Area-Throughput Efficient Implementations of CRAFT Cipher For Internet of Vehicles

Jiahao Xiang[1,2] and Lang Li[1,2*]

[1*]College of Computer Science and Technology, Hengyang Normal University, Hengyang, 421002, China.
[2]Hunan Provincial Key Laboratory of Intelligent Information Processing and Application, Hengyang Normal University, Hengyang, 421002, China.

*Corresponding author(s). E-mail(s): lilang911@126.com;
Contributing authors: simple.xjh@qq.com;

Abstract

Purpose: With extraordinary growth in the Internet of Vehicles (IoV), the amount of data exchanged between IoV devices is growing at an unprecedented scale. Most of the IoV devices need instant response and real-time security to ensure the safety of users. The CRAFT cipher that is a lightweight block cipher for low-area can be used in IoV devices. In order to better adapt to these environment, the objective of this paper is to explore opportunities to optimize area and throughput for CRAFT cipher targeted for low-resource IoV devices. **Methods:** A novel compact CRAFT implementation is proposed in serialized fashion to achieve a small hardware footprint. We propose novel unrolled structure of CRAFT cipher for the high throughput feature. **Results:**The results on Artix-7 show that ... **Conclusion:** Hence, our works let CRAFT cipher more suitable for IoV devices.

Keywords: Lightweight block cipher, Internet of Vehicles, Field-programmable gate array(FPGA), Low-area, High-throughput

1 Introduction

Internet of Vehicles (IoV) is an emerging concept in intelligent transportation systems (ITS) to enhance the existing capabilities of VANETs by integrating with the Internet of Things (IoT) Sharma and Kaushik (2019). As IoT technology continues to advance, IoV technology is also making great progress. But the same security issues that exist in IoT are also were introduced into IoV. At the some time, IoV involves a huge amount of dynamic real-time critical data so its security is a major concern.

Lightweight cryptography is a subfield of cryptography that aims to provide solutions tailored for resource-constrained devices McKay et al (2016). It can provides security with low resource consumption and low delay in IoV environment.

In this work, we propose the three architectures of FPGA implementations for the CRAFT Beierle et al (2019), respectively Round based, Serial, and Loop unrolled. This allows IoV practitioners to select the architectures that best suit their needs. The contributions of this article can be summarized as follows.

The rest of this article is organized as follows. Section 2 presents specification of CRAFT; the proposed the three architectures of FPGA implementations for the CRAFT are present in Section 3; Section 4 presents the implementation results, analysis, and comparison with other similar works; finally, the work is concluded in Section 5.

2 Specification of CRAFT

All notations used in this paper are shown in Table 1. CRAFT is a lightweight tweakable block cipher made out of involutory building blocks. It consists of a 64-bit block, a 128-bit key, and a 64-bit tweak. In this cipher, a 64-bit input plaintext P is transformed into a 64-bit output ciphertext C using a 128-bit key K and a 64-bit tweak T. Figure 1 shows the structure of CRAFT.

Table 1 Notations used in this paper

Notation	Description
TK	64-bit tweakeys
RC_i	64-bit round constant in the i^{th} round
R_i, R'_i	Round function
$\oplus$	Bit-wise sum (XOR)

3 Proposed Architecture

To achieve efficient area and throughput, we have, for the first time, optimized the components of CRAFT and proposed two implementation architectures: Serial and Loop Unrolled.

3.1 Serial Architecture(A1)

Compared to round-based architecture, serial architecture are able to reuse components and significantly reduce area usage, e.g., the number of S-box is reduced from 16 to 1. The clock gating technique is also used to enable each component and reduce the

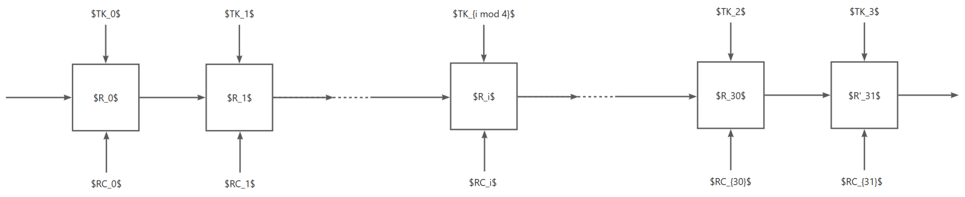

Fig. 1 Structure of CRAFT

energy consumption of encryption. Our proposed architecture is presented in Figure 2. The design includes one Sub-Box, one 4-bit Mix-columns, two register banks for storing keys (called Key-Register) and plaintext (called State-Register), which also act as temporary registers for storing the intermediate results. In order to store intermediate results into State-Register bank, the design has one feedback paths. PermuteNibbles is included in State-Register bank. It is noticeable that since the execution of permute requires 64-bit, in order to reuse the State-Register block, we change the order of execution of Sub-Box and Permute. And the first round of encryption process through the control signal to avoid Permute operation, to ensure the correctness of the encryption algorithm.

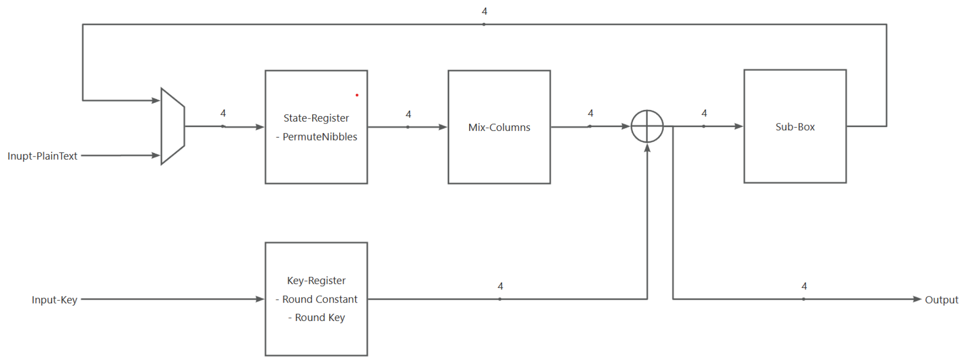

Fig. 2 Serial architecture of CRAFT

3.1.1 Sub-Box Optimization

Sub Box provides a confusing characteristic for the entire encryption algorithm, however it requires a large amount of area. There are different methods of implementation of Sub Box. The most popular implementation is using a lookup table (LUT), such as Lara-Nino et al (2017). However it uses a lot of flip-flop, which will bring a lot of area consumption. Using S-Box's equivalent logical expression for this will reduce area consumption, such as Bao et al (2019), Feng et al (2023).

SAT solvers can be used to find S-Box that satisfy certain implement, such as being resistant to software or hardware implement. In more detail, the S-Box implement can be encoded as Boolean constraints by representing the S-Box as a truth table and then using Boolean variables to represent the input and output bits of the S-Box. The constraints can then be formulated based on the desired implement of the S-Box. Once the S-Box implement are encoded as Boolean constraints, a SAT solver can be used to find a satisfying assignment to these constraints, which corresponds to an S-Box that satisfies the desired implement.

The gate equivalent complexity(GEC) of a SAT instance is the number of logical gates required to implement the Boolean formula that represents the instance. GEC can be calculated by converting the Boolean formula into a circuit of logical gates, such as AND, OR, and NOT gates. The number of gates in the circuit corresponds to the GEC of the instance.

In our design, we optimize and use GEC encoding scheme of Feng et al (2023) to implement the S-Box. Our encoding scheme as follows in Equation 1:

$$
\begin{aligned}
&\forall i \in \{0,1,\ldots,K-1\}: \\
&\qquad\qquad T_i = F_{if}(BB_i[0], \sim(Q_{4i}\cdot Q_{4i+1})\cdot \sim Q_{4i+2}\cdot Q_{4i+3}) \\
&\qquad\qquad\quad + F_{if}(BB_i[1], Q_{4i+2}\cdot(Q_{4i}+Q_{4i+1})) \\
&\qquad\qquad\quad + F_{if}(BB_i[2], Q_{4i}\cdot Q_{4i+1}\cdot Q_{4i+2}) \\
&\qquad\qquad\quad + F_{if}(BB_i[3], Q_{4i+2}) + F_{if}(BB_i[4], Q_{4i}) \\
&\qquad\qquad\quad + F_{if}(BB_i[5], Q_{4i}\cdot Q_{4i+1}) \\
&\qquad\qquad\quad + F_{if}(BB_i[6], Q_{4i}+Q_{4i+1}) + F_{if}(BB_i[7], \max).
\end{aligned}
\tag{1}
$$

where K is numbers of the logical gates, $Q_{4i}-Q_{4i+3}$ is the input of the i^{th} logical gate, T_i is the output of the i^{th} logical gate, and F_{if} is a function that returns the value of the second argument if the first argument is true and returns the value of zero otherwise. The value of max is all one's in the binary expression, which is represented logically as an inverse. BB_i represents the type of the i^{th} logical gate, which is a 8-bit binary number. The different types of logical gate used in this encoding scheme are listed in Table 2.

The optimized architecture of S-Box is shown in Equation 2, where $X_0 - X_3$ is the input of the S-Box and $Y_0 - Y_3$ is the output of the S-Box. The proposed architecture of S-Box is implemented by four MOAI1 gates, three MAOI1 gates, and one AND3 gate. This module of the proposed S-Box reduced the area by 28.9% with Bao et al (2019) (based on gate equivalent estimation on UMC 180nm library).

$$
\begin{aligned}
T_0 &= \mathrm{MAOI1}(X_0, X_1, X_0, X_1) \\
T_1 &= \mathrm{AND3}(X_3, X_2, X_3)
\end{aligned}
$$

Table 2 Encoding of different types of logical gate

logical expression	BB_i[0:7]	gate type
$Q_0 \oplus Q_1$	0 0 0 0 0 0 1 0	XOR
$\sim (Q_0 \oplus Q_1)$	0 0 0 0 0 0 1 1	XNOR
$Q_0 \wedge Q_1$	0 0 0 0 0 1 0 0	AND
$\sim (Q_0 \wedge Q_1)$	0 0 0 0 0 1 0 1	NAND
$Q_0 \vee Q_1$	0 0 0 0 0 1 1 0	OR
$\sim (Q_0 \vee Q_1)$	0 0 0 0 0 1 1 1	NOR
$\sim Q_0$	0 0 0 0 1 0 0 1	NOT
$\sim Q_1$	0 0 0 0 1 0 1 1	NOT
$\sim Q_2$	0 0 0 1 0 0 0 1	NOT
$Q_0 \oplus Q_1 \oplus Q_2$	0 0 0 1 0 0 1 0	XOR3
$\sim (Q_0 \oplus Q_1 \oplus Q_2)$	0 0 0 1 0 0 1 1	XNOR3
$Q_0 \wedge Q_1 \wedge Q_2$	0 0 1 0 0 0 0 0	AND3
$\sim (Q_0 \wedge Q_1 \wedge Q_2)$	0 0 1 0 0 0 0 1	NAND3
$Q_0 \vee Q_1 \vee Q_2$	0 1 1 1 0 1 1 0	OR3
$\sim (Q_0 \vee Q_1 \vee Q_2)$	0 1 1 1 0 1 1 1	NOR3
$\sim ((Q_0 \wedge Q_1) \vee (\sim (Q_2 \vee Q_3)))$	1 0 1 1 0 0 0 0	MAOI1
$\sim (\sim (Q_0 \wedge Q_1) \wedge ((Q_2 \vee Q_3)))$	1 0 1 1 0 0 0 1	MOAI1

$$\begin{aligned}
&T_2 = \mathrm{MAOI1}(X_1, X_2, X_0, X_3)\\
&T_3 = \mathrm{MOAI1}(X_1, X_0, X_2, X_2)\\
&T_4 = \mathrm{MOAI1}(X_3, T_0, T_3, T_3)\\
&T_5 = \mathrm{MOAI1}(T_3, T_0, X_0, T_1)\\
&T_6 = \mathrm{MAOI1}(X_0, T_0, X_3, T_0)\\
&T_7 = \mathrm{MOAI1}(X_0, T_1, T_2, T_2)\\
&Y_0 = T_5 \quad Y_1 = T_7 \quad Y_2 = T_6 \quad Y_3 = T_4
\end{aligned} \tag{2}$$

3.1.2 Mix-Columns Optimization

The Mix-Columns component is a linear transformation of the input column. The input column is multiplied by a constant matrix M to produce the output column. M is a involutory matrix, which means $M^2 = E$, where E is the identity matrix. It is easy to decrypt the ciphertext by multiplying the ciphertext with M again. The Mix-columns component is shown in Equation 3. where $I_{0,j}$, $I_{1,j}$, $I_{2,j}$, and $I_{3,j}$ are the input column, $I'_{0,j}$, $I'_{1,j}$, $I'_{2,j}$, and $I'_{3,j}$ are the output column, and j is the column index, $j \in \{0, \ldots, 3\}$.

$$\begin{bmatrix} I'_{0,j} \\ I'_{1,j} \\ I'_{2,j} \\ I'_{3,j} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_{0,j} \\ I_{1,j} \\ I_{2,j} \\ I_{3,j} \end{bmatrix} \tag{3}$$

In order to reduce the area of this component, we use the serial architecture of Mix-Columns, as shown in Figure 3. The serial architecture of Mix-Columns requires four 4-bit register, two multiplexes and three XOR gates. The operation of Mix-Columns involves three distinct stages: freeze, shift, and add. During the freeze stage, we ensure the register values remain unchanged by setting both CM_0 and CM_1 to 0. In the shift stage, we induce a shift in the register values from RM_0 to RM_4 by setting both CM_0 and CM_1 to 1. Finally, in the add stage, we execute an addition operation on the column values according to Equation 3. This is achieved by setting CM_0 and CM_1 to 0 and 1, respectively.

The timing diagram for the serial architecture of Mix-Columns is depicted in Figure 4. It requires five clock cycles to compute the next columns from the previous ones, and an additional four clock cycles to transfer data from the internal register of Mix-Columns to the State-Register. Therefore, a complete state round requires a total of 36 clock cycles.

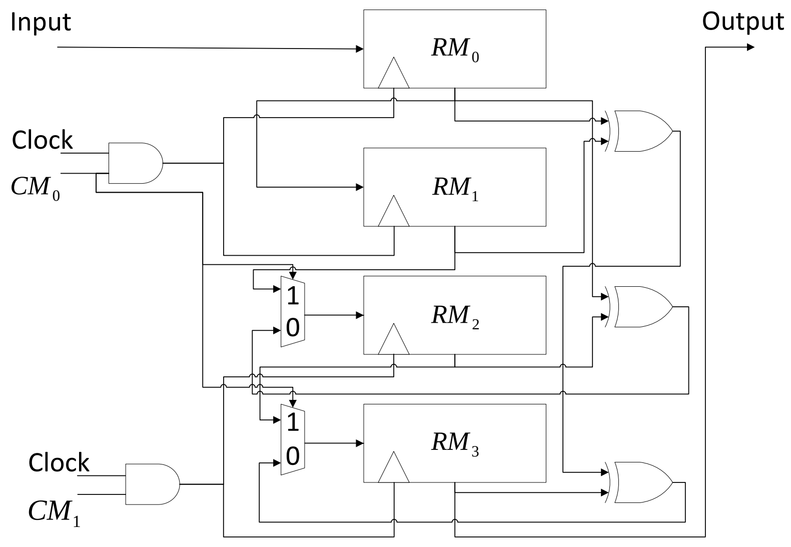

Fig. 3 Serial Architecture of Mix-Columns with clock gating

3.1.3 Control Units

The finite-state machine (FSM) of serial architecture is shown in Figure 5. The initial key and plaintext are stored in Key-Register and State-Register at the same time. After the Store, the key is expanded in Key Schedule. In Mix Columns, one column of State-Register stores in the Mix Columns registers that take four clock cycles for execution Mix-Columns over one column. In Add Key, the stored data in Mix Columns's registers are sent back to State-Register followed by XORing with keys and through the S-box

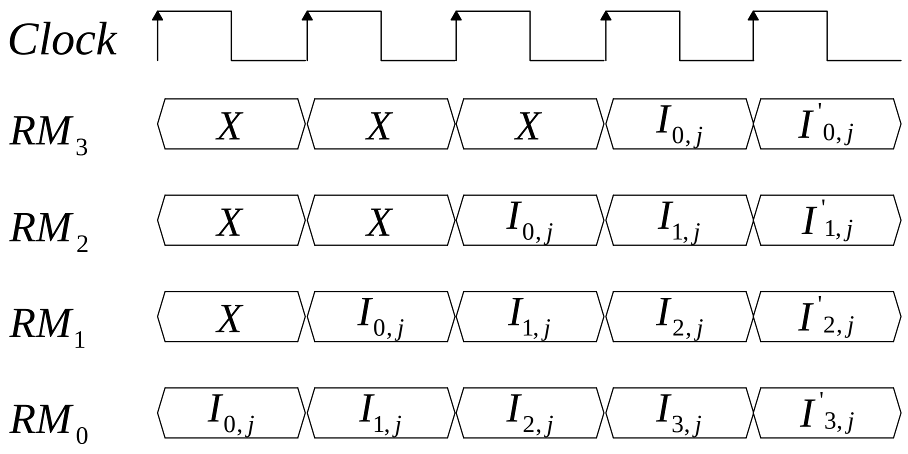

Fig. 4 Timing Diagram for the Serial Architecture of Mix-Columns

component in another four clock cycles. Permute executes in one clock cycle inside the State-Register.

The dynamic power consumption of the encryption is reduced by employing clock gating, a technique discussed in Shahbazi and Ko (2020). The clock gating is separately applied on State Register, Key Register and Mix Columns. For instance, the most power consumption is saved during the Key Schedule phase; the clock of State Register and Mix Columns is disabled to save power because these two blocks are not used in the Key Schedule phase. The timing diagram of the proposed design with the clock gating technique is shown in Figure 6.

3.2 Loop Unrolled Architecture(A2)

4 Experimental Evaluation

In ASIC implementations, the gate equivalent (GE) is often used to evaluate the area consumption of a design. A single GE is equivalent to a two-input NAND gate. To calculate the area in GE, we divide the total area (in μm^2) by the area of a two-input NAND gate (also in μm^2). However, the number of GEs can vary based on the specific technology used, as discussed in McKay et al (2016). For instance, the number of GEs for the same design will differ between UMC 180nm technology and TSMC 180nm technology. Therefore, GE is not suitable for comparing the area consumption of different designs on different technologies. In order to ensure a fair comparison, we evaluate the area consumption of the proposed designs using FPGA implementations, a method also utilized in Mohajerani et al (2020).

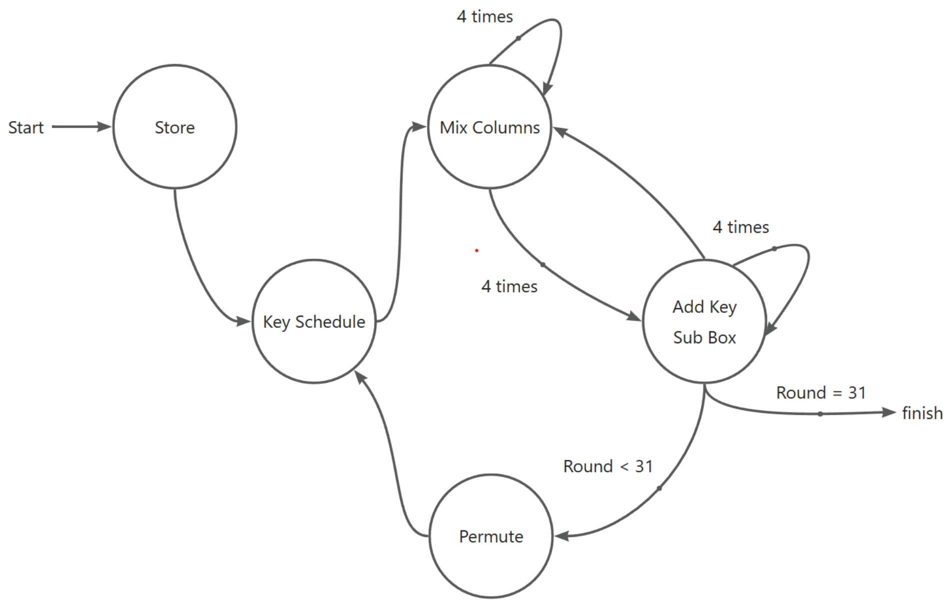

Fig. 5 Finite-state machine for Serial Architecture

4.1 Platform

We implemented the proposed architectures on a Xilinx FPGA board using the ISE Design Suite 14.7 and Vivado v2023.2. We analyzed four different FPGA platforms: Spartan-3 (xc3s200-5ft256), Spartan-6 (xc6slx16-3csg324), Virtex-4 (xc4vlx25-12ff668), and Virtex-5 (xc5vlx50t-3ff1136) using the ISE Design Suite 14.7, a methodology also employed by Lara-Nino et al (2017). For benchmarking on the latest FPGA, we used the Artix-7 (xc7a100tcsg324-1) platform with Vivado v2023.2, as done by Mohajerani et al (2020).

4.2 Area

We use the Area metric to evaluate the area consumption of the proposed designs, which includes components like flip-flops, LUTs, and slices. Given the different types of LUTs, such as LUT-4 and LUT-6, we use Spartan-3 and Virtex-4 for LUT-4, Spartan-6 and Virtex-5 for LUT-6, and Artix-7 for LUT-6. To ensure a fair comparison, we decided not to use the FPGA's embedded memory blocks by disabling the corresponding flags in the VHDL, as suggested in Xilinx (2022).

4.3 Throughput

We evaluate the performance of our proposed designs using the Throughput metric. This metric is assessed using three parameters: the maximum throughput rate, the throughput rate at 200MHz (following the method used by Ghosh et al (2017)), and the

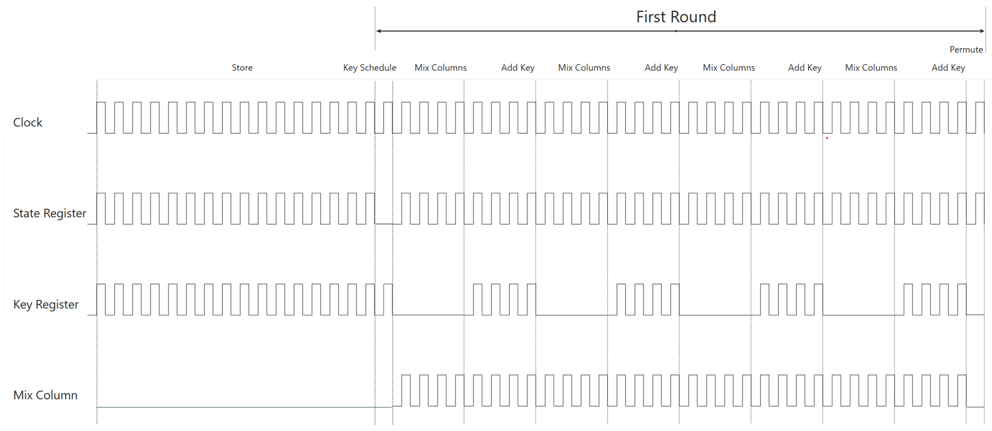

Fig. 6 Timing diagram for Serial Architecture

throughput rate per slice. The maximum throughput rate, calculated using Equation 4, represents the highest rate our designs can achieve. The throughput rate at 200MHz, calculated using Equation 5, shows the rate achievable when the clock frequency is set to 200MHz. Finally, the throughput rate per slice, calculated by dividing the throughput rate by the number of slices (Equation 6), provides a measure of efficiency. In these calculations, the Plaintext Size is 64-bit, Latency refers to the number of clock cycles required to encrypt a single block, and Slices refers to the number of slices used by the design.

$$MaximumThroughput(Thr) = \frac{MaximumFrequency \times PlaintextSize}{Latency} \tag{4}$$

$$Throughput_{@200MHz}(Thr^{*}) = \frac{200MHz \times PlaintextSize}{Latency} \tag{5}$$

$$ThroughputPerSlice = \frac{Throughput_{@200MHz}(Thr^{*})}{Slices} \tag{6}$$

5 Results

6 Conclusion

References

Bao Z, Guo J, Ling S, et al (2019) Peigen–a platform for evaluation, implementation, and generation of s-boxes. IACR Transactions on Symmetric Cryptology pp 330–394

Beierle C, Leander G, Moradi A, et al (2019) Craft: lightweight tweakable block cipher with efficient protection against dfa attacks. IACR Transactions on Symmetric Cryptology 2019(1):5–45

Feng J, Wei Y, Zhang F, et al (2023) Novel optimized implementations of lightweight cryptographic s-boxes via sat solvers. IEEE Transactions on Circuits and Systems I: Regular Papers

Ghosh S, Misoczki R, Zhao L, et al (2017) Lightweight block cipher circuits for automotive and iot sensor devices. In: Proceedings of the Hardware and Architectural Support for Security and Privacy. p 1–7

Lara-Nino CA, Diaz-Perez A, Morales-Sandoval M (2017) Lightweight hardware architectures for the present cipher in fpga. IEEE Transactions on Circuits and Systems I: Regular Papers 64(9):2544–2555

McKay K, Bassham L, Sönmez Turan M, et al (2016) Report on lightweight cryptography. Tech. rep., National Institute of Standards and Technology

Mohajerani K, Haeussler R, Nagpal R, et al (2020) Fpga benchmarking of round 2 candidates in the nist lightweight cryptography standardization process: Methodology, metrics, tools, and results. Cryptology ePrint Archive

Shahbazi K, Ko SB (2020) Area-efficient nano-aes implementation for internet-of-things devices. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 29(1):136–148

Sharma S, Kaushik B (2019) A survey on internet of vehicles: Applications, security issues & solutions. Vehicular Communications 20:100182

Xilinx A (2022) Ultrafast design methodology guide for xilinx fpgas and socs: Super logic region