AEZ: Anything-but EaZy in Hardware

Ekawat Homsirikamol and Kris Gaj

Electrical and Computer Engineering Department
George Mason University,
Fairfax VA , USA
{ehomsiri, kgaj}@gmu.edu

Abstract. We provide the first hardware implementation of AEZ, a third-round candidate to the CAESAR competition for authenticated encryption. Complex, optimized for software, and impossible to implement in a single pass, AEZ poses significant obstacles for any hardware realization. Still, we find that a hardware implementation of AEZ is quite feasible. On Xilinx Virtex-6 FPGAs, our single-core design has a throughput exceeding 3.4 Gbit/s, and uses about 4600 LUTs and about 1250 CLB slices. In terms of the throughput to area ratio, this performance places it on the 12th position among 28 CAESAR candidate families benchmarked during Round 2 of the competition (assuming the key size of at least 96 bits, and the limit on the message size equal to $2^{11} - 1$ bytes). At the same time, AEZ targets a stronger notion of security against the cipher misuse than all other algorithms implemented and ranked ahead of it in the Round 2 hardware benchmarking study.

Keywords: authenticated ciphers, AEAD, CAESAR, FPGA, hardware, MRAE.

1 Introduction

Authenticated encryption (AE) has become the preferred approach, in most settings, for achieving symmetric encryption. This paper describes the first hardware implementation of AEZ [1, 2], a new AE scheme that targets an unprecedentedly strong security notion. Let us back up and provide a bit of context.

The AE goal. An AE scheme takes in a key, a nonce, associated data (AD), and a plaintext. For majority of schemes, it returns a ciphertext and a tag. For some schemes, such as AEZ, it returns just a ciphertext (which is then typically longer than the plaintext). Decryption reverses the process, using the same key, nonce, and associated data (AD), as well as the ciphertext and, optionally, the tag returned by encryption, as an input. It returns either a plaintext or an indication of invalidity. There are two aims. *Confidentiality* requires ciphertexts to be computationally indistinguishable from random bits, while *authentication* assures that no one should be able to produce new and valid ciphertexts without knowing the key.

At present, there are just two widely used AE schemes, CCM and GCM. Both are standardized by ISO and NIST, but neither is particularly modern, efficient, or versatile. To address this, the CAESAR competition for AE schemes began in 2012, attracting some 57 submissions [3].

The AEZ scheme. AEZ [1,2] is one of the more unusual CAESAR candidates. Where many submissions tried to excel in hardware efficiency, software efficiency, or both, AEZ focused on a new and unusually strong security notion. That goal, *robust* authenticated encryption (RAE), guarantees all that a conventional AE scheme does and more. First, it must work *as well as possible* even if nonces *do* repeat. That is the goal of *misuse-resistant* authenticated encryption (MRAE) [4]. But an RAE scheme goes further, achieving this as-good-as-possible behavior for any choice of ciphertext expansion (how much longer a ciphertext is than a plaintext), including none at all.

The cost of RAE. Proponents of RAE and MRAE think that schemes designed to meet these ends will be easier to use and less prone to misuse. But achieving these goals comes at a cost, starting with the fact that they can't be achieved by any one-pass scheme. (A one-pass scheme reads each input and writes each out left-to-right, employing a constant amount of memory.) To encrypt, you must make two passes over the plaintext or employ a buffer as big as the plaintext is long. This is no doubt the reason why, despite the importance of nonce-reuse security, very few CAESAR candidates tried to achieve MRAE. The only Round 2 schemes the authors are aware of are AEZ, Deoxys, HS1-SIV, and Joltik. The comparison among the four of the above schemes in terms of security is beyond the scope of this paper. However, for fairness, it should be mentioned that the security provided by AEZ has a birthday bound of 2^{64} blocks, limited by the state size of the algorithm, which is among the lowest among the Round 2 CAESAR candidates. That means that there are easy distinguishing and forging attacks by the time the adversary queries AEZ with about 2^{64} blocks of message, AD, or nonce. However, the users are protected against these attacks by staying below 2^{48} bytes of data (about 280 TB), by that time, they need to rekey. Increasing this birthday bound was clearly and explicitly a non-goal for the designers of AEZ [1, p. 13]. On the hardware benchmarking side, no VHDL/Verilog implementations of the nonce misuse resistant variants of Deoxys and Joltik, compliant with the CAESAR Hardware API, have been reported to date.

Achieving RAE (which, again, goes beyond MRAE) is an especially tall order, encompassing the ability to encipher arbitrary-length strings. AEZ aims to achieve this with about the efficiency of AES-CTR. The result is the most complicated symmetric encryption scheme we know. AEZ's description spans 1.5 pages of dense pseudocode (excluding the definition of the AES round function and Blake2b) [1].

After explaining that AEZ's name was meant to suggest both *authenticated encryption* (AE) and *easy* (EZ), its authors warn that the alleged easiness refers only to ease of use. "Writing software for AEZ is *not* easy," they write, "while doing a hardware design for AEZ is far worse" [1, pp. 2]. After some interaction

with us, the AEZ designers added in that "From the hardware designer's perspective, AEZ's name might seem ironic, the name better suggesting *anti-easy*, the *antithesis of easy*, or *anything-but easy*" [1, pp. 2–3]. We note that a prior attempt at implementing AEZ by a Master-level student did not succeed, the designer concluding that AEZ was "hardly suitable for hardware" [5, p. 30].

Contributions. In this paper we overcome these difficulties and develop a fully-functional hardware realization of AEZ. Our realization conforms to the CAESAR Hardware API used in the CAESAR competition [6]. We implement everything in the AEZ spec except for the parts that handle arbitrary key lengths, arbitrary ciphertext expansion, and vector-valued AD. Please note that we are not aware of any other Round 2 CAESAR candidate offering arbitrary ciphertext expansion and vector-valued AD.

Our implementation achieves roughly the same throughput as the comparable implementation of AES-GCM, and takes almost the same area as the comparable implementation of OCB. In terms of the throughput to area ratio, our design ranks no. 12 out of 28 benchmarked Round 2 families (assuming the key size greater or equal to 96, and the limit on the message size equal to $2^{11} - 1$ bytes). It trails AES-GCM, only because of the larger area. It outperforms many other AES-based CAESAR candidates, such as CLOC, ELmD, OCB, AES-OTR, SILC, POET, AES-COPA and SHELL.

2 AEZ Overview

AEZ is built on a *generalized block cipher*, Encipher. This object is like a conventional block cipher except that (1) you can feed it any number of bytes (which will get enciphered into the same number of bytes), and (2) you can also provide a *tweak*, which, in this case, is a vector of strings. The tweak is a non-secret value that individualizes the permutation associated to the key.

To create an RAE scheme from its generalized block cipher, AEZ does the following: it takes the input M and it appends to it τ zero bits, where τ is the ciphertext expansion the user wants. Our realization assumes $\tau = 128$. Then you encipher. The result is the final ciphertext. To decrypt with AEZ, reverse the process, deciphering the ciphertext to get an augmented message. If the last τ bits of this augmented message is anything but the all-zero string, the ciphertext is invalid. Otherwise, the rest is the plaintext. For both enciphering and deciphering one uses a tweak that consists of three components (assuming a string-valued AD): an encoding of the ciphertext expansion τ, the nonce N, and the AD A.

Fig. 1 describes the generalized block cipher Encipher. The message, M, is already assumed to be extended with τ zeros that we wish to encipher. Initially, attend only to the top-left and top-right portions of the diagram, and assume that $M = M_1 M'_1 \cdots M_m M'_m M_{\mathsf{u}} M_{\mathsf{v}} M_{\mathsf{x}} M_{\mathsf{y}}$ has a multiple of 32 bytes (but at least 64 bytes). Each subscripted variable is 16 bytes.

The boxes labeled by pairs (j, i) in the diagram show the application of a *tweakable block cipher* (TBC). The key is always K, the key we wish to encipher

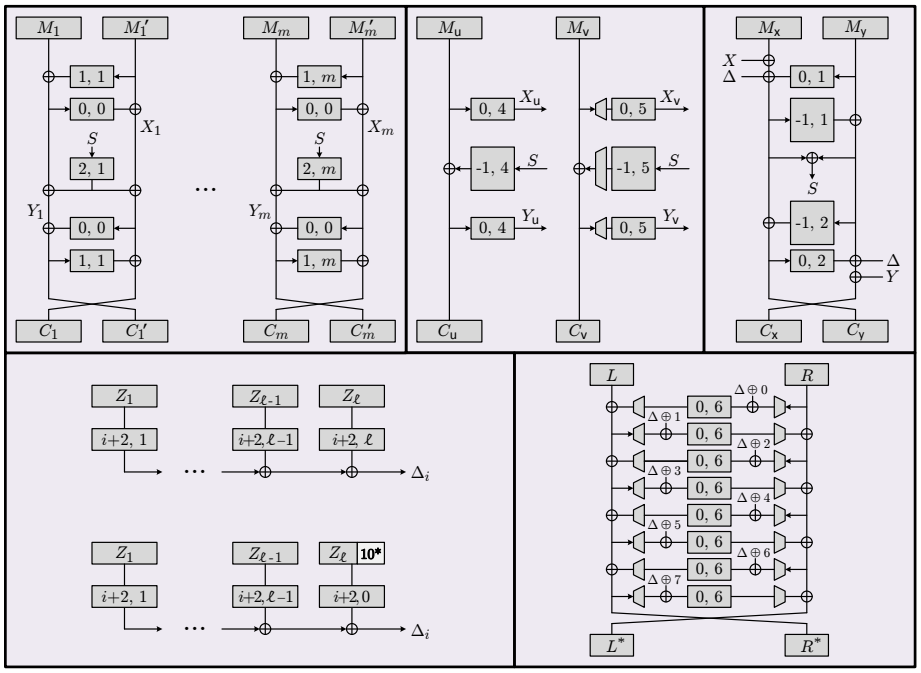


Fig. 1: Illustration of AEZ enciphering, adapted (with permission) from Figure 5 of the AEZ spec [1]. Rectangles with pairs of numbers are tweakable block ciphers, the pair being that tweak (the key, always K, is not shown). **Top row:** enciphering a message M of (32 or more bytes) with AEZ-core. The i-block (top left) is used for the bulk of the message, but the xy-block (top right) comprises the last 32 bytes, while the uv-block (top middle) comprises the prior 0–31 bytes. (The picture shows a uv-block of 17–31 bytes.) The string X is computed via $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus X_\mathsf{u} \oplus X_\mathsf{v}$; if X_u or X_v is undefined then this term is omitted in computing X. The string Y is computed analogously. **Bottom left:** AEZ-hash computes $\Delta = \bigoplus \Delta_i$ from a vector-valued tweak encoding the ciphertext expansion τ, the nonce N, and the AD A. Its i-th component $Z_1 \cdots Z_\ell$ is hashed as shown. **Bottom right:** AEZ-hash, when operating on a string $M = L\ R$ of 16–31 bytes. More rounds are used if M has 1–15 bytes.

under, while the pair in the box is the tweak. Thus the box labeled $(1, 1)$ maps K and the 16-byte M_1' to an unnamed 16-byte value that is xor'ed with M_1 to get another 16-byte value, which is fed into the block cipher labeled $(0, 0)$, and so on. The figure's caption should make the notation clear.

The middle-top portion of the diagram hints at what happens when the plaintext is not a multiple of 32 bytes. For all *other* AE schemes we know, messages that are not multiples of the block size give rise to a final fragment

that includes all the leftover bytes of the message. For AEZ, any leftover bytes form the penultimate chunk instead. This is useful because it ensures that, when 16 or fewer zero bytes are appended to the end of a message, they will always land in a specific block, rather than spanning two. If we want to shortcut the rejection of invalid messages, this feature has a potential to simplify the implementation.

To encipher messages with fewer than 32 bytes, one bypasses the top row of Fig. 1 completely and runs the Feistel network shown at the bottom right instead, splitting the message M into equal-length halves. This algorithm is called AEZ-tiny; the top row is AEZ-core. The two-algorithm approach, one for short messages and another for longer ones, mirrors a large body of cryptographic work in which techniques for "format preserving encryption" (FPE) do not resemble the modes of operation for a "wideblock block cipher."

The TBC used in AEZ is based on AES4 and AES10, which are 4 or 10 round versions of AES. The first is depicted in Fig. 1 as a (j, i)-labeled rectangle for $j \geq 0$; the second is a (j, i)-labeled square for $j = -1$. Neither uses the AES key schedule. In the end, the bulk of the work for enciphering 32 bytes from a long message—one "column" from the top-left of the figure—is the 20 AES rounds associated to the five AES4-based TBC calls. So 10 AES rounds per 16 bytes, the same overhead as AES itself.

One further detail concerns the processing of the empty message $M = \varepsilon$, which AEZ gives special attention to since this is a fairly natural way to realize a message authentication code, the string that is authenticated being the AD.

Among the pleasant characteristics of AEZ is that only the forward direction of the TBC is ever needed, and enciphering and deciphering are virtually identical. Our hardware design benefits from these choices.

3 Hardware Implementation Challenges

AEZ is the most challenging CAESAR candidate to implement in hardware. The reasons for this are summarized below.

Three algorithms in one. AEZ defines three substantially different algorithms: (a) AEZ-prf to process empty messages, (b) AEZ-tiny to process messages of the size smaller than 32 - authenticator length (in bytes) (= 16 bytes for recommended values of parameters), and (c) AEZ-core to process all remaining message sizes. Although these algorithms share the same major building block, TBC, they have a very different internal structure, and implementation requirements. A hardware designer is faced with the decision to either implement these algorithms separately (without resource sharing), which may substantially increase the circuit area, but simplifies scheduling and control, or base the implementation on a single instance of TBC, which has the opposite implications. In our design, we chose the latter approach in order to address the already quite substantial area requirements of AEZ.

Two-passes. As shown in Fig. 1, AEZ-core requires two passes. The first pass is used to calculate S, which is a function of the nonce (public message number), all blocks of the associated data, all blocks of the message, the authenticator length τ, and the key. In the second pass, S is used in calculations involving all message blocks. A hardware designer is faced with the decision to either repeat approximately 40% of computations involving all message blocks, already done in the first pass, or to store intermediate results of the size of the entire message in internal memory. In order to avoid a substantial performance penalty, and keeping in mind that (a) packet sizes in modern communication protocols are relatively small (typically at or below 1500 bytes), and (b) modern FPGAs contains large blocks of memory, which often remain unused by the main cryptographic and data processing tasks, we have decided to follow the latter approach.

Input re-blocking. In a typical hardware implementation of an authenticated cipher, input blocks are provided to the cipher module sequentially, one by one. Only one block is processed at a time. All blocks, except the last one have the same length. The last block is often just padded, and then processed similarly (although rarely identically) as other blocks. After each message block is processed, the corresponding ciphertext block leaves the cipher module. As shown in Fig. 1,

- in AEZ-tiny, the blocks L and R have variable length depending on the size of the message, $|M|$,
- in AEZ-core, the blocks M_u and M_v have variable length depending on the size of the message, $|M|$. On top of that (a) neither of these blocks is the last block of the message, and (b) for certain message lengths $|M_u|$=0. As a result, the implementation of AEZ must internally create and process blocks of data of unconventional sizes, which amounts to input "re-blocking". In hardware, such operation requires variable shifts and rotations, as well as clearing (also known as masking) of variable-size fragments of a block. All these functions have quite substantial area requirements. Additionally, "re-blocking" often requires simultaneous processing of at least two subsequent message blocks, before any of the corresponding ciphertext blocks is released.

Treatment of incomplete blocks. The treatment of incomplete blocks is a particularly complex operation in AEZ. As already mentioned in the previous section, these blocks are not the last blocks of the message, and in spite of that still require padding. Additionally, as shown in Fig. 1, they also require substantially different parameters j and i of the Tweakable Block Cipher ($E_K^{j,i}$).

Need for pre-computations. In order to support the efficient implementation of TBC, the precomputations are highly desirable. The time of these precomputations and the amount of memory required to store the precomputed look-up tables is dependent on the maximum size of the message and the maximum size of associated data. See Section 4.2 for details.

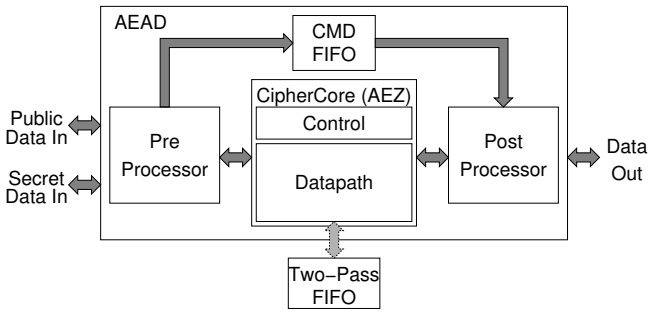


Fig. 2: Top-level design of a two-pass authenticated cipher.

Scheduling. As a result of all the aforementioned factors, the complexity of scheduling and the subsequent difficulty of developing a controller for the hardware implementation of AEZ exceeds the difficulty of any other symmetric cryptographic algorithm the authors are aware of, including all other two-pass CAESAR candidates.

4 Design Architecture

4.1 Interface, Protocol, and Design Parameters

Our implementation is based on the CAESAR Hardware API for Authenticated Ciphers, specified in [6], and its Appendix [7]. This API specifies both the interface and the detailed protocol for communication with the core. On top of that, for high-speed implementations, the authors of this API suggest the use of a top-level design, shown in Fig. 2, and provide the corresponding supporting codes implementing the Pre-Processor, Post-Processor, and CMD FIFO. Our implementation takes full advantage of these resources.

Our hardware design is fully optimized for the maximum throughput to area ratio. Its API and performance makes it suitable for use as a part of practical industry-grade systems based on standard bus interfaces such as ARM AXI-4 (Advanced eXtensible Interface 4) [8].

The hardware design presented in this paper aims to be as complete as the software implementation for the Round 2 version of AEZ (v4), developed by the AEZ team [9], [10]. One significant difference between the software API and the hardware API is as follows: In the software API [11], the only output from authenticated encryption is the Ciphertext, denoted as c, of the length clen. In our hardware API, the output from authenticated encryption is divided into the Ciphertext and the Tag. In case of AEZ, which does not explicitly specify the tag, the tag is understood as follows. For non-empty messages, the tag is a result of enciphering a sequence of zeros, called an authenticator, of the length of τ bits, using the AEZ Encipher algorithm. For empty messages, the tag is a result of calculating the special AEZ-prf function of nonce, associated data, and the authenticator length τ.

The supported parameters are: key length = 384 bits, nonce length = 96 bits, authenticator length (denoted as ABYTES for the length in bytes and τ for the length in bits) = tag length = 16 bytes = 128 bits, maximum AD = $2^{10} - 1$ bytes, and maximum message/ciphertext size = $2^{11} - 1$ bytes. The maximum sizes of the message, ciphertext, and AD were chosen to support the maximum length of the Ethernet v2 packets [12], equal to 1500 bytes. Additionally our choices limit the amount of memory required to implement the Two-Pass FIFO. All these choices are fully compliant with the official CAESAR Hardware API for Authenticated Ciphers, approved by the CAESAR Committee [6].

Our design supports both authenticated encryption and authenticated decryption operation, in such a way that only one of these two operations can be executed at a time (half-duplex). This way our design demonstrates the algorithm's ability to share resources between encryption and decryption. Key scheduling, padding and handling of incomplete blocks is implemented fully in hardware. The result of the decrypted message authentication (Success or Failure) is calculated within the core itself. Any unused portions of the last words of outputs are cleared (filled with zeros) before releasing these words outside of the cipher core.

The secret data input ports, used to enter the key, are separated from the public data input ports, used to enter all remaining data. The Public Data Input (PDI) and Data Output (DO) ports have the data port width equal to 64 bits, the Secret Data Input (SDI) port has the width of 32 bits. Our implementation has only one clock and supports only one input stream at a time.

4.2 Tweakable Block Cipher

Design. AEZ is built on top of the Tweakable Block Cipher (TBC) denoted as $E_K^{j,i}$. In Fig. 1, each call to TBC is denoted as a rectangle with parameters (j, i). The parameter j has discrete integer values -1, 0, 1, and 2 for processing message blocks, and values greater or equal to 3 for processing of nonce and associated data. The parameter i has values varying between 0 and m. For processing of messages, the dependence between the message length (in bytes) and m is as follows: $32 \cdot (m+1) \leq \mathit{message\ length} < 32 \cdot (m+2)$. For processing of messages, $m + 1$, is the number of complete 32-byte message block pairs in Message extended with the 16-byte authenticator. For processing of AD, l is the number of complete 16-byte blocks of AD. When processing incomplete AD blocks, as well as when $j = 0$ or -1, i is set to special values shown in Fig. 1.

The block diagram of the TBC module is shown in Fig. 3. Primary ports of the module are shown in bold font: X is the data input, Y is the result, K is the key. The shaded region is used to calculate Δ, which is a variable dependent on the key K and the parameters j and i. The remaining region is used to perform AES calculations on $X \oplus \Delta$, and an optional XOR of the result of these calculations with Δ.

In the shaded region, the `x2` module represents the Galois field multiplication by two. `I-RAM` and `J-RAM` are two memories used as look-up tables for the precomputed expressions of the form of $2^P I$ and $2^P J$, where $P = 0..15$. The T

register is used to store intermediate values used for the initialization of `I-RAM` and `J-RAM`. The Δ_{i+1} register is used for computing the proper value of Δ to be used by the unshaded region.

Based on the pseudocode of AEZ [1, p. 7] and our assumption about the size of Nonce (96 bits), Δ can take the following values:

- iJ for $j = -1$, $1 \leq i \leq 5$
- iI for $j = 0$, $i = 0, 1, 2, 4, 5, 6$
- $(2^{3+\lfloor (i-1)/8 \rfloor} + ((i-1)\ mod\ 8))I$ for $j = 1, 2$, $1 \leq i \leq m$
- $2^{j-3}L$ for $j = 4, 5$, $i = 0$
- $2^{j-3}L \oplus (2^{3+\lfloor (i-1)/8 \rfloor} \oplus ((i-1)\ mod\ 8))J$ for $j = 3, 5$, $1 \leq i \leq l$.

where,

- $j = 3, 4$, and 5 are used only inside of AEZ-hash(K,T), where T=($[\tau]_{128}$,N,A).
- ($j = 3$, $i = 1$) is used to process the authenticator length, expressed using 128-bits, $[\tau]_{128}$.
- ($j = 4$, $i = 0$) is used only to process a 96-bit Nonce, N, i.e., one incomplete block.
- ($j = 5$, $i \geq 0$) is used only to process AD, which may include an incomplete block (for which i = 0).

Under the assumption that the maximum AD size is $2^{10} - 1$ bytes and the maximum message size is $2^{11} - 1$ bytes, the maximum value of $bn = i - 1$ is equal to $max(bn) = max(i-1) = max(m-1, l-1) = max(l-1) = \lfloor \frac{2^{10}-1}{2^4} \rfloor - 1 = 2^6 - 1$. Thus, $max(3 + \lfloor \frac{i-1}{8} \rfloor) = 3 + \lfloor \frac{2^6-1}{8} \rfloor = 3 + 7 = 10 \leq 15$.

The total number of clock cycles required to pre-compute Δ is based on the number of clock cycles required to calculate the longest possible Δ term, shown in Eq. (1).

$$\Delta \leftarrow 2^{j-3}L \oplus (2^{3+\lfloor (i-1)/8 \rfloor} \oplus ((i-1)\ mod\ 8))J \tag{1}$$

The generalization of Eq. (1) to encompass all possible values of j is shown in Eq. (2), where $Init = 2^{j-3}L$ or 0, $bn = i - 1$, and $A = I,\ J$, or 0.

$$\Delta \leftarrow Init \oplus (bn\ mod\ 8)A \oplus (2^{3+\lfloor bn/8 \rfloor})A \tag{2}$$

Further transformation to convert all terms into 2^P representation is shown in Eq. (3), where $bn[b]$ represents the bit location of bn.

$$\Delta \leftarrow Init \oplus (bn[0])A \oplus (2 \cdot bn[1])A \oplus (4 \cdot bn[2])A \oplus (2^{3+bn[6:3]})A \tag{3}$$

Each term in Eq. (3) requires one clock cycle to calculate. As a result, the maximum number of clock cycles necessary to calculate Δ is 5.

In the *unshaded* region, the Δ_i register is used to store the computed Δ for the final, conditional $\oplus\ \Delta$ operation. This register also frees up the Δ_{i+1} register in the shaded region to allow the pre-computation of Δ for the next input block.

The `State` register is used to store an intermediate value of the state, used as an input to the combinational AES round transformation, denoted by `AES`,

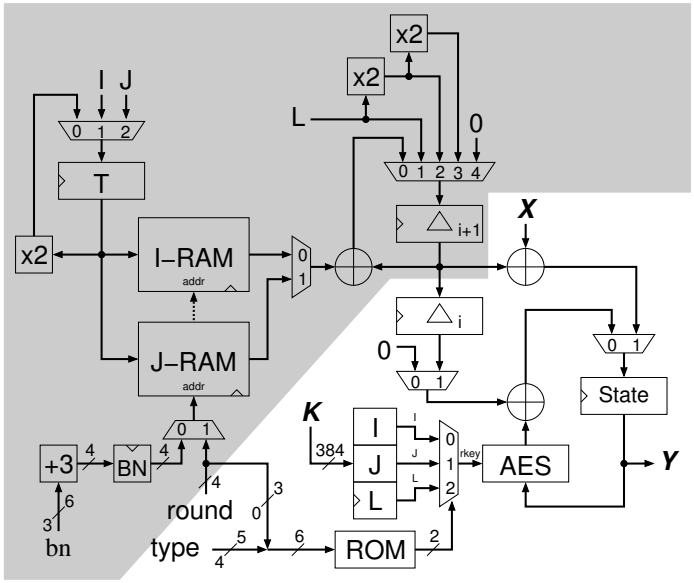


Fig. 3: Block diagram of TBC. Buses have the width of 128 bits unless specified otherwise.

or as an output from the entire TBC function. I, J, and L registers hold three separate 128-bit portions of the 384-bit K. These values serve as round keys to the AES round module. The output of ROM is used to select each round key using the 4-bit *round* signal and the 2-bit *type* signal. The *type* is used to select a key set ($\mathbf{k_1}$, $\mathbf{k_2}$, *or* $\mathbf{K}$). The reader should refer to the pseudocode of AEZ, algorithm $\mathrm{E}_K^{j,i}(X)$, for the exact meaning of $\mathbf{k_1}$ and $\mathbf{k_2}$ [1, p. 7]. The total number of clock cycles required to compute the AES-based transformation, $AES10_k$, $AES4_k$, or $AES4_{k_j}$, is equal to the number of AES rounds plus 1. Thus, depending on a particular transformation, this number is equal to either 5 or 11 clock cycles.

Operation. During the one-time pre-calculations, dependent only on the key K, the I, J, and L registers are initialized with the appropriate portions of K. Then, the RAM modules in the shaded region are filled with $2^P \cdot A$, where $A = I$ or J, and $P = 0..15$. The initialization of I-RAM is achieved by loading I to the T register. The T value is then doubled during each of the subsequent 15 clock cycles. All intermediate values of T are stored at the consecutive locations of I-RAM. The counter *round*, incremented from 0 to 15, is used to address I-RAM during these pre-computations. The same procedure is used for the initialization of J-RAM.

Once the look-up tables stored in I-RAM and J-RAM are initialized, the processing of inputs X can start. A typical operation for each 128-bit block X is separated into two stages. The first stage, located in the shaded region of the

Table 1: Modes of Operation for TBC. Note: $\alpha = 2^{3+bn[6:3]}A$ where $A = I$ or J. Finalization denotes the final XOR with Δ.

mode	$(j,\ i)$	**First Stage (pre-computation)**			**Second Stage (main round)**		
		Init	**I or J**	α	**round**	**Key**	**Finalization**
0	**(0, x)**	0	I	No	4	k_1	No
1	**(1, x)**	0	I	Yes	4	k_1	No
2	**(2, x)**	0	I	Yes	4	k_2	No
3	**(3, 1)**	L	J	Yes	4	k_1	Yes
4	**(4, 0)**	2L	J	No	4	k_1	Yes
5	**(5, 0)**	4L	J	No	4	k_1	Yes
6	**(5, x)**	4L	J	Yes	4	k_1	Yes
7	**(-1, x)**	0	J	No	10	K	No

block diagram, pre-computes the value of Δ, which is dependent on the values of i, j, and K. The second stage, located in the unshaded region, uses the calculated Δ to perform the AES-based computations. The operations of these two stages are categorized into different modes of operation depending on the input parameters j and i, as shown Table 1.

The two stages operate in tandem, with specific actions determined by the *mode*, dependent on the values of j and i, and used by the controller. In case the second stage requires a much longer computation time ($mode = 7$), the subsequent operation of the first stage is stalled until the second stage is completed. For each mode of operation, the first stage begins its operation from the initialization of the Δ_{i+1} register with the $Init$ value. If $j > 0$ and $i > 0$, Δ_{i+1} is then XORed with (bn mod 8) A $= 2^{bn[0]}A \oplus 2^{bn[1]}A \oplus 2^{bn[2]}A$ using three clock cycles. In the last clock cycle of the first stage computations, Δ_{i+1} is XORed with α.

The second stage, in the first clock cycle, XORs the pre-computed Δ value with the input X. The remaining clock cycles are spent on computing the AES rounds. Finalization is performed in the last clock cycle, if required.

Both stages operate in parallel, with the second stage performing calculations dependent on the current inputs X, j, and i, and the first stage performing calculations dependent on the next set of inputs j and i.

4.3 CipherCore

The CipherCore Datapath of AEZ is shown in Fig. 4. In order to limit the size of this block diagram and preserve its readability, control signals, serving as inputs to majority of medium-level components, such as TBC, NPAD, MASK and PAD, are not explicitly shown in this diagram.

TBC is the main encryption module. Its internal structure and operation is described in Section 4.2. This module serves as a focal point for all processing needs in our design. It processes 128 bits of data at a time (half of a block pair for message/ciphertext and a full block for associated data). The surrounding

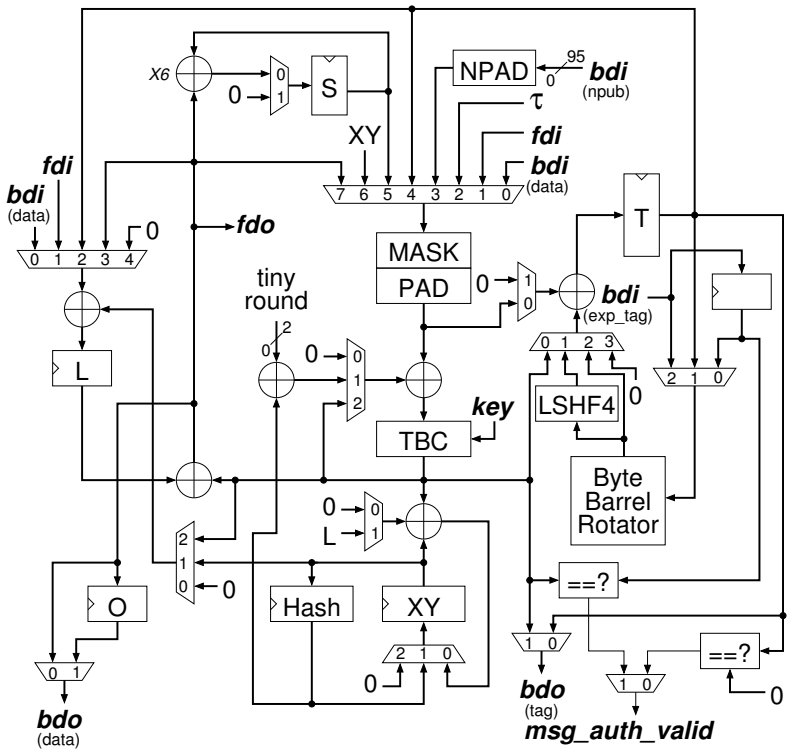


Fig. 4: The CipherCore Datapath of AEZ. Buses have the width of 128 bits unless specified otherwise.

logic is used to facilitate the transfer of data and storage of intermediate results for the main processor. The following description summarizes the usage of the primary auxiliary units.

The `T` register holds data that is being operated on by `TBC`. It is also used as a temporary register to store intermediate values when data shifting is required. The `XY` register holds the accumulated value of Δ from Fig. 1 or $\Delta \oplus XY$ where $XY = XY_1 \oplus ... \oplus XY_m \oplus XY_u \oplus XY_v$ and $XY = X$ for the first pass, and Y for the second pass.

The `S` register is used to hold the S value calculated at the end of the first pass, during processing of M_x and M_y, as shown in Fig. 1. The `O` register is used to hold any output that needs to be delayed in order for the output format to be the same as in the software implementation. The `NPAD` module performs 10* padding for the 96-bit nonce. The `MASK` and `PAD` modules are used to perform masking and padding operations required during processing of the last-but-one message block pair with indices u and v, as well as during AEZ-Tiny operations.

The `Byte Barrel Rotator` module is a variable rotation module. It can rotate by any integer multiple of a full byte. `LSHF4` is a 4-bit left shifter used only for

the AEZ-Tiny operation. It is required when an input block is of an odd size in bytes, and data needs to be split at a boundary of a nibble.

5 Timing Analysis

5.1 Latency

The design latency is given by Eq. (4). It is a function of T_{Hash}, T_{PRF}, T_{Tiny} and T_{Core}, shown in Eqs. (5), (6), (7), and (8), respectively. T_{Core} is a function of T_{Full}, T_{UV}, and T_{XY} shown in Eqs. (9), (10), and (11), respectively. In all these equations |AD| and |M| represent the lengths of AD and message, respectively, in bits.

The detailed formulas are important, as they allow the accurate timing analysis for multiple AD and message sizes, and not only for the case of long messages.

$$\begin{aligned} Latency &= T_{keysetup} + T_{Hash} + T_{PRF} + T_{Tiny} + T_{Core} \\ &= 36 + T_{Hash} + T_{PRF} + T_{Tiny} + T_{Core} \end{aligned} \tag{4}$$

$$T_{Hash} = 15 + \left\lceil \frac{|AD|}{128} \right\rceil \cdot 5 \tag{5}$$

$$T_{PRF} = \begin{cases} 0, & \text{if } |M| > 0 \\ 14, & \text{otherwise} \end{cases} \tag{6}$$

$$T_{Tiny} = \begin{cases} 0, & \text{if } |M| \geq 128 \\ 49, & \text{otherwise} \end{cases} \tag{7}$$

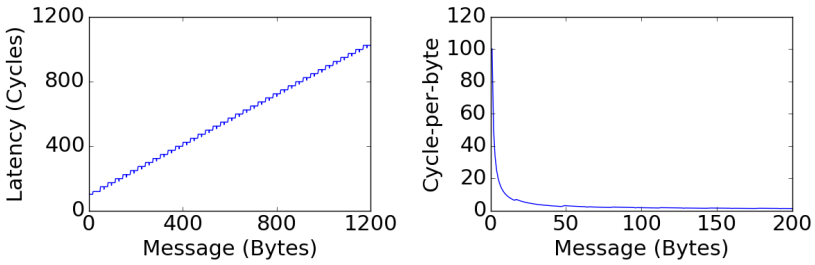
$$T_{Core} = \begin{cases} 0, & \text{if } |M| < 128 \\ 12 + T_{XY}, & \text{elif } |M| = 128 \\ 12 + T_{UV} + T_{XY}, & \text{elif } (|M| - 128) < 256 \\ 12 + T_{Full} + T_{XY}, & \text{elif } (|M| - 128) \bmod 256 = 0 \\ 12 + T_{Full} + T_{UV} + T_{XY}, & \text{otherwise} \end{cases} \tag{8}$$

$$T_{Full} = 25 \cdot \left\lfloor \frac{|M| - 128}{256} \right\rfloor + 5 \tag{9}$$

$$T_{UV} = 11 \cdot \left\lceil \frac{(|M| - 128) \bmod 256}{128} \right\rceil + 13 + \begin{cases} 2, & \text{if } (|M| - 128) \bmod 256 = 128 \\ 4, & \text{otherwise} \end{cases} \tag{10}$$

$$T_{XY} = \begin{cases} 38, & \text{if } (|M| - 128) \bmod 256 > 0 \\ 32, & \text{otherwise} \end{cases} \tag{11}$$

In Fig. 5, we illustrate the quite complex dependence of the (a) latency in clock cycles, and (b) number of clock cycles per byte, on the size of the message in bytes, assuming an empty AD. Based on Fig. 5(b), the number of clock cycles per byte reaches the close-to-optimal performance already at message sizes around 50 bytes.



(a) Latency vs. Message Size (b) Cycle-per-byte vs. Message Size

Fig. 5: The AEZ hardware module latency and the number of cycles per byte as a function of the message size for $|AD| = 0$

5.2 Throughput

Throughput for authenticated encryption and decryption of long messages is given by Eq. (12) and Eq. (13). Eq. (12) applies when $|M| = 0$, and $|AD| \gg 0$, where $\gg$ denotes "much bigger". It is based on the time it takes to perform the AEZ Hash operation (bottom left diagram of Fig. 1). Similarly, Eq. (13) applies when $|AD| = 0$, and $|M| \gg 0$. It is based on the time it takes to perform AEZ Core operation on a full block pair (top left diagram of Fig. 1).

$$Throughput_{AD} = \frac{128}{5} \cdot ClkFreq. \tag{12}$$

$$Throughput_{M} = \frac{256}{25} \cdot ClkFreq. \tag{13}$$

6 Benchmarking in Hardware

6.1 Hardware Results and Comparison with Other CAESAR Candidates

The resource utilization and the maximum clock frequency of the main components of AEZ on Virtex-6 FPGA is shown in Table 2. The TBC module requires about 48% of the flip-flops and 37% of the total LUTs as compared to the CipherCore module. The speed of the design is reduced by a factor of 8% when the unit is integrated with the surrounding logic. The complete unit with the CAESAR Hardware API support (AEAD) requires an additional 15% of flip-flops and 10% of LUTs, on top of the resources required by the CipherCore module. The maximum frequency of operation remains exactly the same.

The comparison with all other Round 2 CAESAR candidates (except Tiaoxin), using the same hardware API, is summarized in Table 3. All results have been obtained using exactly the same FPGA device and FPGA tool versions. Benchmarking involved the optimization of tool options using ATHENa [13], with the

Table 2: Components analysis of AEZ unit on Virtex-6 xc6vlx240tff1156-3 FPGA device

	Resource Utilization		Frequency
	FFs	LUTs	(MHz)
TBC	927	1527	362
CipherCore	1983	4166	335
AEAD	2347	4597	335

same optimization scheme and effort applied to all candidates. The source of these results is the ATHENa database of results [14], reporting FPGA performance for all implementations of Round 2 candidates submitted for benchmarking in June–August 2016. Each Round 2 CAESAR candidate family (except Tiaoxin) is represented in this study by one or more variants recommended by the submitter teams. For all the candidates and AES-GCM, the throughput is based on either encryption or decryption throughput, whichever is lower. Only the performance of the best variant in terms of the Throughput to Area ratio is reported in [14] and in Table 3, with LUTs used as a primary Area metric.

Since based on the CAESAR Hardware API [6], the implementations of single-pass authenticated ciphers are expected to support all message lengths $\leq 2^{32} - 1$, and implementations of two-pass authenticated ciphers are expected to support all lengths $\leq 2^{11} - 1$, it is natural and fair to compare implementations of both types of ciphers for the maximum message length common for both types of ciphers, which is $2^{11} - 1$.

Additionally, 2Kbytes is a practical limit for majority of secure networking protocols, such as IPSec – a primary target for high-speed hardware implementations of authenticated encryption. Authenticated encryption without intermediate tags is in general not a good match for applications requiring protection of large volumes of data-at-rest, due to large access times for reading and writing.

The implementers of 7 single-pass authenticated ciphers included in our comparison (AES-GCM, Deoxys, Joltik, OCB, OMD, PAEQ, and SCREAM) specifically supported the two possible maximum AD/message lengths. All corresponding results presented in Table 3 have been generated with the choice of the maximum AD/message equal to $2^{11} - 1$. This choice has appeared to benefit in a noticeable way only the two of them, OCB and OMD, using a precomputed look-up table, with the size dependent on the maximum AD/message length.

For the remaining candidates, we contacted the designers of the implementations listed in Table 3, and asked them explicitly whether they see any way of optimizing their designs (in terms of area and/or maximum clock frequency) in case the maximum AD/message length is smaller or equal to $2^{11} - 1$. None of the designers responded positively to this question. Similarly, our own analysis and preliminary results led to the conclusion that the maximum benefit in terms of the throughput to area ratio, resulting from applying a lower limit on the AD/message length, is not likely to exceed 3% for any of the remaining one-pass Round 2 CAESAR candidates.

On top of that, both single-pass and two-pass algorithms require *external* memory for the complete functionality, including the temporary storage of decrypted message. In an optimized implementation of the entire system including a two-pass AEAD core, the Two-Pass FIFO and the Output FIFO could be implemented using the same resources. The amount of logic (LUTs) required to multiplex between these two functions of an external memory would be negligible compared to the size of the entire system.

As a result, we believe that the need for an external Two-Pass FIFO, implemented using dedicated FPGA resources, such as Block RAMs, does not put two-pass algorithms in any noticeable disadvantage that could affect the ranking of the candidates (especially to the extent higher than other, more important factors, such as different designer skills and coding styles, different amount of time and effort spent on optimization, etc.)

Based on the results presented in [14], it is fair to say that AEZ outperforms all AES-based CAESAR candidates, other than AEGIS and Deoxys, such as CLOC, ELmD, OCB, AES-OTR, SILC, POET, AES-COPA and SHELL. Our implementation also outperforms the implementation of the only other two-pass Round 2 candidate variant, reported in [14], HS1-SIV. Our implementation of AEZ beats the equivalent implementation of HS1-SIV by a factor of 1.23 in terms of Throughput, 1.83 in terms of Area, and a combined factor of 2.26 in terms of the Throughput/Area ratio. Its Throughput to Area ratio is lower only than that of 11 mostly permutation-based algorithms, none of which fulfills the requirements of robust authenticated encryption (RAE), or even misuse-resistant authenticated encryption (MRAE).

6.2 Comparison with the Optimized Software Implementation

The preliminary results of the software benchmarking using SUPERCOP place AEZ among the top 5 authenticated ciphers on the amd64-architecture platforms [15]. The software benchmark of the optimized software implementation, available at [10], was done on a Skylake-S Intel Core i5-6600 3.3 GHz. The compiler and compilation flags used were: GCC 5.5 with "-march=native -O3". The optimized software implementation was able to achieve the performance of 0.64 cycles-per-byte, equivalent to the throughput of 41.25 Gbit/s for long messages. Comparing to our hardware AEZ core performance on Virtex-6 FPGA, the software is able to achieve approximately 12 times higher throughput, while running at about 10 times higher clock frequency.

Clearly, an optimized software implementation of an AES-based authenticated cipher, running on a modern microprocessor, can easily outperform the corresponding single-core hardware implementation, not just for AEZ, but for majority of other CAESAR candidates. However, one must remember that the hardware resources required by a modern microprocessor, as well as power and energy consumption, are likely much higher than resources required by a single core of AEZ.

On modern FPGAs and All-Programmable Systems on Chip (such as Xilinx Zynq), multiple AEZ cores can be placed and run in parallel to either hard or soft

Table 3: Comparison with other CAESAR candidates, with key sizes greater or equal to 96 bits, on Virtex 6 FPGA.

		Frequency (MHz)	**Throughput (Mbit/s)**	**Area**		**TP/A**	
				(LUTs)	**(SLICEs)**	**(Mbit/s/ LUTs)**	**(Mbit/s/ SLICEs)**
1	MORUS	179.7	46002	3898	1216	**11.801**	37.831
2	ACORN	347.7	11127	1194	421	**9.319**	26.430
3	TriviA-ck	300.2	19213	2310	895	**8.317**	21.467
4	ICEPOLE	304.0	44464	5734	1995	**7.754**	22.288
5	AEGIS	203.1	52001	7980	2143	**6.516**	24.266
6	Ketje	229.5	7345	1270	456	**5.783**	16.107
7	NORX	170.5	16368	2968	1022	**5.515**	16.016
8	ASCON	361.0	5134	1620	489	**3.169**	10.499
9	STRIBOB	276.1	11750	4839	1376	**2.428**	8.539
10	Keyak (River)	163.6	7417	6234	1751	**1.190**	4.236
	AES-GCM	*278.3*	*3239*	*3175*	*1053*	***1.020***	*3.076*
11	Deoxys (NR-128-128)	327.3	2793	3142	951	**0.889**	2.937
12	**AEZ**	**335.3**	**3434**	**4597**	**1246**	**0.747**	**2.756**
13	CLOC	254.6	2963	3983	1154	**0.744**	2.568
14	ELmD	247.5	3168	4302	1607	**0.736**	1.971
15	OCB	292.7	3122	4249	1348	**0.735**	2.316
16	PRIMATEs-GIBBON	224.0	1280	1807	653	**0.708**	1.960
17	Joltik (NR-128-64)	439.9	880	1292	524	**0.681**	1.679
18	Minalpher	280.9	1831	2879	1104	**0.636**	1.659
19	PAEQ	258.9	4537	8328	2300	**0.545**	1.973
20	AES-OTR	256.9	2741	5102	1385	**0.537**	1.979
21	SCREAM	170.4	1039	2052	834	**0.506**	1.246
22	Pi-Cipher	170.0	1740	3535	1077	**0.492**	1.616
23	SILC	280.7	1562	3378	989	**0.462**	1.579
24	PRIMATEs-HANUMAN	225.1	693	1769	626	**0.392**	1.107
25	POET	231.2	2959	7695	2444	**0.385**	1.211
26	HS1-SIV	221.7	2769	8392	2219	**0.330**	1.248
27	AES-COPA	214.9	2500	7754	2358	**0.322**	1.060
28	OMD	242.2	940	3562	1243	**0.264**	0.756
29	AES-JAMBU (SIMON)	209.8	186	1376	453	**0.135**	0.411
30	SHELL	16.3	522	81197	22830	**0.006**	0.023

embedded microprocessor core (such as ARM or MicroBlaze). Their availability would free the microprocessor to perform other critical tasks. It would also allow significantly outperforming a single dedicated microprocessor core. For example, the largest Xilinx Virtex-6 FPGA (XC6VLX760) can host up to 95 AEZ Cores, reaching throughput in excess of 326 Gbit/s.

Results of software implementations of AEZ on multiple other platforms, including ARM, can be found in [15].

7 Conclusions

We have developed an efficient implementation of AEZ that outperforms comparable implementations of the majority of other AES-based Round 2 CAESAR candidates. It places 12th in terms of the Throughput to Area ratio, in the ranking of 28 candidates participating in the hardware benchmarking study (assuming the maximum message length of $2^{11}-1$ bytes), and is outperformed only

by single-pass, mostly permutation-based algorithms. Our preliminary analysis strongly suggests that AEZ can outperform majority of the CAESAR candidates and the current standard, AES-GCM, in software, approximately match the performance of AES-GCM in hardware, and at the same time offer a new unprecedented level of resistance against the cipher misuse.

References

1. V. T. Hoang, T. Krovetz, and P. Rogaway, "AEZ v4.1: Authenticated Encryption by Enciphering," Oct 2015. [Online]. Available: http://web.cs.ucdavis.edu/~rogaway/aez/aez.pdf
2. ——, "Robust authenticated-encryption: AEZ and the problem that it solves," in *Advances in Cryptology - EUROCRYPT 2015, Sofia, Bulgaria, April 26-30, 2015*, pp. 15–44.
3. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. (2016, January) Cryptographic competitions. [Online]. Available: http://competitions.cr.yp.to/index.html
4. P. Rogaway and T. Shrimpton, "A provable-security treatment of the key-wrap problem," in *Advances in Cryptology - EUROCRYPT 2006, St. Petersburg, Russia, May 28 - June 1, 2006*, pp. 373–390.
5. C. Arnould, "Towards Developing ASIC and FPGA Architectures of High-Throughput CAESAR Candidates," Master's thesis, ETH Zurich, March 2015.
6. E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, P. Yalla, J.-P. Kaps, and K. Gaj, "CAESAR Hardware API," Cryptology ePrint Archive, Report 2016/626, 2016, http://eprint.iacr.org/2016/626.
7. Cryptographic Engineering Research Group (CERG) at GMU. (2016, Jun.) Addendum to the CAESAR Hardware API v1.0. [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=CAESAR
8. ARM. AMBA Specifications. [Online]. Available: http://www.arm.com/products/system-ip/amba-specifications.php
9. T. Krovetz, "AEZ v4.1 reference code," Sep 2015. [Online]. Available: http://www.cs.ucdavis.edu/~rogaway/aez
10. ——, "Aez v4.1 aes-ni version," Oct 2015. [Online]. Available: http://www.cs.ucdavis.edu/~rogaway/aez
11. (2014, Jan) Caesar call for submissions, final. [Online]. Available: https://competitions.cr.yp.to/caesar-call.html
12. C. Hornig, "A standard for the transmission of ip datagrams over ethernet networks," Internet Requests for Comments, RFC Editor, STD 41, April 1984.
13. K. Gaj, J.-P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, and B. Y. Brewster, "ATHENa – automated tool for hardware evaluation: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs," in *20th International Conference on Field Programmable Logic and Applications - FPL 2010*. IEEE, 2010, pp. 414–421.
14. Cryptographic Engineering Research Group (CERG) at GMU. (2015, Jul.) GMU ATHENa Database of Results. [Online]. Available: https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/rankings_view
15. D. Bernstein and T. L. (editors). (2016, October) eBACS: ECRYPT Benchmarking of Cryptographic Systems. [Online]. Available: https://bench.cr.yp.to