A Hardware-Efficient Implementation of CLOC for On-chip Authenticated Encryption

Conference Paper · July 2018

DOI: 10.1109/ISVLSI.2018.00064

4 authors, including:

Mahmoud Aly Elmohr
University of Waterloo
6 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Mustafa Khairallah
Nanyang Technological University
17 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Anupam Chattopadhyay
Nanyang Technological University
202 PUBLICATIONS **1,164** CITATIONS
SEE PROFILE

A Hardware-efficient Implementation of CLOC for On-Chip Authenticated Encryption

Mahmoud A. Elmohr
Department of Electrical and Computer Engineering,
University of Waterloo, Ontario, Canada
Email: mahmoud.a.elmohr@ieee.org

Sachin Kumar[1], Mustafa Khairallah[2],
and Anupam Chattopadhyay[1]
School of Computer Science and Engineering[1],
School of Physical and Mathematical Sciences[2]
Nanyang Technology University, Singapore[1,2]
Email: anupam@ntu.edu.sg

***Abstract*—With the huge number of candidates for the CAESAR competition for authenticated encryption, the task of designing efficient implementations for these candidates becomes a big challenge. The main goal of this competition is to find smaller, faster, energy-efficient and secure authenticated encryption schemes. In this paper, an area efficient hardware implementation of CLOC, one of the 15 candidates for the third round of CAESAR competition is presented. CLOC represents a new mode of using AES, in order to provide both encryption/decryption and MAC functionalities. Since the hardware design of AES is a well studied problem, the challenge is to accommodate the mode functionality with small area, high performance and low power overhead. The proposed hardware implementation for the CLOC is developed by sharing the AES core by applying a pipeline technique. By using commercial synthesis flows and 65 nm ASIC technology, it shows, for low power applications, that proposed hardware architecture of CLOC reduces the area by 42.85% and consumes 37.8% less power when compared with the existing high throughput implementation of CLOC. In addition, area-efficiency of the proposed design is also improved by 17.6% and the proposed design consumes only 2.6 μW.**

***Keywords*—AES, Authenticated Encryption, ASIC implementation. Technology mapping, logic optimization.**

I. Introduction

The security goals of any communication system can be defined with mainly two goals: confidentiality and integrity. While encryption provides confidentiality, this may not be sufficient to the systems in which data is streamed in insecure channels. Such systems also need data integrity and authentication in order to ensure that original information has not been modified by unauthorized or unknown parties. Thus, Authenticated Encryption (AE) has emerged as a scheme that provides both confidentiality and integrity of the data simultaneously [1].

The straightforward way to design an AE scheme is by using a secure encryption scheme and a secure Message Authentication Code (MAC) with two separate keys and use both of them as a generic composition. This composition could have three different approaches: MAC-then-encrypt, Encrypt-and-MAC, and Encrypt-then-MAC, and only the former one is guaranteed to be secure under condition that both the encryption scheme and the MAC are secure [2]. The main disadvantages of such approacesh are: being slow, requiring two different keys and even may have error implementations if not handled carefully.

Thus, dedicated AE schemes have been introduced to overcome the generic composition method. Most of these schemes were developed as modes for block ciphers such as the Counter with CBC-MAC (CCM) mode [3], the Galois Counter Mode (GCM) [4] and the Offset Codebook Mode (OCB) [5].

In order to fill the gap and standardize the authenticated encryption schemes; NIST and Daniel Bernstein co-founded the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) in 2013 [6], opening the call for submissions that should provide advantages over AES-GCM and suitable for widespread adoption. The call was responded by 57 submissions in total passing by rounds of evaluation, resulting in only 15 candidates so far in the third round. In order to facilitate hardware implementations for the candidates; a generic hardware Application Programming Interface (API) was introduced in [7] to meet the requirements of all algorithms submitted to the CAESAR competition, as well as many earlier developed authenticated ciphers, such as AES-GCM and AES-CCM. The API consists of: a preprocessor that handles the input in a standard form and passes it to the cipher core, a post-processor that handles the output to be in the standard form, and a FIFO. Apparently, a generic API would have more area overhead than a specific one, moreover it could limit the performance of some candidate algorithms, and may favor one algorithm over another, thus should not be used solely to decide the winning candidate. However, it is essential in such competitions to provide a common base for all candidates and a common testing environment.

In this paper, a new hardware architecture for CLOC [8], one of the CAESAR third round candidates, is presented. The new architecture uses a round-based AES implementation as a primitive. However, only one encryption core is used, as opposed to the two-core architecture based CLOC in [9]. With many applications that do not require the high speed provided by the current ASIC fabrication technologies and AES implementations, such as RFID tags, smart cards and WSNs, the cost of area and power becomes more vital. Consequently, the proposed design has been implemented and compared to the existing CLOC architecture in [9]. The design has been synthesized for both high speed and low power

targets. Table III and Table IV show the summary of results for both applications with the high speed requirements and those with low power requirements respectively.

The remaining of the paper is organized as follows: In Section II a description of the CLOC scheme and its functions is presented as well as an illustration of the existing hardware implementation of CLOC in [9]. In Section III we present the proposed implementation, illustrating its architecture and differentiating between it and the existing design. Section IV represents the results of both designs using Synopsys Design Compiler tool with TSMC 65nm technology. And finally we conclude our work in Section V.

II. CLOC Authenticated Encryption Scheme

Notation: M: Unencrypted Message, C: Encrypted Message, T, T*: Tag, N: Nonce, V: Temporary Tag, A: Associated Data, $\perp$: Error.

CLOC [8] is a block cipher mode of operation for authenticated encryption with associated data (AEAD). The design of CLOC aims at being provably secure and optimizing the implementation overhead beyond the block cipher, the precomputation complexity, and the memory requirement. The main advantage for CLOC is how it handles short input data efficiently, and is suitable for use with embedded processors. CLOC has two variations: one based on AES and the other is based on TWINE [10] ciphers with the same components remaining. In this context, we adopt the AES version of CLOC.

The two main algorithms CLOC-E for encryption and CLOC-D for decryption use four subroutines, HASH, PRF, ENC, and DEC, and could be described as follows:

- CLOC-E(N, A, M):
  1) V $\leftarrow$ HASH(N,A)
  2) C $\leftarrow$ ENC(V,M)
  3) T $\leftarrow$ PRF(V,C)
  4) return (C, T)
- CLOC-D(N, A, C, T)
  1) V $\leftarrow$ HASH(N,A)
  2) T* $\leftarrow$ PRF(V,C)
  3) If T != T* then return $\perp$
  4) M $\leftarrow$ DEC(V,C)
  5) return M

Beside AES as the main function in CLOC, there are other functions to be defined as follows:

- Fix0 / Fix1: assigns the first bit of the data block to be 0 or 1 respectively.
- H / F1 / F2 / G1 / G2 : are simple linear functions of XORs and permutations.
- OZP: pads blocks less than 128 bit with 1 and a sequence of 0s. However this function is handled by the CAESAR API, so it is not implemented in the proposed design.

There are two existing hardware implementations of CLOC, the design in [11] is a low area implementation with an 8-bits datapath using an 8-bits serialized AES core, the main target of that design was achieving the lowest possible area, neglecting throughput. The other implementation in [9] is a high-speed implementation using two regular 128-bit AES cores to achieve the highest possible throughput, however, at the expense of large area.

Since the implementation in [9] was intended for maximum possible throughput, it used two AES cores exploiting the parallelism of ENC and PRF subroutines, which in turn increased the area massively, and reduced the area efficiency due to the fact that both functions are sequential to the HASH function leaving one AES core unused for approximately 50% of the running time assuming long plaintext and long associative data as well.

III. Proposed Architecture of CLOC

The idea of making the proposed design efficient is to use only one instance of each functional block and reuse them in a multi-cycle approach. In order to achieve that, and to have a single configurable datapath accomodating the four subroutines' datapaths; a control unit based on a Finite State Machine was developed to alter the datapath using eight multiplexers and two registers as shown in Figure 1.

The internal functional blocks shown in Figure 1 could be detailed as follows:

- The blocks Fix0, Fix1, F1, F2, G1, G2 and H are direct implementations for the corresponding CLOC primitive functions described in Section II and detailed in [8].
- The AES core used is an encryption only round-based AES core described in [9].
- The block "Conc" is used to concatenate the nonce with a constant that depends on the parameters of CLOC as described in [8]. However only the two recommended sets of parameters are implemented in both designs.
- "Expand" block operates on bdi_valid_bytes signal provided by the API and expands it to its bits equivalent. The API delivers the bdi_valid_bytes signal representing the valid bytes of the input to indicate if the current block of input is not occupying the whole block.
- AND gates are used to mask signals with $0s$ reducing their size to the same size of the input in case of being less than 128 bits. That is mainly dependent on the expanded bdi_valid_bytes signal.
- XOR gate is used by the four subroutines.
- The block "MSB" is used to extract only the 96 most significant bits of the output as the size of the tag T.

A. HASH Datapath

The HASH subroutine's datapath starts with the associated data A within the API's bdi signal passing through Mux-A, applying Fix0 function to it, and encrypts the result with the help of an AES core after passing through Mux-B. The most significant bit of the associated data decides whether the result could be passed to H function or passed as it is through Mux-C. The result is then XORed with the next block of the associated data which passes this time through Mux-D, and the encryption process of the result is repeated again after passing Mux-E, Mux-F and Mux-B. This process keeps going

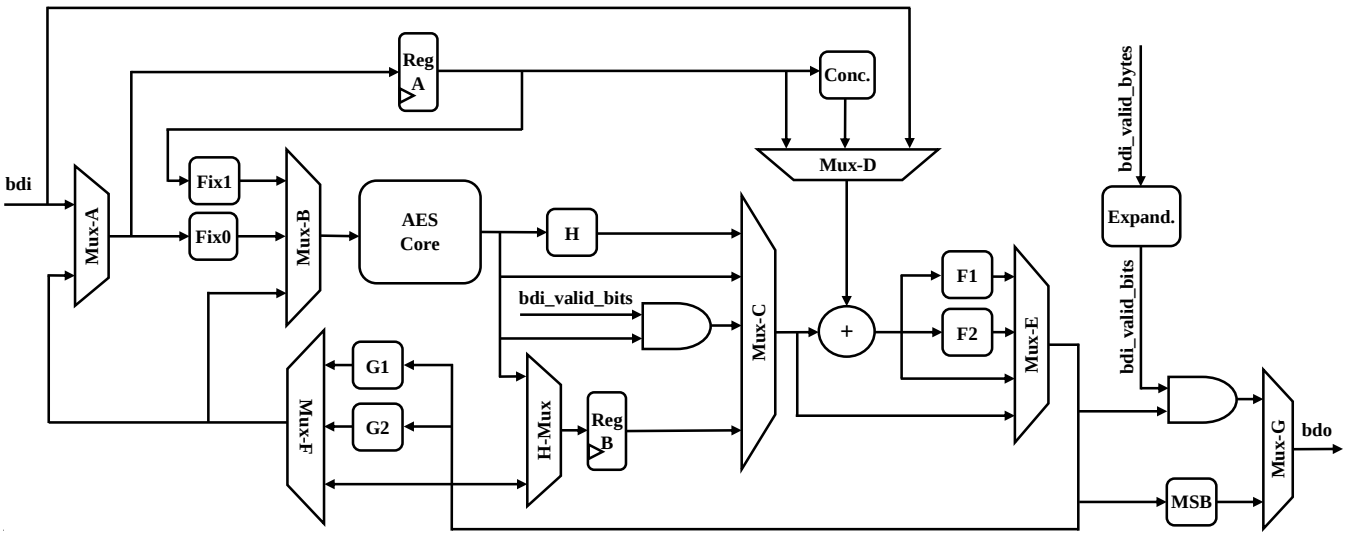

Figure 1: Optimized CLOC implementation architecture.

on until the last block of the associated data, at which the encrypted data is XORed with the Nonce N stored in Reg-A concatenated with a constant depending on the parameters of CLOC. The Nonce is stored in Reg-A due to the fact that the API delivers the Nonce within the bdi signal before the associated data, thus needs to be stored. At the end, either F1 or F2 functions applies on the result and passes through Mux-E depending on the size of the last block of the associated data resulting in the intermediate tag V which passes through Mux-H to be stored in Reg-B for further usage by the remaining subroutines.

B. PRF Datapath

If there is no plaintext M to encrypt nor a ciphertext C to decrypt, the intermediate tag V is extracted from Reg-B, passed through Mux-C and Mux-E as it is and G1 function is applied to it, and passed by Mux-F and Mux-B before being encrypted by AES producing the tag T. Otherwise the intermediate tag V passes with the same datapath except for applying G2 function instead of G1, and the result of the encryption is passed through Mux-C to be XORed with one block of the produced ciphertext C stored in Reg-A by ENC subroutine in case of encryption or the received ciphertext C passed within the API's bdi signal in case of decryption, both passing by Mux-D. The result of the XOR passes through Mux-E, Mux-F and Mux-B to be encrypted again, then stored in Reg-B after passing through Mux-H. The result is stored in Reg-B due to the fact that PRF subroutine uses the ciphertext blocks produced by the ENC subroutine or the ciphertext blocks used also by the DEC subroutine, thus stores the current state in Reg-B, hands over the datapath to either ENC or DEC subroutines waiting the next ciphertext block and the same process goes until the last block of the ciphertext, at which the XORing result is fed to either F1 or F2 functions depending on the size of the last block of the ciphertext. Then finally the result passes through Mux-E, Mux-F and Mux-B to be encrypted for the last time to produce the tag T. To output the tag within the API's bdo signal, the result passes through Mux-E and its most 96 significant bits are extracted through MSB block and finally output through Mux-G.

C. ENC Datapath

The ENC datapath encrypts the intermediate tag V produced by the HASH subroutine previously and already stored in Reg-B after passing through Mux-C, Mux-E, Mux-F and Mux-B. The encryption result is passed through Mux-C and XORed with the first block of the plaintext M within the API's bdi signal passed through Mux-D, producing the first block of the ciphertext C. The produced cipher text is stored in Reg-A through Mux-E, Mux-F and Mux-A, then the datapath is handed over to the PRF subroutine. After the PRF hands over the datapath again, the produced ciphertext block is extracted from Reg-A and fed to Fix1 function, and passes through Mux-B to be encrypted and XORed with another block of the plaintext producing another block of the ciphertext. The process repeats until the last block of the plaintext with only one difference which is that the encrypted result this time is ANDed with the bdi_valid_bits to have an output with the same length of the last plaintext block to be XORed with. Each produced ciphertext is passed through Mux-E and ANDed again with the bdi_valid_bits to be output finally through Mux-G within the API's bdo signal. The bdi_valid_bits is an internal signal produced by the Expand block to convert the API's bdi_valid_bytes to its equivalent bits representing the valid bits of the last block of the input data.

D. DEC Datapath

The DEC subroutine naturally reuses the ENC datapath but in reverse. At first the intermediate tag V is encrypted, XORed with the API's bdi signal which this time represents a block of the ciphertext C, producing the first block of the plaintext M. At the same time the delivered ciphertext is

stored in Reg-A through Mux-A and the datapath is handed over to the PRF subroutine. After the PRF hands over the datapath again, the ciphertext block is extracted from Reg-A and fed to Fix1 function and passes through Mux-B to be encrypted and XORed with another block of the ciphertext producing another block of the plaintext. Similar to the ENC subroutine, the process repeats until the last block of the plaintext which is handled the same way as in ENC subroutine. Also each plaintext is output through Mux-G passing by the same datapath as ENC subroutine.

IV. Experimental Results and Comparison

The hardware performance of the proposed hardware implementation of CLOC (also denoted as Optimized CLOC) is evaluated by comparing against the given hardware implementation of CLOC in [9]. A comparison against the design in [11] is not possible due to the unreported throughput of the design. While the design in [11] targets low area neglecting performance, our proposed design as well as the design in [9] target high performance and both are compatible with CAESAR hardware API, which facilitates a fair comparison. Thus, only a comparison against the work in [9] is presented. The Optimized CLOC is described in Verilog language and functionally verified by given test vectors with the help of modelsim tool. Both designs are synthesized and mapped to TSMC 65nm standard cell library by using Synopsys Design Compiler version J-2014.09. Upon doing synthesis, both designs were constrained for achieving maximum speed along with their area-constrained. Area is measured in μm, as well as in kilo gate equivalent (KGE) in order to make it technology independent. Table I shows synthesis results comparison of Optimized CLOC and CLOC in [9] in terms of area, critical path delay and power performance parameters. Based on the experimental results, it shows that Optimized CLOC is 29.29% smaller and achieved 21.7% less power requirements when compared with CLOC. It also requires a clock frequency that is 21.17% faster. This is because only one AES core was used in the proposed design rather than using two AES cores, which in turn reduced the area as well as power of the proposed design. While the frequency is increased due to the use of intermediate registers in the Optimized CLOC.

Table I: ASIC synthesis results comparison between our design and the design from [9]

Design	Delay (ns)	Area (μm^2)	Area (KGE)	Power (mW)
[9]	1.70	136,618.6	94.87	25.02
Optimized CLOC	1.34	96,605.3	67.09	19.59

In order to estimate throughput, both designs were examined with large plaintext and large associated data namely 100 block for each, which is equivalent to 12800 bits for each. As Table II shows, the number of cycles for our design is increased due to using only one AES core, but due to the higher frequency our design can reach; the decline in throughput is small.

The area efficiency is defined as throughput/area. These metrics play an important role in judging modern digital circuits, including cryptographic accelerators. Table III shows that the proposed design is 20.6% more area efficient and 21.7% less power consumer.

Table II: Comparison of throughput of our design and the design from [9]

Design	Frequency (MHz)	# of cycles	Throughput (bits/cycle)	Throughput (Mbps)
[9]	588.23	2252	5.68	3341.18
Optimized CLOC	746.26	3352	3.82	2850.75

Table III: Comparison between the proposed architecture of CLOC and the architecture of CLOC from [9]

Design	Area (KGE)	Area Efficiency (Mbps/KGE)	Power (mW)
[9]	94.87	35.22	25.03
Optimized CLOC	67.09	42.49	19.59
Improvement	29.29%	20.6%	21.7%

However, the high speed achieved by these implementations is not required for applications such as smart cards, passive Radio-Frequency Identification (RFID) tags and Wireless Sensor Networks (WSNs). For such applications, low power and small area are more crucial. Hence, the comparison has also been performed at 100 KHz, leading to smaller area and power for both implementations. More significantly, the area and power difference is more significant at 100 KHz, with the Optimized CLOC operating at only 2.6 μW, as shown in Table IV.

Table IV: Comparison between the proposed architecture of CLOC and the architecture of CLOC from [9] at 100 KHz

Design	Area (KGE)	Area Efficiency (Kbps/KGE)	Power (μW)
[9]	81.53	6,966.8	4.28
Optimized CLOC	46.59	8,199.1	2.60
Improvement	42.85%	17.6%	37.80%

V. Conclusion

In this paper, a high-speed and hardware-efficient implementation for CLOC authenticated encryption cipher is proposed particularly for achieving better throughput per area performance. In what follows, reduction in area was achieved by calling the functions sequentially while critical path delay is reduced by inserting the immediate registers in the optimized design. Based on ASIC synthesis results, it shows that the proposed design proved to be less in area by 29.29% and improved area efficiency (throughput per area) by 20.6% when compared with its contender.

References

[1] T. Kohno, J. Viega, and D. Whiting, "The CWC authenticated encryption (associated data) mode," *ePrint Archives*, 2003.

[2] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," *Journal of Cryptology*, vol. 21, no. 4, pp. 469–491, Oct 2008.

[3] M. J. Dworkin, "SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," Gaithersburg, MD, United States, Tech. Rep., 2007.

[4] D. A. McGrew and J. Viega, "The Security and Performance of the Galois/Counter Mode (GCM) of Operation," in *Proceedings of the 5th International Conference on Cryptology in India*, ser. INDOCRYPT'04, Chennai, India, 2004, pp. 343–355.

[5] P. Rogaway, M. Bellare, and J. Black, "OCB: A block-cipher mode of operation for efficient authenticated encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 365–403, 2003.

[6] CAESAR Competition, "CAESAR submissions," https://competitions.cr.yp.to/caesar-submissions.html, 2016.

[7] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, and K. Gaj, "Implementer's Guide to the CAESAR Hardware API," 2016.

[8] T. Iwata, K. Minematsu, J. Guo, and S. Morioka, "CLOC: authenticated encryption for short input," in *International Workshop on Fast Software Encryption*. Springer, 2014, pp. 149–167.

[9] George Mason University, "ATHENa: Automated Tools for Hardware EvaluatioN," https://cryptography.gmu.edu/athena/, 2017.

[10] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "Twine: A lightweight block cipher for multiple platforms," in *International Conference on Selected Areas in Cryptography*. Springer, 2012, pp. 339–354.

[11] S. Banik, A. Bogdanov, and K. Minematsu, "Low-area hardware implementations of CLOC, SILC and AES-OTR," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 71–74.