IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2025, No. 2, pp. 192–205. DOI:10.46586/tosc.v2025.i2.192-205

Addendum to How Small Can S-boxes Be?

Yu Sun[1,5], Lixuan Wu[1], Chenhao Jia[2], Tingting Cui[2,3], Kai Hu[1,5,3,4(✉)] and Meiqin Wang[1,3,4]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China
{yu.sun,lixuanwu}@mail.sdu.edu.cn, {kai.hu,mqwang}@sdu.edu.cn
[2] School of Cyberspace, Hangzhou Dianzi University, Hangzhou, China
{222270059,cuitingting}@hdu.edu.cn
[3] State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao, 266237, China
[4] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China
[5] Quan Cheng Laboratory, Jinan, China

Abstract. In ToSC 2025(1), Jia et al. proposed an SAT-aided automatic search tool for the S-box design. A part of the functionality of this tool is to search for implementations of an S-box with good area and gate-depth complexity. However, it is well-known that the gate depth complexity cannot precisely reflect the latency of an implementation. To overcome this problem, Rasoolzadeh introduced the concept of *latency complexity*, a more precise metric for the latency cost of implementing an S-box than the gate depth complexity in the real world.

In this addendum, we adapt Jia et al.'s tool to prioritize latency as the primary metric and area as the secondary metric to search for good implementations for existing S-boxes. The results show that the combination of Jia et al.'s tool and Rasoolzadeh's latency complexity can lead to lower-latency S-box implementations. For S-boxes used in `LBlock`, `Piccolo`, `SKINNY-64`, `RECTANGLE`, `PRESENT` and `TWINE`, which are popular targets in this research line, we find new implementations with lower latency. We conducted synthesis comparisons of the area and latency under multiple standard libraries, where our results consistently outperformed in terms of latency. For example, for `LBlock-S`$_0$, our solution reduces latency by around 50.0% $\sim$ 73.8% compared to previous implementations in TSMC 90nm library with the latency-optimized synthesis option.

Keywords: S-box · low-latency · automatic search · SAT

1 Introduction

In recent years, the design and implementation of low-latency cryptographic primitives have garnered significant research interest. Among these efforts, minimizing the latency of S-box implementations has emerged as a pivotal challenge for achieving efficient real-time cryptographic primitives. While a multitude of automated tools have been developed to explore optimal S-box implementations, existing solutions predominantly prioritize area optimization over latency performance.

In [JPST17], Jean et al. introduced the tool `LIGHTER`, which is an open-source tool that can search for small-area implementations for S-boxes. At FSE 2016, Stoffelen [Sto16] used the SAT tool to find S-box implementations with small areas, but only two-input gates and the INV gates were considered. Lu et al. [LWH+21] extended Stoffelen's tool by additionally considering more complex gates such as the 4-input gate `MAOI1` and they managed to find S-box implementations with even smaller areas than Stoffelen's tool. Most

Received: 2025-03-01 Accepted: 2025-05-01 Published: 2025-06-11

recently, Jia et al. [JCL+25] improved the SAT search algorithm and better results are found. It is worth mentioning that Jia et al.'s tool does not only consider the area metric, but also the gate-depth complexity as the secondary metric.

Typically, the gate-depth complexity is defined as the minimum length of the longest path from an input bit to an output bit across all possible implementations of an S-box. This metric is used to mathematically model the lowest latency for implementing the S-box. However, since different types of gates have different latency costs, this definition is generally considered a coarse estimation of the minimum latency cost for hardware implementation. In [Ras22], Rasoolzadeh introduced the concept of *latency complexity*, a more precise metric for the latency cost of implementing a function compared to gate-depth complexity.

The latency complexity is defined as the minimum value for the longest path concerning the number of only NAND and NOR gates from any input to any output for implementing the function while the set of allowed gates to use is {INV, NAND, NOR}. According to [Ras22], except for the INV gate, whose fan-in number is one, the 2-bit NAND gate and the 2-bit NOR gate have the minimum latency, in almost all ASIC libraries. Since {INV, NAND, NOR} has the completeness property, we can construct any (vectorial) Boolean function using these three gates. At the same time, the latency of circuits implemented using this basis will be very small.

Rasoolzadeh present an algorithm to find the smallest latency complexity for many simple Boolean functions and S-boxes. Thus, for a low-latency S-box implementation, we can in theory implement each coordinate function of the S-box with the smallest-latency-complexity circuit from Rasoolzadeh's tool. However, simply running Rasoolzadeh's algorithm multiple times to generate circuits for all S-box output bits would likely result in independent circuits for each output bit, with no shared logic between them. This approach would lead to a significant increase in area metric.

Our contributions. We combine Jia et al.'s tool and the latency complexity ideas in this paper, and provide new hardware implementations for some popular S-boxes with the state-of-the-art latency numbers synthesized in different libraries. Unsurprisingly, our implementations for S-boxes used in `LBlock`, `Piccolo`, `SKINNY-64`, `RECTANGLE`, `PRESENT` and `TWINE`, which are popular targets in this research line, consistently outperform implementations from other tools such as `LIGHTER`, Stoffelen's, Lu et al.'s and Jia et al.'s SAT tools. For example, for $\texttt{LBlock-S}_0$, our solution reduces latency by around 50.0% $\sim$ 73.8% compared to previous implementations in TSMC 90nm library with the latency-optimized synthesis option. Additionally, compared to Rasoolzadeh's algorithm, our implementations are significantly better in the area metric[1].

2 Low-Latency Implementation Search for S-boxes

We utilize Jia et al.'s automated search method [JCL+25], but the scope of gates to be encoded is now limited to {INV, NAND, NOR} gates. Notably, an INV gate can be equivalently represented as a NAND gate or a NOR gate with two identical inputs. Therefore, in practice, we only need to encode two types of gates, namely {NAND, NOR} [2]. Since INV gate has significantly lower latency compared to the NAND and NOR gates, we exclude it from the latency complexity calculation.

Before starting the search process, we define G as the number of gates and D as the maximum latency complexity for the S-box implementation. The search algorithm adopts

[1] In fact, Rasoolzadeh's algorithm only finds circuits with the minimum latency complexity for a given Boolean function and does not consider the area metric.

[2] If the automated tool assigns two equal values to the two inputs of NAND or NOR, we count this gate as INV.

Algorithm 1: Automatic search model for searching S-box circuits with a given number of gates G and latency complexity D.

Input: n: the length of the S-box. $\mathsf{Sbox}[\cdot]$: an n-bit to n-bit S-box. G: the number of gates. D: the latency complexity.
Output: A search model for implementing S-box circuits.

for $x \leftarrow 0$ **to** $2^n - 1$ **do**
  $x = x_0 \| x_1 \| \dots \| x_{n-1}$;
  $\mathsf{Sbox}(x) = y_0 \| y_1 \| \dots \| y_{n-1}$;
  for $i \leftarrow 0$ **to** $G - 1$ **do**
    $q_{2i} = (\sum_{j=0}^{n-1} a_{0,i}^{j} \cdot x_j) + (\sum_{j=0}^{i-1} a_{0,i}^{n+j} \cdot t_j)$ `// inputs of current gate`
    $q_{2i+1} = (\sum_{j=0}^{n-1} a_{1,i}^{j} \cdot x_j) + (\sum_{j=0}^{i-1} a_{1,i}^{n+j} \cdot t_j)$ `// inputs of current gate`
    if $b_i = 0$ **then** `// current gate is NAND(INV)`
      $t_i = \neg(q_{2i} \cdot q_{2i+1})$ `// output of current gate`
    else `// current gate is NOR(INV)`
      $t_i = \neg(q_{2i} \vee q_{2i+1})$ `// output of current gate`
    $tempd_0 \leftarrow$ latency complexity of q_{2i};
    $tempd_1 \leftarrow$ latency complexity of q_{2i+1};
    if $a_{0,i}^{0} \| \dots \| a_{0,i}^{n+i-1} = a_{1,i}^{0} \| \dots \| a_{1,i}^{n+i-1}$ **then** `// current gate is INV`
      $d_i = tempd_0$ `//` $q_{2i} = q_{2i+1},\ tempd_0 = tempd_1$
    else `// current gate is NAND(NOR)`
      $d_i = \max(tempd_0, tempd_1) + 1$
  for $i \leftarrow 0$ **to** $n - 1$ **do**
    $y_i = (\sum_{j=0}^{n-1} c_i^j \cdot x_j) + (\sum_{j=0}^{G-1} c_i^{n+j} \cdot t_j)$;
    $e_i \leftarrow$ latency complexity of y_i;
for $i \leftarrow 0$ **to** $G - 1$ **do**
  $\sum_{j=0}^{n+i-1} a_{0,i}^{j} = 1$; $\sum_{j=0}^{n+i-1} a_{1,i}^{j} = 1$; $a_{0,i}^{0} \| \dots \| a_{0,i}^{n+i-1} \geq a_{1,i}^{0} \| \dots \| a_{1,i}^{n+i-1}$;
for $i \leftarrow 0$ **to** $n - 1$ **do**
  $\sum_{j=0}^{n+G-1} c_i^j = 1$;
  $e_i \leq D$;

a gate-by-gate iterative approach: for each gate of the G gates, denoted by $g_0, g_1, \dots, g_{N-1}$, its inputs may originate from either the original S-box inputs or the outputs of preceding gates. Specifically, during the processing of gate g_i, there are $n + i$ candidate input values, where n is the number of orginal input bits of the S-box and i is the number of previous gate outputs. The whole algorithm is provided in Algorithm 1.

The depth D of Algorithm 1 is set according to [Ras22]. We set an initial value for G, and reduce G one by one, until the tool returns infeasible. In practice, the search time might be long, so we set a time limit of days. If the algorithm fails to finish for G gates in 4 days, the implementation of $G + 1$ gates is taken.

3 Results

We apply our algorithm to the S-boxes used in `LBlock`, `Piccolo`, `SKINNY-64`, `RECTANGLE`, `PRESENT` and `TWINE` and obtain their implementations, the number of gates G and the latency complexity D as shown in column "Model" in Table 1 and Table 2.

For the synthesis experiments, we used *Synopsys Design Compiler T-2022.03-SP2* with the synthesis option set to "compile_ultra -no_autoungroup -no_boundary_optimization". We synthesized the S-boxes used in `LBlock`, `Piccolo`, `SKINNY-64`, `RECTANGLE`, `PRESENT`

Table 1: Comparison of area-optimized in the TSMC 90nm. The rows with ✓means that there is no other implementations with less gates.

S-box		Methods [JPST17]	[Sto16]	[LWH+21]	[JCL+25]	Ours	Model G	D	opt.
LBlock S_0	A(GE)	16.25	16.75	16.25	16.25	27.25	28	4	✓
	L(ns)	0.51	0.36	0.67	0.32	0.20			
	P(μW)	4.4998	4.8249	4.3757	4.8136	5.1478			
	E(fJ)	2.2949	1.7370	2.9317	1.5404	1.0296			
Piccolo	A(GE)	12.75	12.50	12.75	12.75	25.50	26	4	✓
	L(ns)	0.31	0.22	0.31	0.26	0.16			
	P(μW)	3.1298	3.2214	3.1206	3.3455	4.6631			
	E(fJ)	0.9702	0.7087	0.9674	0.8698	0.7461			
SKINNY-64	A(GE)	13.00	12.25	13.00	13.00	27.00	28	4	✓
	L(ns)	0.32	0.22	0.32	0.32	0.19			
	P(μW)	3.3349	3.1606	3.3382	3.3349	4.9072			
	E(fJ)	1.0672	0.6953	1.0682	1.0672	0.9324			
RECTANGLE	A(GE)	18.25	18.50	18.00	18.00	48.25	53	4	✓
	L(ns)	0.46	0.33	0.61	0.46	0.24			
	P(μW)	6.0852	6.7434	6.4543	6.6218	9.7273			
	E(fJ)	2.7992	2.2253	3.9371	3.0460	2.3346			
PRESENT	A(GE)	21.25	-	-	-	46.25	49	4	
	L(ns)	0.77	-	-	-	0.21			
	P(μW)	8.8287	-	-	-	8.7407			
	E(fJ)	6.7981	-	-	-	1.8355			
TWINE	A(GE)	21.50	-	-	-	39.25	41	4	
	L(ns)	0.69	-	-	-	0.17			
	P(μW)	6.9411	-	-	-	7.0277			
	E(fJ)	4.7894	-	-	-	1.1947			

and TWINE using the TSMC 90nm library and compared the circuits generated by five distinct methods, i.e., ours and the methods in [JPST17, Sto16, LWH+21, JCL+25].

The first method is based on [JPST17], which uses a graph-based search algorithm for small-area circuits but cannot guarantee optimality. The second method follows [Sto16], formulating S-box implementation as a SAT problem to minimize gate count. The third method, from [LWH+21], improves on Stoffelen's work to optimize area under a standard cell library. The fourth method, from [JCL+25], considers both area and depth complexity. The fifth method is our algorithm, which uses automated search to find minimal gate count and lowest latency implementations under the {INV, NAND, NOR} basis.

The synthesis results in TSMC 90nm library are summarized in Table 1 and Table 2, for area-optimized synthesis and latency-optimized synthesis respectively. For more comparison results in other libraries, please refer to the Appendix A.

From Table 1 where the area-optimized option is used in synthesis, the results show our solution's significant latency superiority. For LBlock-S_0, our solution reduces latency by 60.8% compared to [JPST17], 44.4% compared to [Sto16], 70.1% compared to [LWH+21], and 37.5% compared to [JCL+25]. Similar substantial improvements are observed for Piccolo, SKINNY-64, RECTANGLE, PRESENT and TWINE S-boxes.

As shown in Table 2 with latency-optimized synthesis, our solution also demonstrates superior performance compared to previous works. For LBlock-S_0, it reduces the previous best latency (achieved by [Sto16]) by 50.0% while simultaneously lowering energy consumption by 66.6%. This dual optimization is consistently observed across all tested S-boxes, while maintaining competitive area efficiency. Our solution provides a more balanced and efficient approach to cryptographic S-box design.

We analyzed the reasons why our circuits perform better in terms of latency. According

Table 2: Comparison of latency-optimized in the TSMC 90nm. The rows with ✓means that there is no other implementations with less gates.

S-box		Methods					Model		
		[JPST17]	[Sto16]	[LWH+21]	[JCL+25]	Ours	G	D	opt.
LBlock-S_0	A(GE)	32.50	32.00	34.25	28.50	33.25	28	4	✓
	L(ns)	0.33	0.22	0.42	0.23	0.11			
	P(µW)	16.0231	13.8856	21.3766	13.3583	9.2687			
	E(fJ)	5.2876	3.0548	8.9782	3.0724	1.0196			
Piccolo	A(GE)	35.50	35.50	35.50	40.25	33.75	26	4	✓
	L(ns)	0.15	0.13	0.15	0.13	0.10			
	P(µW)	14.9259	14.2635	14.9878	17.4565	9.2445			
	E(fJ)	2.2389	1.8543	2.2482	2.2693	0.9245			
SKINNY-64	A(GE)	38.50	26.00	38.50	38.50	30.75	28	4	✓
	L(ns)	0.15	0.14	0.15	0.15	0.11			
	P(µW)	16.1620	10.7082	16.1472	16.1620	8.2156			
	E(fJ)	2.4243	1.4991	2.4221	2.4243	0.9037			
RECTANGLE	A(GE)	26.50	39.75	27.00	35.25	61.75	53	4	✓
	L(ns)	0.29	0.21	0.39	0.27	0.12			
	P(µW)	15.3718	20.8294	18.6513	22.1446	20.4166			
	E(fJ)	4.4578	4.3742	7.2740	5.9790	2.4500			
PRESENT	A(GE)	45.50	-	-	-	54.25	49	4	
	L(ns)	0.47	-	-	-	0.12			
	P(µW)	30.2626	-	-	-	15.2059			
	E(fJ)	14.2234	-	-	-	1.8247			
TWINE	A(GE)	36.75	-	-	-	52.25	41	4	
	L(ns)	0.44	-	-	-	0.10			
	P(µW)	22.7494	-	-	-	15.3318			
	E(fJ)	10.0097	-	-	-	1.5332			

to [Ras22], all circuits can be implemented using the basis {INV, NAND, NOR} to achieve a small latency. Moreover, our automated search has significantly reduced the area of the circuits generated in this process.

4 Conclusion and Discussion

In this work, we enhance Jia et al.'s SAT-based S-box search tool by integrating Rasoolzadeh's latency complexity metric, yielding optimized low-latency implementations for S-boxes in several lightweight ciphers (e.g., LBlock, Piccolo). The synthesis results demonstrate significant improvements in both latency and energy consumption for most cases.

However, for certain low-latency S-boxes (see Appendix B), our method shows no advantage over LUT-based implementations. This limitation may be due to two key factors: (1) the already highly optimized nature of these S-boxes leaves minimal room for further improvement in latency, and (2) the current gate-type constraints {INV, NAND, NOR} in our SAT model may be insufficient to capture the full optimization potential of diverse gate combinations. While adding more gate-types for SAT tool is feasible, accurately modeling their latency interactions remains challenging.

Furthermore, prior work implies that limiting INV gates reduces energy consumption in cryptographic circuits. Therefore, integrating INV gate constraints into our SAT-based framework presents a promising direction for future work.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This research is supported by the National Key R&D Program of China(Grant No. 2024YFA1013000, 2023YFA1009500), the National Natural Science Foundation of China (Grant No. 62032014, U2336207), Department of Science & Technology of Shandong Province (No.SYS202201), Quan Cheng Laboratory (Grant No. QCLZD202301, QCLZD202306). Kai Hu is supported by the National Cryptologic Science Fund of China (2025NCSF02007), the National Natural Science Foundation of China (62402283), the Natural Science Foundation of Shandong Province (2025HWYQ-025), the Natural Science Foundation of Jiangsu Province (BK20240420) and Program of Qilu Young Scholars of Shandong University. Tingting Cui is specially supported by the Open Project Program from Key Laboratory of Cryptologic Technology and Information Security (Ministry of Education), Shandong University.

References

[JCL+25] Chenhao Jia, Tingting Cui, Qing Ling, Yan He, Kai Hu, Yu Sun, and Meiqin Wang. How small can s-boxes be? *IACR Trans. Symmetric Cryptol.*, 2025(1):592–622, 2025.

[JPST17] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Trans. Symmetric Cryptol.*, 2017(4):130–168, 2017.

[LWH+21] Zhenyu Lu, Weijia Wang, Kai Hu, Yanhong Fan, Lixuan Wu, and Meiqin Wang. Pushing the limits: Searching for implementations with the smallest area for lightweight s-boxes. *IACR Cryptol. ePrint Arch.*, page 1644, 2021.

[Ras22] Shahram Rasoolzadeh. Low-latency boolean functions and bijective s-boxes. *IACR Trans. Symmetric Cryptol.*, 2022(3):403–447, 2022.

[Sto16] Ko Stoffelen. Optimizing s-box implementations for several criteria using SAT solvers. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 140–160. Springer, 2016.

A More Comparison Results

Table 3: Comparison of area-optimized in the NanGate 45nm. The rows with ✓means that there is no other implementations with less gates.

S-box		Methods					Model		
		[JPST17]	[Sto16]	[LWH+21]	[JCL+25]	Ours	G	D	opt.
LBlock S_0	A(GE)	18.00	17.00	18.33	16.33	27.00	28	4	✓
	L(ns)	0.33	0.25	0.42	0.22	0.13			
	P(µW)	6.4549	6.0492	6.4368	6.0752	6.7219			
	E(fJ)	2.1301	1.5123	2.7035	1.3365	0.8738			
Piccolo	A(GE)	14.33	12.67	14.33	13.00	25.33	26	4	✓
	L(ns)	0.17	0.15	0.17	0.15	0.11			
	P(µW)	4.5506	4.0721	4.5367	4.2301	6.0227			
	E(fJ)	0.7736	0.6108	0.7712	0.6345	0.6625			
SKINNY-64	A(GE)	14.67	12.33	14.67	14.67	26.67	28	4	✓
	L(ns)	0.17	0.17	0.17	0.17	0.12			
	P(µW)	4.8327	4.1525	4.8379	4.8327	6.2515			
	E(fJ)	0.8216	0.7059	0.8224	0.8216	0.7502			
RECTANGLE	A(GE)	20.67	18.67	20.33	18.00	47.33	53	4	✓
	L(ns)	0.34	0.25	0.42	0.32	0.18			
	P(µW)	8.8282	8.5194	10.0826	8.4577	12.2843			
	E(fJ)	3.0016	2.1298	4.2347	2.7065	2.2112			
PRESENT	A(GE)	23.67	-	-	-	45.33	49	4	
	L(ns)	0.54	-	-	-	0.15			
	P(µW)	12.4276	-	-	-	11.0258			
	E(fJ)	6.7109	-	-	-	1.6539			
TWINE	A(GE)	24.33	-	-	-	38.67	41	4	
	L(ns)	0.44	-	-	-	0.13			
	P(µW)	10.4576	-	-	-	8.8497			
	E(fJ)	4.6013	-	-	-	1.1505			

Table 4: Comparison of latency-optimized in the NanGate 45nm. The rows with ✓means that there is no other implementations with less gates.

S-box		Methods					Model		
		[JPST17]	[Sto16]	[LWH+21]	[JCL+25]	Ours	G	D	opt.
LBlock S_0	A(GE)	31.67	33.67	35.00	39.67	28.33	28	4	✓
	L(ns)	0.20	0.16	0.25	0.15	0.08			
	P(µW)	18.9471	19.0453	23.3128	22.6263	7.7239			
	E(fJ)	3.7894	3.0472	5.8282	3.3939	0.6179			
Piccolo	A(GE)	26.00	27.67	26.00	16.67	28.00	26	4	✓
	L(ns)	0.10	0.10	0.10	0.12	0.07			
	P(µW)	12.8767	12.0513	12.8603	6.6460	7.9148			
	E(fJ)	1.2877	1.2051	1.2860	0.7975	0.5540			
SKINNY-64	A(GE)	26.67	18.33	26.67	26.67	29.67	28	4	✓
	L(ns)	0.10	0.11	0.10	0.10	0.07			
	P(µW)	13.2346	7.4483	13.2239	13.2346	8.7282			
	E(fJ)	1.3235	0.8193	1.3224	1.3235	0.6110			
RECTANGLE	A(GE)	31.00	25.00	33.33	37.00	52.67	53	4	✓
	L(ns)	0.18	0.18	0.24	0.20	0.08			
	P(µW)	23.6798	15.0182	29.1481	29.0527	17.5255			
	E(fJ)	4.2624	2.7033	6.9955	5.8105	1.4020			
PRESENT	A(GE)	42.67	-	-	-	51.33	49	4	
	L(ns)	0.28	-	-	-	0.08			
	P(µW)	32.1321	-	-	-	15.6089			
	E(fJ)	8.9970	-	-	-	1.2487			
TWINE	A(GE)	39.67	-	-	-	45.33	41	4	
	L(ns)	0.28	-	-	-	0.07			
	P(µW)	28.8655	-	-	-	12.3019			
	E(fJ)	8.0823	-	-	-	0.8611			

Table 5: Comparison of area-optimized in the UMC 55nm. The rows with ✓means that there is no other implementations with less gates.

S-box		Methods					Model		
		[JPST17]	[Sto16]	[LWH+21]	[JCL+25]	Ours	G	D	opt.
LBlock S_0	A(GE)	17.00	20.50	16.25	20.75	27.25	28	4	✓
	L(ns)	0.55	0.40	0.67	0.36	0.19			
	P(µW)	6.2255	7.1976	5.9208	7.5492	7.0804			
	E(fJ)	3.4240	2.8790	3.9669	2.7177	1.3453			
Piccolo	A(GE)	12.75	14.75	12.75	15.75	25.50	26	4	✓
	L(ns)	0.30	0.26	0.30	0.27	0.15			
	P(µW)	4.1034	4.8655	4.0907	5.1651	6.4207			
	E(fJ)	1.2310	1.2650	1.2272	1.3946	0.9631			
SKINNY-64	A(GE)	13.00	13.00	13.00	13.00	27.00	28	4	✓
	L(ns)	0.29	0.24	0.29	0.29	0.19			
	P(µW)	4.3130	4.3493	4.3182	4.3130	6.7344			
	E(fJ)	1.2508	1.0438	1.2523	1.2508	1.2795			
RECTANGLE	A(GE)	18.25	20.00	18.00	23.25	48.25	53	4	✓
	L(ns)	0.49	0.40	0.65	0.48	0.26			
	P(µW)	7.6396	9.0433	8.6734	10.2401	13.1188			
	E(fJ)	3.7434	3.6173	5.6377	4.9152	3.4109			
PRESENT	A(GE)	23.75	-	-	-	46.25	49	4	
	L(ns)	0.79	-	-	-	0.22			
	P(µW)	12.4037	-	-	-	11.9440			
	E(fJ)	9.7989	-	-	-	2.6277			
TWINE	A(GE)	21.50	-	-	-	39.25	41	4	
	L(ns)	0.68	-	-	-	0.19			
	P(µW)	9.2117	-	-	-	9.5478			
	E(fJ)	6.2640	-	-	-	1.8141			

Table 6: Comparison of latency-optimized in the UMC 55nm. The rows with ✓means that there is no other implementations with less gates.

S-box		Methods					Model		
		[JPST17]	[Sto16]	[LWH+21]	[JCL+25]	Ours	G	D	opt.
LBlock S_0	A(GE)	64.50	60.50	73.25	29.25	50.75	28	4	✓
	L(ns)	0.31	0.25	0.40	0.30	0.10			
	P(µW)	26.1807	21.4405	32.1908	10.0536	11.8675			
	E(fJ)	8.1160	5.3601	12.8763	3.0161	1.1867			
Piccolo	A(GE)	38.25	56.25	38.25	34.00	54.25	26	4	✓
	L(ns)	0.16	0.15	0.16	0.19	0.09			
	P(µW)	10.5730	16.2056	10.6541	11.4665	12.7408			
	E(fJ)	1.6917	2.4308	1.7047	2.1786	1.1467			
SKINNY-64	A(GE)	45.75	33.50	45.75	45.75	45.50	28	4	✓
	L(ns)	0.15	0.17	0.15	0.15	0.10			
	P(µW)	13.5258	12.0729	13.5156	13.5258	10.8074			
	E(fJ)	2.0289	2.0524	2.0273	2.0289	1.0807			
RECTANGLE	A(GE)	46.25	40.25	62.00	70.00	96.75	53	4	✓
	L(ns)	0.27	0.29	0.37	0.31	0.11			
	P(µW)	19.3770	19.4352	34.4339	36.1315	25.5495			
	E(fJ)	5.2318	5.6362	12.7405	11.2008	2.8104			
PRESENT	A(GE)	66.50	-	-	-	81.50	49	4	
	L(ns)	0.45	-	-	-	0.11			
	P(µW)	35.0182	-	-	-	20.0234			
	E(fJ)	15.7582	-	-	-	2.2026			
TWINE	A(GE)	66.50	-	-	-	90.25	41	4	
	L(ns)	0.44	-	-	-	0.09			
	P(µW)	31.4058	-	-	-	21.5401			
	E(fJ)	13.8186	-	-	-	1.9386			

B Application in Low-lantency S-boxes

Table 7: Comparison of area-optimized in the NanGate 45nm.

S-box		Methods LUT-based	Ours
QARMA σ_0	A(GE)	14.00	19.33
	L(ns)	0.12	0.12
QARMA σ_1	A(GE)	15.67	28.67
	L(ns)	0.16	0.11
QARMA σ_2	A(GE)	19.33	28.67
	L(ns)	0.11	0.14
MIDORI S_0	A(GE)	13.33	21.00
	L(ns)	0.08	0.09
MIDORI S_1	A(GE)	15.33	19.33
	L(ns)	0.09	0.10
PRINCE	A(GE)	14.67	22.33
	L(ns)	0.13	0.12

Table 8: Comparison of latency-optimized in the NanGate 45nm.

S-box		Methods LUT-based	Ours
QARMA σ_0	A(GE)	17.33	21.67
	L(ns)	0.04	0.07
QARMA σ_1	A(GE)	19.33	31.33
	L(ns)	0.07	0.07
QARMA σ_2	A(GE)	25.33	34.00
	L(ns)	0.05	0.08
MIDORI S_0	A(GE)	19.67	26.33
	L(ns)	0.04	0.05
MIDORI S_1	A(GE)	18.00	20.67
	L(ns)	0.06	0.07
PRINCE	A(GE)	31.67	25.67
	L(ns)	0.04	0.06

C Implementation of Some S-boxes

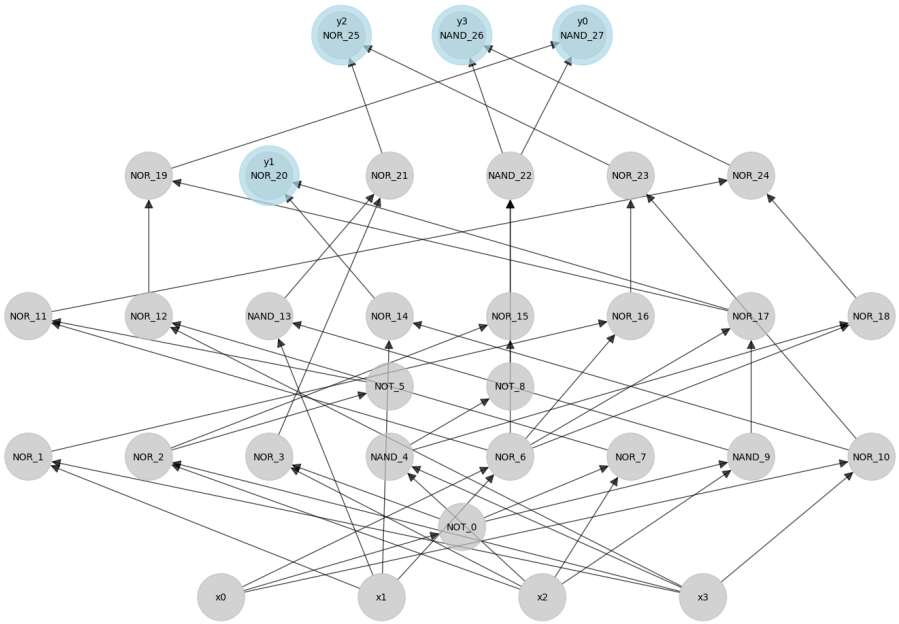

Figure 1: Implementation of `LBlock` S_0.

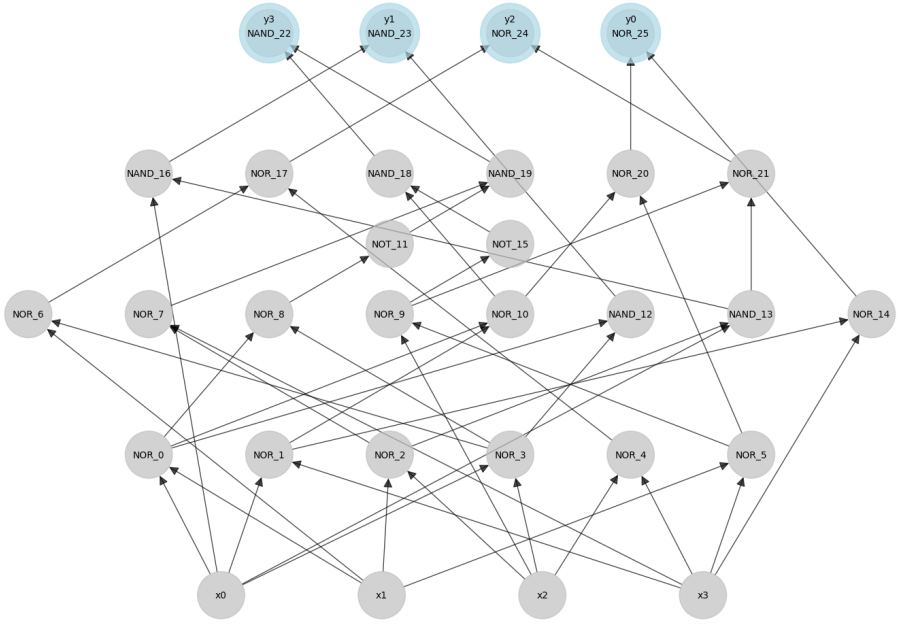

Figure 2: Implementation of `Piccolo` S-box.

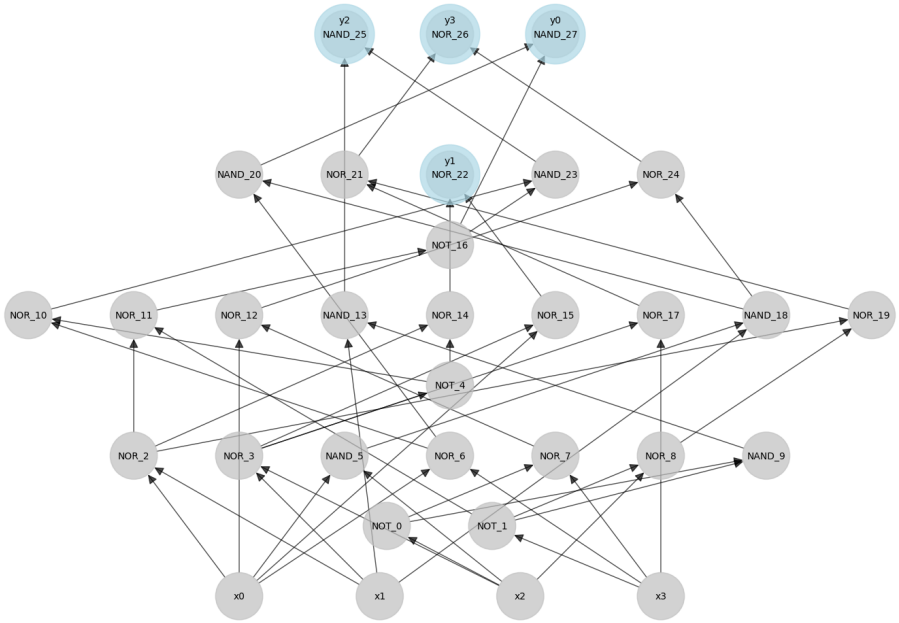

Figure 3: Implementation of `SKINNY-64` S-box.

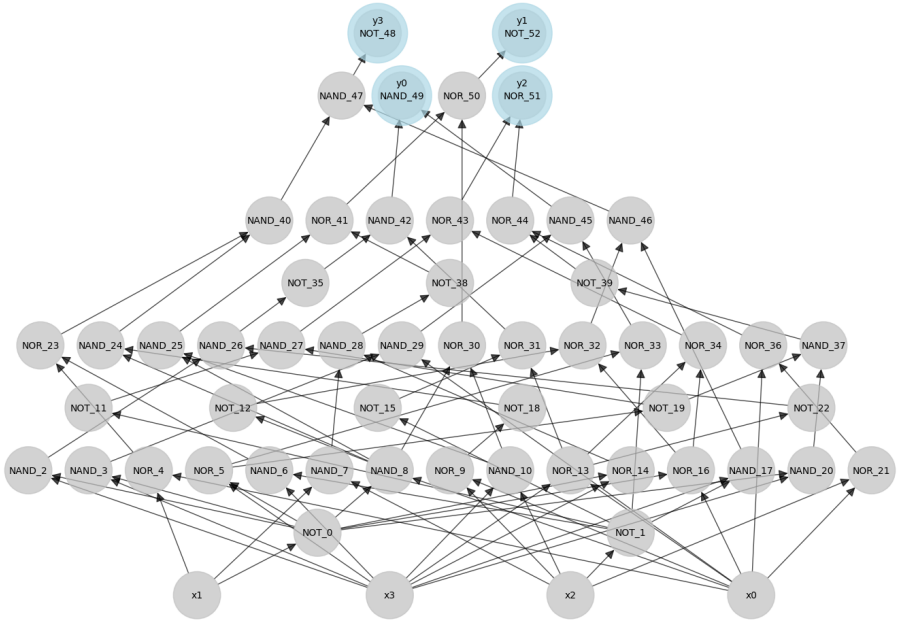

Figure 4: Implementation of `RECTANGLE` S-box.

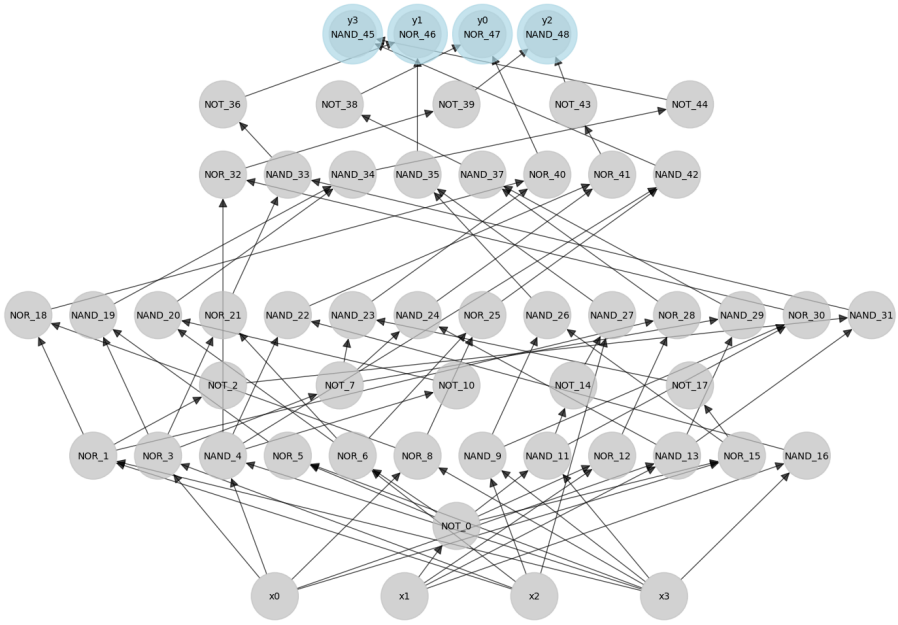

Figure 5: Implementation of `PRESENT` S-box.

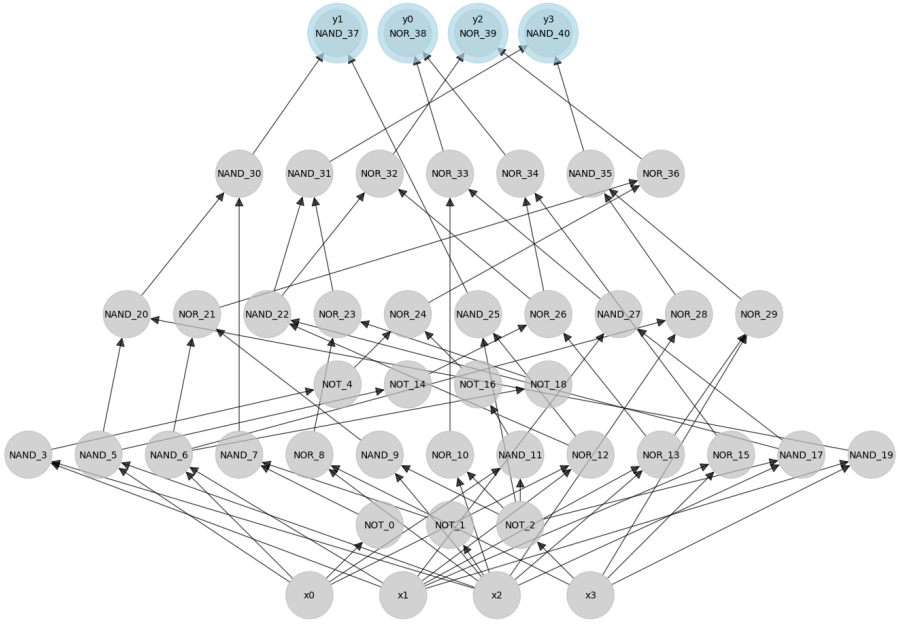

Figure 6: Implementation of `TWINE` S-box.