using CNOT gates

B. Logic Libraries

Given a particular technology, a set of (functionally complete) logic gates forms a logic library. The common reversible libraries[3], that are implemented with LIGHTER-R, are given here.

- *NCT*: NOT, CNOT, Toffoli gates.
- *MCT*: Multiple control Toffoli gates.

It may be noted that, creating a new logic library is relatively straightforward with LIGHTER-R, if needed. For that purpose, one has to create a library description file (preferably in .conf format) that contains the gates belonging to the library together with the corresponding costs.

C. Cost Metrics

In LIGHTER, the cost metric for hardware is taken as the gate equivalent for ASIC, as well as the number of instructions for CPU[4]. Broadly speaking, the cost for each of the gates is uniform (i.e., 1) for CPU (so, the cost is the total number of instructions); whereas the cost is non-uniform for ASIC, as the cost reflects the corresponding gate equivalent (so, the cost is the weighted total of the number of gates).

In case of reversible computing, the ASIC analogy can be given by the *quantum cost*. Another metric could be the *two-qubit cost* – we do not include the *single-qubit cost*, following [11, Section 6.1]. Detailed discussion of the metrics are out of scope for this work, one may refer to [12, Section 2.3] for this purpose. Analogous to the CPU cost, we also define the usual gate count metric. We present the cost metrics in Table I for the relevant gates. These metrics are adopted from the RCVIWER+ tool (version 2.5)[5].

TABLE I: Cost metrics for the relevant reversible gates

Gate	Quantum Cost	Two-qubit Cost	Gate Count
NOT (Tof_1)	1	0	1
CNOT (Tof_2)	1	1	1
CCNOT (Tof_3)	5	5	1
Tof_4	13	5	1

III. AN INTRODUCTION TO LIGHTER

A. Context of an SBox

As already mentioned, an SBox is a fundamental component in a lot of symmetric key ciphers. On several designs, it is the only component that inserts non-linearity. Technically, an SBox is a look-up table. It maps an m-bit input to an n-bit output. In other words, it is a vectorial Boolean function that maps $\text{GF}(2^m) \rightarrow \text{GF}(2^n)$. Such an SBox is described as an $m \times n$ SBox. Bijective SBoxes, which are a permutation of n-bits, are more commonly used nowadays and are often described as an n-bit SBox.

Ciphers like AES use an 8-bit SBox (i.e., an 8-bit look-up table). Recently, the community is more focused on lightweight designs[6]. Since an 8-bit SBox is generally considered costly, 4-bit SBoxes are predominantly used in recent designs; PRESENT [13], PRINCE [14], PRIDE [15] and SKINNY [16] to name a few. For example, $S =$ BF32AC916780E5D4 is the SBox used in PRINCE. What is implied here: The input 0 is mapped to b, the input 1 is mapped to output f, ..., the input f is mapped to 4. The inverse SBox S^{-1} can be derived from the definition of S.

There are $2^4! \approx 2^{44.25}$ 4-bit SBoxes. The choice of an SBox, however, is far from random — designers are careful to choose an SBox keeping certain specific objectives in mind. Multiple design criteria are used, so that it has necessary properties that can adequately block several attacks (refer to [17, Section 3.1]); at the same time, the implementation cost is within a reasonable bound. Finding a suitable SBox is still an active area of research today.

Although the traditional concept of an SBox is a look-up table; it is often considered as a vectorial Boolean function. This allows to implement an SBox as a combinatorial circuit with no requirement for a memory element. Also, it makes certain mathematical analysis on an SBox easier. For example, denoting the input bits by x_3 (MSB), x_2, x_1, x_0 (LSB) and the output bits by y_3 (MSB), y_2, y_1, y_0 (LSB); the PRINCE SBox can be described by the vectorial Boolean function in the *algebraic normal form*, as given next (these functions are also called the *coordinate functions*):

$$
\begin{aligned}
y_0 &= 1 \oplus x_0x_1 \oplus x_2 \oplus x_1x_2 \oplus x_0x_1x_2 \oplus x_3 \oplus x_0x_3 \oplus x_2x_3,\\
y_1 &= 1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_0x_1x_2 \oplus x_1x_3 \oplus x_1x_2x_3,\\
y_2 &= x_0 \oplus x_0x_1 \oplus x_3 \oplus x_0x_3 \oplus x_1x_3 \oplus x_0x_1x_3 \oplus x_1x_2x_3,\\
y_3 &= 1 \oplus x_1 \oplus x_1x_2 \oplus x_0x_1x_2 \oplus x_3 \oplus x_0x_1x_3 \oplus x_2x_3 \oplus\\
&\quad x_0x_2x_3.
\end{aligned}
$$

B. How LIGHTER Works

A short overview of the LIGHTER tool is given here for the sake of completeness. For more details, one may refer to [5]. In summary, LIGHTER take a 4×4 SBox which is a permutation (i.e., there is a bijection from its 4-bit input to the 4-bit output), a logic library, a cost metric for the gates in the library; and returns a (near) optimal implementation of the SBox in terms of the gates from the library in a C-like syntax.

The idea in LIGHTER is to use a graph based meet-in-the-middle (MITM) search algorithm to determine the (near) optimal implementation of a given 4×4 SBox S. This is done with respect a particular logic library, $\mathcal{B}$, which contains the cost metric. Given an SBox S: $\text{GF}(2^4) \rightarrow \text{GF}(2^4)$, the algorithm outputs the implementation of S whose instructions belong to $\mathcal{B}$.

[3]Adopted from http://www.revlib.org/realizations.php?lib=1.

[4]The authors use the term 'software cost' to indicate the number of CPU instructions. We specifically choose the term CPU cost, since GPU can also be termed as software.

[5]Obtained from https://ceit.aut.ac.ir/QDA/RCV.htm.

[6]One may refer to the ongoing NIST competition on lightweight cryptography: https://csrc.nist.gov/projects/lightweight-cryptography.

The algorithm of LIGHTER constructs a sequence of logic gates, starting from the identity function (i.e., the trivial SBox – 0123456789ABCDEF, it is denoted by I) in forward direction, and also from the target function, which is the implementation of the SBox using the logic gates from the library $\mathcal{B}$, in the reverse direction, till they intersect (meet-in-the-middle). This MITM strategy is similar to the well-known breadth first search algorithm in the context of graph theory.

With respect to the problem of finding the optimal implementation of the given SBox, the logic operations from $\mathcal{B}$ are used to build the graph. A path from vertex v_1 to vertex v_2 exists only if v_2 can be deduced from v_1 by applying a gate from $\mathcal{B}$. The weight of this edge is the corresponding cost given in the cost metric. Once the graph is constructed, any path from the source vertex I to the destination vertex S is an implementation of the SBox in terms of logic gates from $\mathcal{B}$.

The algorithm constructs the graph by incrementally computing all valid nodes that can be constructed at a distance λ from the vertex that has not been added to the graph yet. Once computed, the vertex can be added and all new nodes can be connected. The algorithm maintains all nodes in separate lists, where nodes at the same distance belong to the same list. This structure facilitates the efficient generation of new nodes. Initially, the value λ is 0 and is incremented as the new vertices are added to the graph. A threshold can be set for λ beyond which the algorithm is terminated and the solution is returned.

In the implementation of the algorithm, a modified version of Dijkstra's shortest path algorithm along with the MITM strategy is used. Dijkstra's algorithm is used to continue expanding the graph until the current best path from I to S, i.e the current best implementation of the SBox cannot be improved any further. A simpler BFS based MITM strategy would not work as the graph is weighted, the solution generated by such an implementation is valid but not necessarily the optimal.

When building the graph, LIGHTER tries all gate combinations up to distance λ. At the same time, since it assumes the SBox is a permutation, it only creates new vertices which are permutations. Interestingly, this idea inherently is applicable for any reversible logic synthesis. This means that LIGHTER is capable of implementing a 4×4 SBox using reversible logic libraries. This observation is the core motivation of this research.

The MITM strategy for computing optimal SBox implementation is effective only for small functions S: $GF(2^c) \rightarrow GF(2^c)$ where $c \leq 4$. When creating new vertices, LIGHTER does an exhaustive search for all SBoxes which are within a distance of λ. This becomes practically infeasible when the size of the SBox is too large. Hence, the authors set the limit up to 4-bit SBoxes. It may be mentioned that few improvements on the implementation of LIGHTER are made to make it more efficient in [17], although the basic algorithm remains the same.

Encoding: In order to facilitate logic operations in a vectorial way, LIGHTER uses a custom encoding scheme for the SBoxes. This encoding converts a given SBox from the 16-entry look-up based representation to a 64-bit Boolean vector. For better clarity, we give an example of the LIGHTER encoding with the PRINCE SBox, BF32AC916780E5D4 in Table II. The look-up based SBox is written row-wise; which is then converted to the corresponding binary vector which is written column-wise; then the table is read row-wise as four 16-bit vectors which are denoted by $z_3, \ldots, z_0$, respectively. Finally, the SBox is encoded as the 64-bit Boolean vector: $z_3 z_2 z_1 z_0$ = CE2A 44CF F8C8 E346. We also show application to XOR and AND operation on this SBox through $z_0 \oplus z_1$ (= 1B8E) and $z_0 \wedge z_1$ (= E040), respectively; which are obtained by applying the respective gates bit-wise.

IV. Reversible SBox Implementation with LIGHTER

A. Scope

1) Advantages: Adopting LIGHTER as the back-end for implementing 4×4 SBoxes has the following major advantages.

- **An end-to-end solution.** This tool is particularly useful since other such tools available are more generic in nature and the user has to tune it to work with this problem. As noted in [5], it (almost) always outperforms other such tools. The source codes are written in pure C++ (no external dependency), and are available as free.
- **No garbage/ancilla lines.** As LIGHTER only considers bijective SBoxes while searching, no additional ancilla/garbage line is created in the process. In fact, because of this reason, it is possible to use LIGHTER as the back-end (the algorithm remains unchanged).

2) Limitations: While working on this project, we encounter the following issues which cannot be handled for reversible logic synthesis using LIGHTER as the back-end.

- **Restricted to gates with classical analogy.** LIGHTER cannot handle gates which do not have any classical computing counterpart. For example, one may consider the Hadamard gate, which is given by the permutation matrix: $\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. This gate is needed to create superposition of states, for example, which could be of prominent importance in quantum computers.
- **No support for multiple output lines.** In general, gates in classical computing have only one output bit. LIGHTER inherently assumes this. For this reason; reversible gates, which have more than one output lines, are not directly supported. For our implementation, we only take one output bit and discard the rest. This does not cause additional cost for gates such as CNOT (as one output line directly passes the corresponding input); but may incur additional cost for gates such as Fredkin (as two out of three output lines do not pass any input). For this reason, we implement the Fredkin gate in two different gates; where one respective non-trivial output line is discarded. Another problem associated is that, certain libraries are not supported in LIGHTER. For example, consider the Peres library (consisting of multiple control Peres gates). One output line is common in both the MCT and the P gates. In this case, LIGHTER ignores the implementation

TABLE II: LIGHTER encoding: Example with PRINCE SBox (BF32AC916780E5D4)

Look-up Based SBox →	B	F	3	2	A	C	9	1	6	7	8	0	E	5	D	4	Encoded SBox ↓
z_3	1	1	0	0	1	1	1	0	0	0	1	0	1	0	1	0	CE2A
z_2	0	1	0	0	0	1	0	0	1	1	0	0	1	1	1	1	44CF
z_1	1	1	1	1	1	0	0	0	1	1	0	0	1	0	0	0	F8C8
z_0	1	1	1	0	0	0	1	1	0	1	0	0	0	1	1	0	E346
$z_0 \oplus z_1$	0	0	0	1	1	0	1	1	1	0	0	0	1	1	1	0	1B8E
$z_0 \wedge z_1$	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	E040

that produces a duplicate output. This makes sense if all the gates have only one output line (which is the case in classical computing), but is inconvenient for reversible computing. For this reason, we omit the implementation of the Peres gates.

- **No support for depth/delay optimization.** The circuit depth can be defined by the minimum number of edges in the shortest path from the start vertex to the destination vertex. Similarly, given a delay metric for the gates in the logic library, the circuit delay can be defined as the sum of the sequential path delays[7]. LIGHTER does not support optimization based on depth/delay; so does not LIGHTER-R.
- **Initial & final bit permutations are considered free.** LIGHTER inherently assumes that any bit permutation done before or after the combinatorial part is free. This makes sense particularly in ASIC; as bit permutations are just wiring, which are taken as free. For CPU, rest of the source codes can be written in such a way those permutations are available automatically. However, such bit swapping is not free in the reversible domain, in general. For this purpose the Swap gate (or three CNOT gates, see Figure 3) may be used, which has a quantum cost of 3. It can be, however, shown that this additional quantum cost will not exceed 12 (since at most initial and 2 final bit swappings can happen).
- **No support for gates with 0 cost.** At times, it may be required to consider gates with 0 cost. For example, the NOT gate has 0 two-qubit cost. In the classical paradigm, one such example could be the multiplicative complexity where the logic library is {AND, XOR}; and the cost for the AND gate is unity, and that of the XOR gate is 0. Through our empirical analysis, we observe that; LIGHTER apparently does not support any logic library which has a gate with 0 cost[8]. In fact, this observation is valid if the cost becomes less than 0.01. For this reason, we set the cost for the NOT gate as 0.01.
- **Fredkin gates are not supported.** With our implementation, we observe that the Fredkin gates are not properly functioning with LIGHTER back-end. This is apparently caused by a bug in the LIGHTER implementation[9].

B. Method

Implementing reversible gates in LIGHTER-R follows the same format as that of the implementation of classical gates in LIGHTER. Consider, for example, the implementation of the CCNOT (Tof_3) taken from the source code of LIGHTER-R given in Code 1. Other reversible gates in the logic library are implemented in a similar fashion. Certain adjustments are made to make LIGHTER-R easy to use. One additional feature we use is to add support for .tfc[10] format (the generated C files can be converted to this format).

C. Results

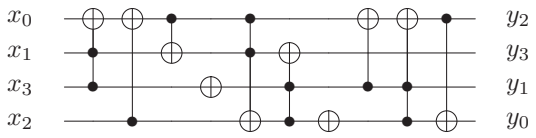

FIG. 4: SKINNY SBox (C6901A2B385D4E7F) with NCT (optimized for quantum cost)

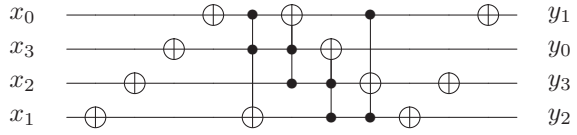

FIG. 5: PICCOLO SBox (E4B238091A7F6C5D) with MCT (optimized for two-qubit cost)

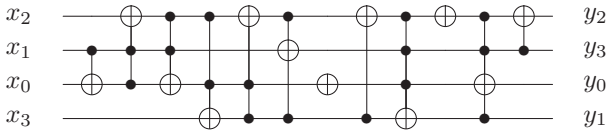

FIG. 6: PRINCE SBox (BF32AC916780E5D4) with MCT (optimized for gate count)

With the backdrop presented, we now show specific examples of LIGHTER-R. Figure 4 shows the reversible implementation

[7]The circuit depth metric can be obtained from the delay metric by setting uniform delay for all logic gates.

[8]In such a case, LIGHTER does not output any implementation.

[9]The following error message is returned:

```
terminate called after throwing an instance of
'std::out_of_range'
what(): map::at
Aborted (core dumped).
```

[10]See http://webhome.cs.uvic.ca/~dmaslov/mach-read.html.

CODE 1: Example of reversible gate implementation in LIGHTER-R: Toffoli (Tof_3) gate

```
fun_tmp.bit_slice[i] = (f.bit_slice[(i+a)%N] & f.bit_slice[(i+b)%N]) ^ f.bit_slice[i];
int _ = f.bit_slice[(i+a)%N]; // Garbage line
int __ = f.bit_slice[(i+b)%N]; // Garbage line
```

of the SKINNY SBox (C6901A2B385D4E7F) with the NCT library, optimized for quantum cost. The cost here is 26 (the gate count is 10, the two-qubit cost is 24). The implementation of the PICCOLO [18] SBox (E4B238091A7F6C5D); with the MCT library, optimized for two-qubit cost, is given in Figure 5. The cost in this case is 20, but it is reported as 20.07 as the NOT gate is implemented with cost 0.01 – see Section IV-A2 for more details (the gate count is 11 and the quantum cost is 27). The MCT implementation of the PRINCE SBox (BF32AC916780E5D4), optimized for gate count, is shown in Figure 6. The cost is 12 (the quantum cost is 54 and the two-qubit cost is 38).

V. CONCLUSION & FUTURE WORKS

In this document, we attempt to implement the symmetric key algorithms optimally using reversible circuits. As the first step, we tweak an existing tool, LIGHTER, which aims at optimally implementing a 4×4 SBox with respect to a given logic library (with a cost metric). The original tool is developed keeping the existing CMOS technology and CPU in mind. The novelty of our work lies in developing a front-end to the original LIGHTER tool, such that now it can work with the common reversible logic gates. The basic features of LIGHTER that are useful in reversible implementation are: 1) Only bijection is considered. 2) No fan-out circuit is generated. 3) Every time a line is replaced by a new content, the old content of the line is used in the corresponding update function.

We believe this work will boost future researches in the relevant fields. As follow-up, we list a few possible directions. The problems mentioned in Section IV-A2 can be tackled (such as, extending LIGHTER-R to support gates with non-classical analogy, optimization for depth/delay, support for 0 cost gates and so on). A complete cipher can be optimized and implemented in the reversible paradigm. Apart from the reversible paradigm, other future technologies (such as, the memristor) can also be considered.

REFERENCES

[1] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, Sep. 1971.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, 1996, pp. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[3] K. Datta, V. Shrivastav, I. Sengupta, and H. Rahaman, "Reversible logic implementation of AES algorithm," in *Proceedings of the 8th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2013, 26-28 March, 2013, Abu Dhabi, UAE*, 2013, pp. 140–144. [Online]. Available: https://doi.org/10.1109/DTIS.2013.6527794

[4] B. Langenberg, H. Pham, and R. Steinwandt, "Reducing the cost of implementing aes as a quantum circuit," Cryptology ePrint Archive, Report 2019/854, 2019, https://eprint.iacr.org/2019/854.

[5] J. Jean, T. Peyrin, S. M. Sim, and J. Tourteaux, "Optimizing implementations of lightweight building blocks," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 4, pp. 130–168, 2017. [Online]. Available: https://doi.org/10.13154/tosc.v2017.i4.130-168

[6] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, July 1961.

[7] A. Bérut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz, "Experimental verification of landauer's principle linking information and thermodynamics," *Nature*, vol. 483, pp. 187–189, 2012.

[8] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev.*, vol. 17, no. 6, pp. 525–532, Nov. 1973. [Online]. Available: http://dx.doi.org/10.1147/rd.176.0525

[9] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, "Reversible logic gate using adiabatic superconducting devices," *Nature Scientific Reports*, vol. 4, 2014. [Online]. Available: http://dx.doi.org/10.1038/srep06354

[10] G. Cattaneo, A. Leporati, and R. Leporini, "Fredkin gates for finite-valued reversible and conservative logics," *Journal of Physics A: Mathematical and General*, vol. 35, no. 46, p. 9755, 2002.

[11] D. Maslov and M. Saeedi, "Reversible circuit optimization via leaving the boolean domain," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 6, pp. 806–816, 2011.

[12] M. Saeedi and I. L. Markov, "Synthesis and optimization of reversible circuits - A survey," *CoRR*, vol. abs/1110.2574, 2011. [Online]. Available: http://arxiv.org/abs/1110.2574

[13] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," in *CHES*, vol. 4727. Springer, 2007, pp. 450–466.

[14] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract," in *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, 2012, pp. 208–225. [Online]. Available: https://doi.org/10.1007/978-3-642-34961-4_14

[15] M. R. Albrecht, B. Driessen, E. B. Kavun, G. Leander, C. Paar, and T. Yalçin, "Block ciphers - focus on the linear layer (feat. PRIDE)," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, 2014, pp. 57–76. [Online]. Available: https://doi.org/10.1007/978-3-662-44371-2_4

[16] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, 2016, pp. 123–153. [Online]. Available: https://doi.org/10.1007/978-3-662-53008-5_5

[17] Z. Bao, J. Guo, S. Ling, and Y. Sasaki, "PEIGEN - a platform for evaluation, implementation, and generation of s-boxes," *IACR Trans. Symmetric Cryptol.*, vol. 2019, no. 1, pp. 330–394, 2019. [Online]. Available: https://doi.org/10.13154/tosc.v2019.i1.330-394

[18] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: An ultra-lightweight blockcipher," in *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, 2011, pp. 342–357. [Online]. Available: https://doi.org/10.1007/978-3-642-23951-9_23