CAESAR Hardware API

Ekawat Homsirikamol, William Diehl, Ahmed Ferozpuri, Farnoud Farahmand, Panasayya Yalla, Jens-Peter Kaps, and Kris Gaj

Cryptographic Engineering Research Group
George Mason University
Fairfax, Virginia 22030
email: {ehomsiri, wdiehl, aferozpu, ffarahma, pyalla, jkaps, kgaj}@gmu.edu

Abstract. In this paper, we define the CAESAR hardware Application Programming Interface (API) for authenticated ciphers. In particular, our API is intended to meet the requirements of all algorithms submitted to the CAESAR competition. The major parts of our specification include: minimum compliance criteria, interface, communication protocol, and timing characteristics supported by the core. All of them have been defined with the goals of guaranteeing (a) compatibility among implementations of the same algorithm by different designers, and (b) fair benchmarking of authenticated ciphers in hardware.

1 Minimum Compliance Criteria

The recommended minimum compliance criteria are listed below:

1.1 Encryption/Decryption

Authenticated encryption and decryption should be implemented within one core, but only one of these two operations can be executed at a time (half-duplex).

This feature demonstrates an algorithm's ability to use shared resources for encryption and decryption.

Alternatives (not recommended):

a) separate cores for encryption and decryption (simplex)
b) authenticated encryption and decryption within one core, with both operations capable of running in parallel (full-duplex).

1.2 Variants

Only a variant indicated in the cipher specification as the primary recommendation has to be implemented.

Other variants, if implemented, should be selectable by changing the default values of generics or constants before synthesis. The implementation of these variants should not affect any benchmarking results for the main variant.

1.3 Key scheduling

Key scheduling should be fully implemented within the hardware core.

This approach takes into account very different contributions of the key scheduling unit to the entire cipher core area, which are specific for each algorithm.

An alternative (not recommended):

a) generation of round keys outside of the cipher core, e.g., in software.

1.4 Incomplete blocks

The core should properly handle incomplete blocks in associated data, message, and ciphertext.

Handling of incomplete blocks substantially increases the core area for multiple candidates, due to the large area required for variable shifts.

An alternative (not recommended):

a) handling only associated data, messages, and ciphertexts composed of full blocks.

1.5 Padding

Padding in hardware, assuming that an unused portion of the last input data word is filled with zeros.

Padding cost, in terms of area, is algorithm dependent, and not negligible. In some algorithms, padding in software may need to be reversed in hardware because the tag calculation uses an unpadded last block.

Alternatives (not recommended):

a) Padding in hardware, assuming that an unused portion of the last block is filled with zeros.
b) Padding in software, followed, if needed, by modifications of the last blocks in hardware.

1.6 Unused portions of the last block

Clearing any unused portions of the last word during encryption and decryption.

An alternative (not recommended):

a) potentially leaking some key-related data using unused portions of the last block.

1.7 Decrypted message release

Releasing the decrypted message blocks immediately.

We assume that the delayed release of decrypted data, dependent on the result of authentication, will be handled by an external circuit, which is FIFO-based and similar for each candidate.

An alternative (not recommended):

a) storing a decrypted message internally, until the result of verification is known.

Pros: More complete functionality.
Cons: Complicates the design and benchmarking. Makes the calculation of the output latency and throughput dependent on the output buffer size and implementation details (e.g., support for simultaneous reading and writing).

1.8 Empty AD/message/ciphertext

Allowing empty associated data, empty message/ciphertext, and and empty input (no AD, no message/ciphertext)

Empty input could be used together with the input message number, Npub, for user authentication.

Alternatives (not recommended):

a) not allowing empty associated data
b) not allowing empty message/ciphertext
c) not allowing empty input.

1.9 Supported maximum size of AD/plaintext/ciphertext

single-pass authenticated ciphers:	2^{32} bytes
two-pass authenticated ciphers:	2^{11} bytes

Maximum sizes defined in the CAESAR candidates' specifications are unrealistic. Too large values may affect both area and maximum clock frequency of the hardware core (e.g., because of wide internal counters).

2^{11} bytes $>$ 1500 bytes $=$ maximum transmission unit (MTU) of popular communication protocols, such as Ethernet v2.

1.10 Fractions of bytes

The size of all inputs is assumed to be expressed in bytes. As a result, the core should support only inputs composed of full bytes. No fractions of bytes should be allowed.

An alternative (not recommended):

a) the size of inputs expressed in bits.

Allowing inputs of arbitrary size in bits would substantially increase the area required for handling of incomplete blocks.

1.11 Maximum number of independent streams of data processed in parallel

The core should process only one stream of data (i.e., a single independent input understood as composed of any subset of Npub, Nsec, AD, Message, Ciphertext, and Tag, supported by the encryption or decryption operation of a given authenticated cipher) at a time (without an overlap). We refer to such core as a single-stream implementation. The single-stream implementation may still take advantage of parallel processing for blocks belonging to the same input/stream.

An alternative (not recommended):

a) a multi-stream implementation that supports processing of multiple independent inputs/streams in parallel.

In the multi-stream implementations:

- Throughput is limited only by the maximum circuit area.
- Multiple messages/ciphertexts processed in parallel would require multiple public data input (PDI) and data outputs (DO) ports. See Section 2 for the detailed descriptions of these ports.

1.12 External memory

Single-pass algorithms:	No
Two-pass algorithms:	Yes (but only for results of the first pass)

1.13 One clock domain

One clock input. Clock operating at the maximum clock frequency determined by the critical path located entirely inside of the hardware module.

An alternative (not recommended):

a) separate clocks for input module, output module, and cipher core.

Pros: Possible smaller values of data bus widths.
Cons: Difficulties with determining the maximum clock frequency of the cipher core.

1.14 Passing unchanged parts of the input to the output

Parts of the data inputs that are not changed by encryption or decryption operations, respectively, are not passed to the output. In particular, Npub and AD are not a part of the output from either encryption or decryption. See Fig. 4.

This assumption removes the need for any bypass FIFO necessary to pass any unchanged data to the output. Any formatting of an output from decryption, for the purpose of transmission through the network or decryption, is assumed to be performed outside of the cipher core.

An alternative (not recommended):

a) Passing unchanged parts of the input to the output.

Pros: More complete functionality.
Cons: The design time and area overhead for adding standard functionality that may be implemented in a coherent way outside of the authenticated cipher core.

1.15 Permitted widths of data ports (in bits)

Public Data Input (PDI) and Data Output (DO) ports:

Lightweight implementations:	$w = 8, 16, 32$
High-speed implementations:	$32 \leq w \leq 256$.

Secret Data Input (SDI) ports:

Lightweight implementations:	$w = 8, 16, 32$
High-speed implementations:	$32 \leq sw \leq 64$.

See Section 2 and Fig. 1 for the exact meaning of PDI, SDI, DO, w and sw.

Implementations of a particular authenticated cipher, with the same w and sw, following all other minimum compliance criteria, should be clearly compatible. Implementations with different values of w or sw should be compatible under the assumption that the decryption input is reformatted in software or hardware (from one input word width to another) using a universal function/circuit, common for all candidates.

2 Interface

The general idea of the CAESAR interface for an authenticated cipher core (denoted by AEAD) is shown in Fig. 1. The interface is composed of three major data buses for:

– Public Data Inputs (`PDI`)

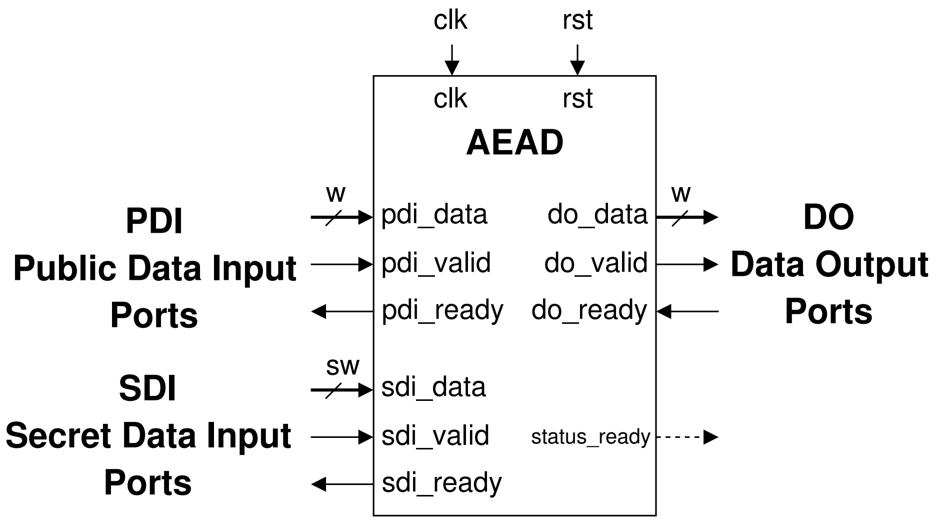


Fig. 1: AEAD Interface

- Secret Data Inputs (SDI), and
- Data Outputs (DO), respectively,

as well as the corresponding handshaking control signals, named *valid* and *ready*. The *valid* signal indicates that the data is ready at the source, and the *ready* signal indicates that the destination is ready to receive them.

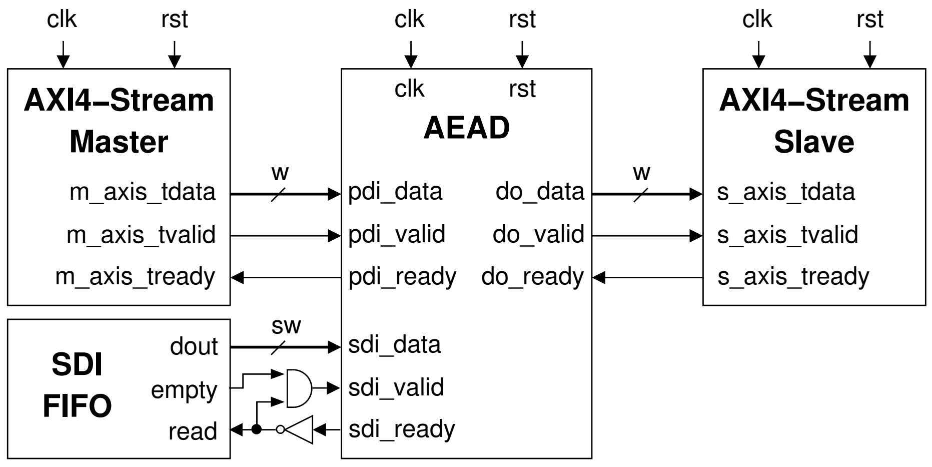


Fig. 2: Typical external circuits: AXI4-Stream IPs

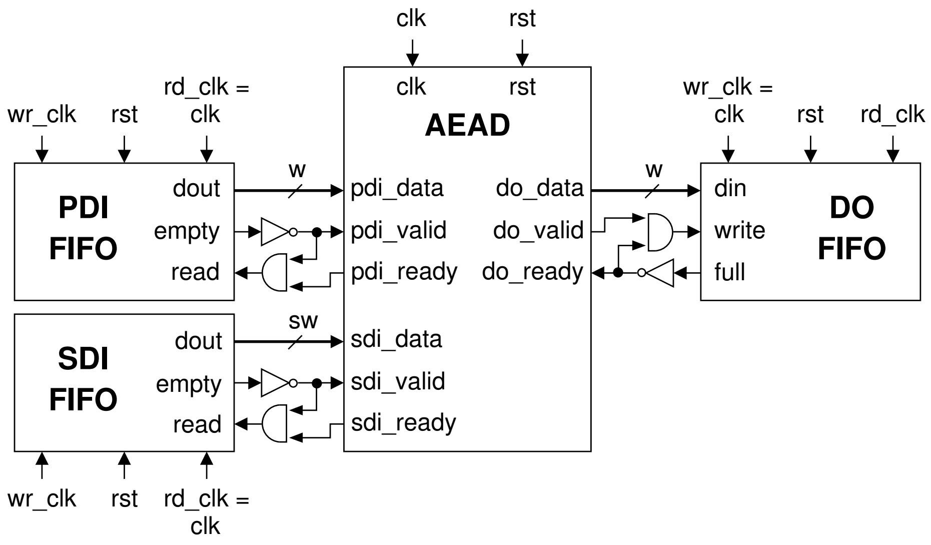


Fig. 3: Typical external circuits: FIFOs

The physical separation of Public Data Inputs (such as the message, associated data, public message number, etc.) from Secret Data Inputs (such as the key) is dictated by the resistance against any potential attacks aimed at accepting public data, manipulated by an adversary, as a new key.

The handshaking signals are a subset of major signals used in the AXI4-Stream interface [1]. As a result AEAD can communicate directly with the AXI4-Stream Master through the Public Data Input, and with the AXI4-Stream Slave through the Data Output, as shown in Fig. 2. At the same time, AEAD is also capable of communicating with much simpler external circuits, such as FIFOs, as shown in Fig. 3.

In both cases, the Secret Data Input is connected to a FIFO, as the amount of data loaded to the core using this input port does not justify the use of a separate AXI4-Stream Master, such as DMA.

An additional advantage of using FIFOs at all data ports is their potential role as suitable boundaries between the two clock domains, used for communication and computations, accordingly. This role is facilitated by the use of separate read and write clocks, shown in Fig. 3 as `rd_clk` and `wr_clk`, accordingly. For a better compatibility with the AXI communication interface, all FIFOs mentioned in our description are assumed to operate in the First-Word Fall-Through mode (as opposed to the standard mode).

The reset input can be either synchronous or asynchronous, and either active-high or active-low, depending on the conventions used in a given technology (e.g., FPGA vs. ASIC), as well as the personal preference of the designers.

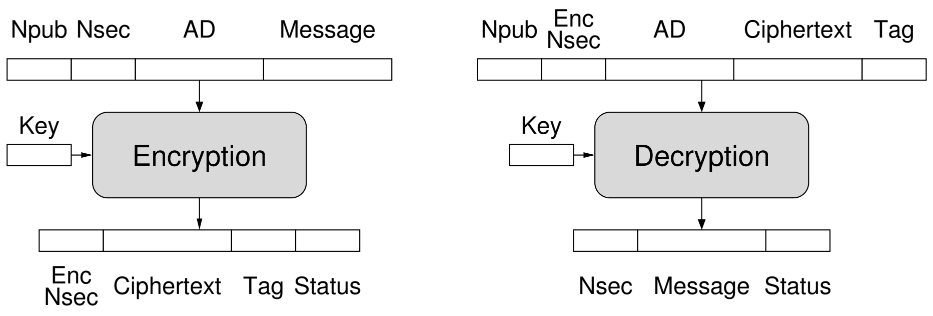


Fig. 4: Input and Output of an Authenticated Cipher Core. Notation: Npub - Public Message Number, Nsec - Secret Message Number, Enc Nsec - Encrypted Secret Message Number, AD - Associated Data

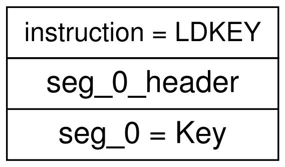


Fig. 5: Format of Secret Data Input for loading the key

3 Communication Protocol

All parts of a typical input and a typical output of an authenticated cipher are shown in Fig. 4, for encryption and decryption, respectively. Npub denotes Public Message Number, such as Nonce or Initialization Vector. Nsec denotes Secret Message Number, which was recently introduced in some authenticated ciphers and is a part of the CAESAR software API [2]. Both Npub and Nsec are typically assumed to be unique for each message encrypted using a given key. The difference is that Npub is sent to the other side in clear, while Nsec is sent in the encrypted form.

All parts of an input to encryption, other than a key, are optional, and can be omitted. If a given part is omitted, it is assumed to be an empty string.

The proposed format of the Secret Data Input is shown in Fig. 5. The entire input starts with an instruction, which in case of `SDI` is limited to Load Key (LDKEY). The instruction is followed by segments. Each segment starts with a separate header, describing its type and size. In case of `SDI`, the only segment type necessary to meet the minimum compliance criteria is: `Key`, denoting a string of bits carrying an authenticated cipher key.

The proposed format of the Public Data Input is shown in Fig. 6. The allowed instruction types are: Activate Key (ACTKEY), Authenticated Encryption (ENC), and Authenticated Decryption (DEC). The Activate Key instruction, typically directly precedes the Authenticated Encryption or Authenticated Decryption instruction. `PDI` is divided into segments. Segment types allowed

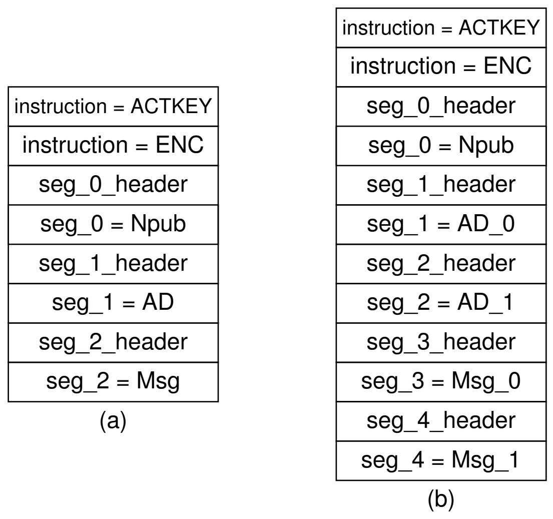


Fig. 6: Format of Public Data Input in case of a) one segment for each data type, b) multiple segments for AD and Message

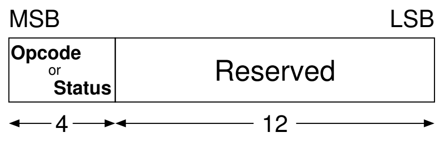
during authenticated encryption include: Public Message Number (Npub), Secret Message Number (Nsec), Associated Data (AD), and Message. Segment types allowed during authenticated decryption include: Public Message Number (Npub), Encrypted Secret Message Number (Enc Nsec), Associated Data (AD), Ciphertext, and Tag.

Any segment type can be omitted, if it is not required by a given cipher. *However, empty AD, empty message, and empty ciphertext must be provided using a separate segment, with the Segment Length field of the respective header set to 0.* Public and Secret Message Numbers can only use one segment, as their sizes are typically quite small (in the range of 16 bytes).

The Associated Data and Message can be (but do not have to be) divided into multiple segments (as shown in Fig. 6b). *The maximum size of each segment is assumed to be $2^{16} - 1$ bytes for single-pass authenticated ciphers, and 2^{11} bytes for two-pass authenticated ciphers.* The primary reasons for dividing AD and Message into multiple segments is that the full message size may be unknown when authenticated encryption starts, and/or the maximum single segment size (specified above) is smaller than the message size.

The instruction/status format is shown in Fig. 7. For instruction, the Opcode field determines which operation should be executed next. For status, the Opcode field is replaced by the Status field, which can be set to only two values, PASS or FAIL.

The segment header format is shown in Fig. 8. The segment header consists of:



Opcode:

0010 – Authenticated Encryption (ENC)
0011 – Authenticated Decryption (DEC)
0100 – Load Key (LDKEY)
0111 – Activate Key (ACTKEY)

Status:

1110 – Success
1111 – Failure

Others – Reserved

Note: If w < 16, more than one word should be used

Fig. 7: Instruction/Status Format

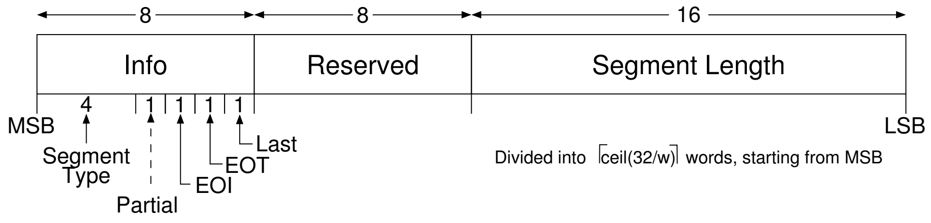


Fig. 8: Segment Header Format

Table 1: Segment Type Encoding

Encoding	Type	Encoding	Type
0000	*Reserved*	1000	Tag
0001	AD	1001	*Reserved*
0010	Npub||AD	1010	Length
0011	AD||Npub	1011	*Reserved*
0100	Plaintext	1100	Key
0101	Ciphertext	1101	Npub
0110	Ciphertext||Tag	1110	Nsec
0111	*Reserved*	1111	Enc Nsec

– 4-bit *Segment Type* indicates the type of data that the current segment contains. The type encoding is defined in Table 1.
– 1-bit optional *Partial* bit indicates that the current segment contains an incomplete block of message or the corresponding ciphertext. The only CAESAR candidate we are aware of that requires this bit is the Round 2 AES-COPA.
– 1-bit *EOI* (End-Of-Input) indicates that the current segment is the last segment of input other than the *Length* segment, *Tag* segment, or any empty segment.
– 1-bit *EOT* (End-Of-Type) indicates that the current segment is the last segment of the current *Segment Type*.
– 1-bit *Last* indicates that the current segment is the last segment, i.e. no more segments are associated with the given instruction.
– 8 reserved bits for future extensions.
– 16-bit *Segment Length* to specify the size of data in the given segment in *bytes*.

Several examples, illustrating the correct values of the flags EOI, EOT, and Last, for multiple realistic scenarios are shown in Table 2.

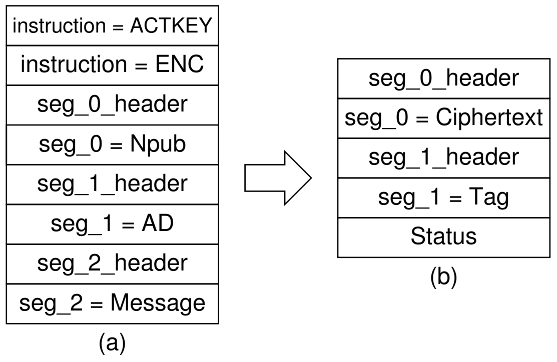


Fig. 9: Format of Public Data Input (PDI) and Data Output (DO) of authenticated decryption operation for ciphers that do not use Nsec: a) PDI, b) DO

Figures 9 and 10 present typical format of input (`PDI`) and output (`DO`) of authenticated encryption and decryption operation, respectively, for the ciphers that do not use Nsec. At the input (`PDI` ports), a message typically starts with the key activation instruction (ACTKEY), followed by an operational instruction (ENC or DEC). Header and data segments for different types of data subsequently follow. For encryption and decryption operation, the order typically is Npub, AD, Data (Plaintext or Ciphertext) and Tag (for decryption only). The order of segment types that can be processed by a given core is a feature of the specific implementation, and needs to be clearly documented.

Table 2: Examples of correct values of input flags for encryption and decryption operations in different scenarios. BS = block size. MBS = an integer multiple of the block size.

Types	Size	EOI	EOT	Last	Types	Size	EOI	EOT	Last	Types	Size	EOI	EOT	Last	Types	Size	EOI	EOT	Last
Typical																			
Example A: AD = 0, Message = 0										**Example B: AD = 0, Message > 0**									
Encryption					**Decryption**					**Encryption**					**Decryption**				
Npub	>0	1	1	0	Npub	>0	1	1	0	Npub	>0	0	1	0	Npub	>0	0	1	0
AD	0	0	1	0	AD	0	0	1	0	AD	0	0	1	0	AD	0	0	1	0
Msg	0	0	1	1	CT	0	0	1	0	Msg	>0	1	1	1	CT	>0	1	1	0
					TAG	>0	0	1	1						TAG	>0	0	1	1
Example C: AD > 0, Message = 0										**Example D: AD > 0, Message > 0**									
Encryption					**Decryption**					**Encryption**					**Decryption**				
Npub	>0	0	1	0	Npub	>0	0	1	0	Npub	>0	0	1	0	Npub	>0	0	1	0
AD	>0	1	1	0	AD	>0	1	1	0	AD	>0	0	1	0	AD	>0	0	1	0
Msg	0	0	1	1	CT	0	0	1	0	Msg	>0	1	1	1	CT	>0	1	1	0
					TAG	>0	0	1	1						TAG	>0	0	1	1
Ciphertext Expansion (AD > 0)																			
Example E: Message = 0										**Example F: Message < BS**									
Encryption					**Decryption**					**Encryption**					**Decryption**				
Npub	>0	0	1	0	Npub	>0	0	1	0	Npub	>0	0	1	0	Npub	>0	0	1	0
AD	>0	1	1	0	AD	>0	0	1	0	AD	>0	0	1	0	AD	>0	0	1	0
Msg	0	0	1	1	CT	BS	1	1	0	Msg	<BS	1	1	1	CT	BS	1	1	0
					TAG	>0	0	1	1						TAG	>0	0	1	1
Example G: (Message % BS) = 0										**Example H: (Message % BS) > 0**									
Encryption					**Decryption**					**Encryption**					**Decryption**				
Npub	>0	0	1	0	Npub	>0	0	1	0	Npub	0	0	1	0	Npub	>0	0	1	0
AD	>0	0	1	0	AD	>0	0	1	0	AD	0	0	1	0	AD	>0	0	1	0
Msg[0]	MBS	1	0	0	CT[0]	MBS	0	0	0	Msg[0]	MBS	0	0	0	CT[0]	MBS	0	0	0
Msg[1]	0	0	1	1	CT[1]	BS	1	1	0	Msg[1]	<BS	1	1	1	CT[1]	BS	1	1	0
					TAG	16	0	1	1						TAG	16	0	1	1

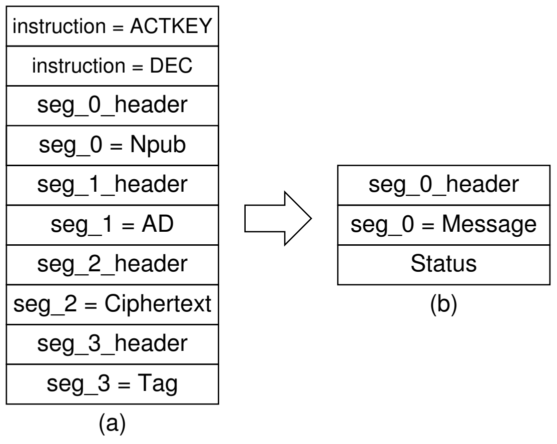


Fig. 10: Format of Public Data Input (PDI) and Data Output (DO) of authenticated decryption operation for ciphers that do not use Nsec: a) PDI, b) DO

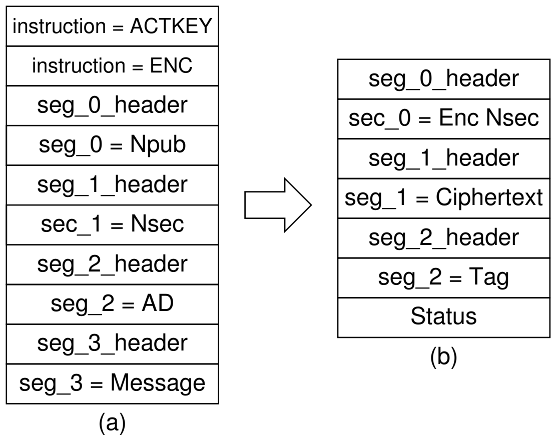


Fig. 11: Format of Public Data Input (PDI) and Data Output (DO) of authenticated decryption operation for ciphers that use Nsec: a) PDI, b) DO

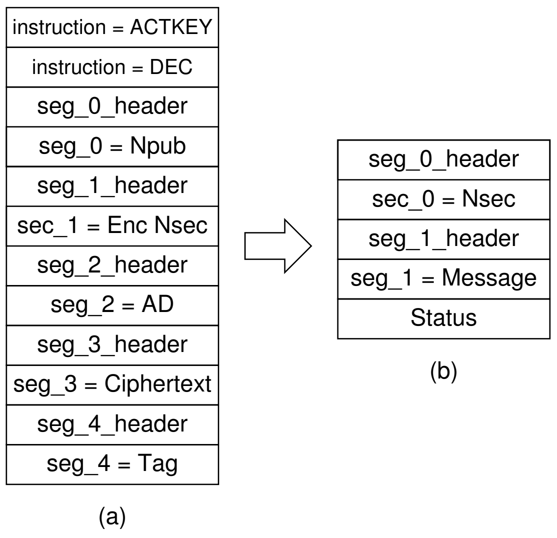


Fig. 12: Format of Public Data Input (PDI) and Data Output (DO) of authenticated decryption operation for ciphers that use Nsec: a) PDI, b) DO

For ciphers that do not use NSec, at the output (`DO` ports), the cryptographic core needs to only output the ciphertext and the tag for encryption, and the message and the status for decryption. In the case that Nsec is used, additional segments should be added as shown in Figures 11 and 12.

For some authenticated ciphers (e.g., AES-CCM), the entire lengths of associated data and message/ciphertext have to be known before the encryption/decryption starts. In order to make it possible, an optional Segment Type, called Length is defined. This segment contains only the total length of associated data concatenated with the total length of message/ciphertext, expressed in bytes.

For authenticated ciphers that utilize *ciphertext expansion*, i.e., the ciphertext size can be larger than the message size, it is recommended to split the last message segment into multiple segments in each case when data size is larger than the algorithm's block size. In particular, the last segment should be always smaller than the authenticated cipher's block size. If it is equal to the block size, then a new segment header containing zero length should be inserted. This special formatting is to ensure that the segment length for the last segment of a particular type can be changed without the need to buffer data for the whole segment.

4 Timing Characteristics

Figures 13 and 14 specify the timing characteristics of the ports `PDI` and `DO`, respectively. Input ports are shown in **blue** and the output ports in **red**. The contents of data buses are read and acknowledged when `*_valid` and its corresponding `*_ready` are both asserted. Data is assumed to be present at the output of the source module when `*_valid` is asserted.

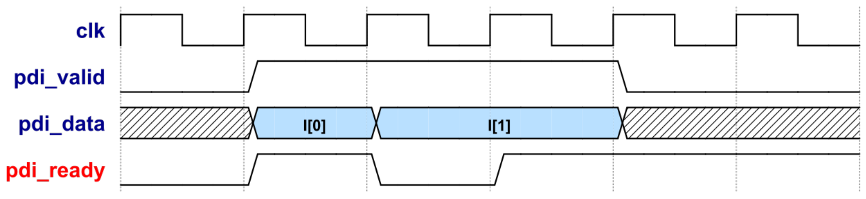


Fig. 13: Example timing diagram for `PDI`

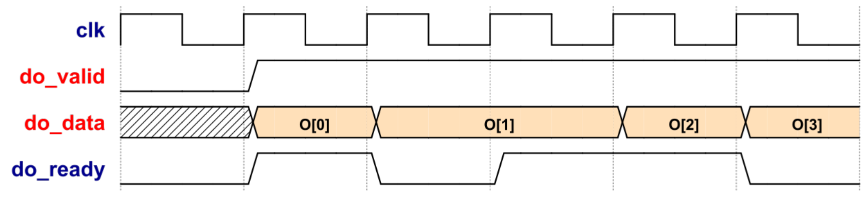
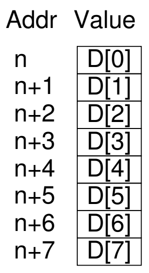
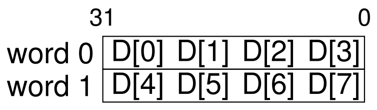


Fig. 14: Example timing diagram for `DO`



(a) Memory



(b) 32-bit word representation

Fig. 15: Data representation

5 Conclusions

We have defined the full specification of the hardware API for authenticated ciphers, suitable for hardware benchmarking of candidates competing in the

CAESAR contest [2] and their comparison with a previous generation of authenticated encryption algorithms, such as AES-GCM and AES-CCM.

Our proposal meets one of the fundamental properties of every properly defined API:

If a given algorithm is implemented independently by two different groups using the same API, one should be able to

– encrypt a message using the first implementation, and
– decrypt it using the second implementation.

To be exact, our assumption is that either

1. both implementations use the same values of the data port widths w and sw, or
2. simple reformatting (word width conversion) of the input to decryption is performed outside of the cipher core (in software or hardware).

On top of that, in both cases, any missing segments, not passed to the output of encryption (namely Npub and AD) need to be properly combined with output from encryption in order to obtain a complete and valid input to decryption.

A similar API, described in [3], has been successfully used to implement and benchmark over a dozen of Round 1 CAESAR candidates, all qualified to Round 2 of the competition.

References

1. ARM. AMBA Specifications. [Online]. Available: http://www.arm.com/products/system-ip/amba-specifications.php
2. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. (2016, January) Cryptographic competitions. [Online]. Available: http://competitions.cr.yp.to/index.html
3. E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, M. U. Sharif, and K. Gaj, “GMU Hardware API for Authenticated Ciphers,” Cryptology ePrint Archive, Report 2015/669, 2015.